Superposition with Lambdas (Technical Report)

Alexander Bentkamp[1(✉)], Jasmin Blanchette[1,2], Sophie Tourret[2], Petar Vukmirović[1], and Uwe Waldmann[2]

[1] Vrije Universiteit Amsterdam, Amsterdam, the Netherlands
{a.bentkamp,j.c.blanchette,p.vukmirovic}@vu.nl
[2] Max-Planck-Institut für Informatik, Saarland Informatics Campus, Saarbrücken, Germany
{jblanche,stourret,uwe}@mpi-inf.mpg.de

Abstract. We designed a superposition calculus for a clausal fragment of extensional polymorphic higher-order logic that includes anonymous functions but excludes Booleans. The inference rules work on $\beta\eta$-equivalence classes of λ-terms and rely on higher-order unification to achieve refutational completeness. We implemented the calculus in the Zipperposition prover and evaluated it on TPTP and Isabelle benchmarks. The results suggest that superposition is a suitable basis for higher-order reasoning.

1 Introduction

Superposition [5] is widely regarded as the calculus par excellence for reasoning about first-order logic with equality. To increase automation in proof assistants and other verification tools based on higher-order formalisms, we propose to generalize superposition to an extensional, polymorphic, clausal version of higher-order logic (also called simple type theory). Our ambition is to achieve a *graceful* extension, which coincides with standard superposition on first-order problems and smoothly scales up to arbitrary higher-order problems.

Bentkamp, Blanchette, Cruanes, and Waldmann [11] recently designed a family of superposition-like calculi for a λ-free fragment of higher-order logic, with currying and applied variables. We adapt their "extensional nonpurifying" calculus to also support λ-expressions (Section 3). Our calculus does not support first-class Booleans; it is conceived as the penultimate milestone towards a superposition calculus for full higher-order logic. If desired, Booleans can be encoded in our logic fragment using an uninterpreted type and uninterpreted "proxy" symbols corresponding to equality, the connectives, and the quantifiers.

Designing a higher-order superposition calculus poses three main challenges:

1. In first-order logic, superposition is parameterized by a ground-total simplification order $\succ$, but such orders do not exist for λ-terms considered equal up to β-conversion. The relations designed for proving termination of higher-order term rewriting systems, such as HORPO [41] and CPO [22], lack many of the desired properties (e.g., transitivity, stability under substitution).

2. Higher-order unification is undecidable and may give rise to an infinite set of incomparable unifiers. For example, the constraint $\mathsf{f}\,(y\,\mathsf{a}) \stackrel{?}{=} y\,(\mathsf{f}\,\mathsf{a})$ admits infinitely many independent solutions of the form $\{y \mapsto \lambda x.\, \mathsf{f}^n\, x\}$.
3. In first-order logic, to rewrite into a term s using an oriented equation $t \approx t'$, it suffices to find a subterm of s that is unifiable with t. In higher-order logic, this is insufficient. Consider superposition from $\mathsf{f}\,\mathsf{c} \approx \mathsf{a}$ into $y\,\mathsf{c} \not\approx y\,\mathsf{b}$. The left-hand sides can obviously be unified by $\{y \mapsto \mathsf{f}\}$, but the more general substitution $\{y \mapsto \lambda x.\, z\,x\,(\mathsf{f}\,x)\}$ also gives rise to a subterm $\mathsf{f}\,\mathsf{c}$ after β-reduction. The corresponding inference generates the clause $z\,\mathsf{c}\,\mathsf{a} \not\approx z\,\mathsf{b}\,(\mathsf{f}\,\mathsf{b})$.

To address the first challenge, we adopt η-short β-normal form to represent $\beta\eta$-equivalence classes of λ-terms. In the spirit of Jouannaud and Rubio's early joint work [40], we state requirements on the term order only for ground terms (i.e., closed monomorphic $\beta\eta$-equivalence classes); the nonground case is connected to the ground case via stability under substitution. Even on ground terms, it is impossible to obtain all desirable properties. We sacrifice compatibility with arguments (the property that $s' \succ s$ implies $s'\,t \succ s\,t$) and compensate for it with an *argument congruence* rule (ArgCong), as in Bentkamp et al. [11].

For the second challenge, we accept that there might be infinitely many incomparable unifiers and enumerate a complete set (including the notorious flex–flex pairs [38]), relying on heuristics to keep the combinatorial explosion under control. The saturation loop must also be adapted to interleave this enumeration with the theorem prover's other activities (Section 6). Despite its reputation for explosiveness, higher-order unification is a conceptual improvement over SK combinators, because it can often *compute* the right unifier.

Consider the conjecture $\exists z.\, \forall x\, y.\, z\,x\,y \approx \mathsf{f}\,y\,x$. After negation, clausification, and skolemization (which are as for first-order logic), the formula becomes $z\,(\mathsf{sk}_{\mathsf{x}}\,z)\,(\mathsf{sk}_{\mathsf{y}}\,z) \not\approx \mathsf{f}\,(\mathsf{sk}_{\mathsf{y}}\,z)\,(\mathsf{sk}_{\mathsf{x}}\,z)$. Higher-order unification quickly computes the unique unifier: $\{z \mapsto \lambda x\, y.\, \mathsf{f}\,y\,x\}$. In contrast, an encoding approach based on combinators, similar to the one implemented in Sledgehammer [50], would blindly enumerate all possible SK terms for z until the right one, $\mathsf{S}\,(\mathsf{K}\,(\mathsf{S}\,\mathsf{f}))\,\mathsf{K}$, is found. Given the definitions $\mathsf{S}\,z\,y\,x \approx z\,x\,(y\,x)$ and $\mathsf{K}\,x\,y \approx x$, the E prover [59] in *auto* mode needs to perform 3756 inferences to derive the empty clause.

For the third challenge, when applying $t \approx t'$ to perform rewriting inside a higher-order term s, the idea is to encode an arbitrary context as a fresh higher-order variable z, unifying s with $z\,t$; the result is $(z\,t')\sigma$, for some unifier σ. This is performed by a dedicated *fluid subterm superposition* rule (FluidSup).

Functional extensionality (the property that $\forall x.\, y\,x \approx z\,x$ implies $y \approx z$) is also considered a challenge for higher-order reasoning [14], although similar difficulties arise with the first-order theories of sets and arrays [35]. Our approach is to add extensionality as an axiom and provide optional rules as optimizations (Section 5). With this axiom, our calculus is refutationally complete with respect to extensional Henkin semantics (Section 4).

We implemented the calculus in the Zipperposition prover [28] (Section 6). Our empirical evaluation includes benchmarks from the TPTP [63] and interactive verification problems exported from Isabelle/HOL [23] (Section 7). The

results appear promising and suggest that an optimized implementation inside a competitive prover such as E [59], SPASS [68], or Vampire [46] would outperform existing higher-order automatic provers.

2 Logic

Our extensional polymorphic clausal higher-order logic is a restriction of full TPTP THF [16] to rank-1 (top-level) polymorphism, as in TH1 [42]. In keeping with standard superposition, we consider only formulas in conjunctive normal form, without explicit quantifiers or Boolean type. We use Henkin semantics [15, 32, 36], as opposed to the standard semantics that serves as the foundation of the HOL systems [34]. By admitting nonstandard models, Henkin semantics is not subject to Gödel's first incompleteness theorem, allowing us to claim the soundness and refutational completeness of our calculus.

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 function type constructor $\to \in \Sigma_{\mathsf{ty}}$ to be present. A type τ, υ 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 use the notation $\bar{a}_n$ or $\bar{a}$ to stand for the tuple $(a_1, \ldots, a_n)$ or product $a_1 \times \cdots \times a_n$, where $n \geq 0$. We write κ for $\kappa()$ and $\tau \to \upsilon$ for $\to(\tau, \upsilon)$. A type declaration is an expression of the form $\Pi\bar{\alpha}_m.\ \tau$ (or simply τ if $m = 0$), where all type variables occurring in τ belong to $\bar{\alpha}_m$.

We fix a set Σ of (function) symbols $\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{f}, \mathsf{g}, \mathsf{h}, \ldots$, with type declarations, written as $\mathsf{f} : \Pi\bar{\alpha}_m.\ \tau$ or f, and a set $\mathcal{V}$ of term variables with associated types, written as $x : \tau$ or x. We require the presence of a symbol $\mathsf{diff} : \Pi\alpha, \beta.\ (\alpha \to \beta) \to (\alpha \to \beta) \to \alpha \in \Sigma$. We use this symbol to express the polymorphic functional extensionality axiom

$$y\ (\mathsf{diff}\langle\alpha, \beta\rangle\ y\ z) \not\approx z\ (\mathsf{diff}\langle\alpha, \beta\rangle\ y\ z) \vee y \approx z \qquad \textsc{Ext}$$

which we will assume to be contained in all problems.

The sets $(\Sigma_{\mathsf{ty}}, \mathcal{V}_{\mathsf{ty}}, \Sigma, \mathcal{V})$ form the signature. The set of *raw λ-terms* is defined inductively as follows. Every $x{:}\tau \in \mathcal{V}$ is a raw λ-term of type τ. If $\mathsf{f} : \Pi\bar{\alpha}_m.\ \tau \in \Sigma$ and $\bar{\upsilon}_m$ is a tuple of types, called *type arguments*, then $\mathsf{f}\langle\bar{\upsilon}_m\rangle$ (or simply f if $m = 0$) is a raw λ-term of type $\tau\{\bar{\alpha}_m \mapsto \bar{\upsilon}_m\}$. If $x : \tau$ and $t : \upsilon$, then the *λ-expression* $\lambda x.\ t$ is a raw λ-term of type $\tau \to \upsilon$. If $s : \tau \to \upsilon$ and $t : \tau$, then the *application* $s\ t$ is a raw λ-term of type υ.

The α-renaming rule is defined as $(\lambda x.\ t) \to_\alpha (\lambda y.\ t\{x \mapsto y\})$, where y does not occur free in t and is not captured by a λ in t. Raw λ-terms form equivalence classes modulo α-renaming, called *λ-terms*. A *proper* subterm of a λ-term t is any subterm of t that is distinct from t itself. A variable occurrence is *free* in a λ-term if it is not bound by a λ-expression. A λ-term is *ground* if it is built

without using type variables and contains no free term variables. Using the spine notation [26], λ-terms can be decomposed in a unique way as a non-application *head* t applied to zero or more arguments: $t\, s_1 \ldots s_n$ or $t\, \bar{s}_n$ (abusing notation).

The β- and η-reduction rules are defined on λ-terms as $(\lambda x.\, t)\, u \to_\beta t\{x \mapsto u\}$ and $(\lambda x.\, t\, x) \to_\eta t$. For β, bound variables in t are renamed if necessary to avoid capture; for η, the variable x must not occur free in t. The λ-terms form equivalence classes modulo $\beta\eta$-reduction, called *$\beta\eta$-equivalence classes* or simply *terms*. When defining operations that need to analyze the structure of terms, we use the η-short β-normal form $t{\downarrow_{\beta\eta}}$, obtained by applying $\to_\beta$ and $\to_\eta$ exhaustively, as a representative of the equivalence class t. Many authors prefer the η-long β-normal form [38, 40, 49], but in a polymorphic setting it has the drawback that instantiating a type variable by a function type can lead to η-expansion. We reserve the letters s, t, u, v for terms and w, x, y, z for variables, and write $: \tau$ to indicate their type.

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 $L_1 \vee \cdots \vee L_n$ is a finite multiset of literals L_j. The empty clause is written as $\bot$.

In general, a substitution $\{\bar{\alpha}_m, \bar{x}_n \mapsto \bar{v}_m, \bar{s}_n\}$, where each x_j has type τ_j and each s_j has type $\tau_j\{\bar{\alpha}_m \mapsto \bar{v}_m\}$, maps m type variables to m types and n term variables to n terms. The letters θ, ρ, σ are reserved for substitutions. Substitutions are lifted to terms and clauses in a capture-avoiding way; for example, $(\lambda x.\, y)\{y \mapsto x\} = (\lambda x'.\, x)$. The composition $\rho\sigma$ applies ρ first: $t\rho\sigma = (t\rho)\sigma$. The notation $\sigma[\bar{x}_n \mapsto \bar{s}_n]$ denotes the substitution that replaces each x_i by s_i and that otherwise coincides with σ. A *complete set of unifiers* on a set X of variables for two terms s and t is a set U of unifiers of s and t such that for every unifier ρ of s and t there exists a member $\sigma \in U$ and a substitution θ such that $x\sigma\theta = x\rho$ for all $x \in X$. We use $\mathrm{CSU}_X(s, t)$ to denote an arbitrary (ideally, minimal) complete set of unifiers on X for s and t. The set X will consist of the free variables of the clauses in which s and t occur and will be left implicit.

2.2 Semantics

A *type interpretation* $\mathcal{I}_{\mathsf{ty}} = (\mathcal{U}, \mathcal{J}_{\mathsf{ty}})$ is defined as follows. The *universe* $\mathcal{U}$ is a nonempty collection of nonempty sets, called *domains*. 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, such that for all domains $\mathcal{D}_1, \mathcal{D}_2 \in \mathcal{U}$, $\mathcal{J}_{\mathsf{ty}}(\to)(\mathcal{D}_1, \mathcal{D}_2)$ is a subset of the function space from $\mathcal{D}_1$ to $\mathcal{D}_2$. The semantics is *standard* if $\mathcal{J}_{\mathsf{ty}}(\to)(\mathcal{D}_1, \mathcal{D}_2)$ is the entire function space for all $\mathcal{D}_1, \mathcal{D}_2$.

A *type valuation* ξ is a function that maps every type variable to a domain. 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 $[\![\tau_1]\!]^\xi_{\mathcal{I}_{\mathsf{ty}}}, \ldots, [\![\tau_n]\!]^\xi_{\mathcal{I}_{\mathsf{ty}}}$. 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 $\mathsf{f} : \Pi\bar{\alpha}_m.\ \tau$ and domain tuple $\bar{\mathcal{D}}_m \in \mathcal{U}^m$ a value $\mathcal{J}(\mathsf{f}, \bar{\mathcal{D}}_m) \in [\![\tau]\!]^{\xi}_{\mathcal{I}_{\mathsf{ty}}}$, where ξ is the type valuation that maps each α_i to D_i.

The comprehension principle states that every function designated by a λ-expression is contained in the corresponding domain. Loosely following Fitting [32, Section 2.4], we initially allow λ-expressions to designate arbitrary elements of the domain, to be able to define the denotation of a term. We impose restrictions afterwards using the notion of a proper interpretation. A *λ-designation function* $\mathcal{L}$ for a type interpretation $\mathcal{I}_{\mathsf{ty}}$ is a function that maps a valuation ξ and a λ-expression of type τ to elements of $[\![\tau]\!]^{\xi}_{\mathcal{I}_{\mathsf{ty}}}$. A type interpretation, an interpretation function, and a λ-designation function form an (extensional) *interpretation* $\mathcal{I} = (\mathcal{I}_{\mathsf{ty}}, \mathcal{J}, \mathcal{L})$. For an interpretation $\mathcal{I}$ and a valuation ξ, the denotation of a term is defined as $[\![x]\!]^{\xi}_{\mathcal{I}} = \xi(x)$, $[\![\mathsf{f}\langle\bar{\tau}_m\rangle]\!]^{\xi}_{\mathcal{I}} = \mathcal{J}(\mathsf{f}, [\![\bar{\tau}_m]\!]^{\xi}_{\mathcal{I}_{\mathsf{ty}}})$, $[\![s\ t]\!]^{\xi}_{\mathcal{I}} = [\![s]\!]^{\xi}_{\mathcal{I}}([\![t]\!]^{\xi}_{\mathcal{I}})$, and $[\![\lambda x.\ t]\!]^{\xi}_{\mathcal{I}} = \mathcal{L}(\xi, \lambda x.\ t)$. For ground terms t, the denotation does not depend on the choice of the valuation ξ, which is why we sometimes write $[\![t]\!]_{\mathcal{I}}$ for $[\![t]\!]^{\xi}_{\mathcal{I}}$.

An interpretation $\mathcal{I}$ is *proper* if $[\![\lambda x.\ t]\!]^{\xi}_{\mathcal{I}}(a) = [\![t]\!]^{\xi[x\mapsto a]}_{\mathcal{I}}$ for all λ-expressions $(\lambda x.t)$ and all valuations ξ. If a type interpretation $\mathcal{I}_{\mathsf{ty}}$ and a interpretation function $\mathcal{J}$ can be extended by a λ-designation function $\mathcal{L}$ to a proper interpretation $(\mathcal{I}_{\mathsf{ty}}, \mathcal{J}, \mathcal{L})$, then this $\mathcal{L}$ is unique [32, Proposition 2.18].

Given an interpretation $\mathcal{I}$ and a valuation ξ, an equation $s \approx t$ is *true* if (and only if) $[\![s]\!]^{\xi}_{\mathcal{I}}$ and $[\![t]\!]^{\xi}_{\mathcal{I}}$ are equal and it is *false* otherwise. 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. A clause set is true if all its clauses are true. A proper interpretation $\mathcal{I}$ is a *model* of a clause set N, written $\mathcal{I} \models N$, if $N \cup \{\textsc{Ext}\}$ is true in $\mathcal{I}$ for all valuations ξ.

2.3 Skolemization

As observed by Bentkamp et al. [11], a problem expressed in higher-order logic must be transformed into clausal normal form before the calculi can be applied. This process works as in the first-order case, except for skolemization. The issue is that skolemization, when performed naively, is unsound for λ-free higher-order logic with a Henkin semantics [52, Section 6]. The crux of the issue is that skolemization introduces new functions that can be used to instantiate variables.

The theoretical part of this report is not affected by this subtlety because the problems are given in clausal form. In the implementation, our solution is to claim soundness only with respect to models that satisfy the axiom of choice, which is the semantics mandated by the TPTP THF format [64]. By contrast, our completeness result holds with respect to arbitrary models as defined in Section 2.2.

2.4 Axiomatization of Booleans

Our clausal logic lacks Booleans, but these can easily be axiomatized as follows. We extend the signature with a nullary type constructor $\mathit{bool} \in \Sigma_{\mathsf{ty}}$ equipped with the proxy symbols $\mathsf{true}, \mathsf{false} : \mathit{bool} \in \Sigma$, $\mathsf{implies} : \mathit{bool} \to \mathit{bool} \to \mathit{bool} \in \Sigma$, $\mathsf{forall} : \Pi\alpha.\ (\alpha \to \mathit{bool}) \to \mathit{bool}$, and $\mathsf{equal} : \Pi\alpha.\ \alpha \to \alpha \to \mathit{bool}$, characterized by the following axioms:

$$\mathsf{true} \not\approx \mathsf{false}$$
$$a \approx \mathsf{true} \vee a \approx \mathsf{false}$$
$$a \approx \mathsf{true} \vee \mathsf{implies}\, a\, b \approx \mathsf{true}$$
$$b \not\approx \mathsf{true} \vee \mathsf{implies}\, a\, b \approx \mathsf{true}$$
$$\mathsf{implies}\, a\, b \not\approx \mathsf{true} \vee a \not\approx \mathsf{true} \vee b \approx \mathsf{true}$$
$$p \not\approx (\lambda x.\, \mathsf{true}) \vee \mathsf{forall}\langle\alpha\rangle\, p \approx \mathsf{true}$$
$$\mathsf{forall}\langle\alpha\rangle\, p \not\approx \mathsf{true} \vee p \approx (\lambda x.\, \mathsf{true})$$
$$x \not\approx y \vee \mathsf{equal}\langle\alpha\rangle\, x\, y \approx \mathsf{true}$$
$$\mathsf{equal}\langle\alpha\rangle\, x\, y \not\approx \mathsf{true} \vee x \approx y$$

Proxies for the $\neg$, $\vee$, and $\wedge$ connectives and the $\exists$ quantifier can be defined in terms of the above—for example:

$$\mathsf{not} \approx (\lambda a.\, \mathsf{implies}\, a\, \mathsf{false})$$
$$\mathsf{exists}\langle\alpha\rangle \approx (\lambda p.\, \mathsf{not}\, (\mathsf{forall}\langle\alpha\rangle\, (\lambda x.\, \mathsf{not}\, (p\, x))))$$

Similarly, Hilbert choice can be axiomatized as the proxy symbol $\mathsf{choice} : \Pi\alpha.\, (\alpha \to \mathit{bool}) \to \alpha$, characterized by the axiom

$$p\, x \not\approx \mathsf{true} \vee p\, (\mathsf{choice}\langle\alpha\rangle\, p) \approx \mathsf{true}$$

Using Hilbert choice, we can provide alternative definitions for $\exists$ and $\forall$:

$$\mathsf{exists}\langle\alpha\rangle \approx (\lambda p.\, p\, (\mathsf{choice}\langle\alpha\rangle\, p))$$
$$\mathsf{forall}\langle\alpha\rangle \approx (\lambda p.\, \mathsf{not}\, (\mathsf{exists}\langle\alpha\rangle\, (\lambda x.\, \mathsf{not}\, (p\, x))))$$

The above axiomatization of Booleans can be used in a prover to support full higher-order logic (with or without Hilbert choice), corresponding to the TPTP THF format variants TH0 (monomorphic) [64] and TH1 (polymorphic) [42]. The prover's clausifier would transform the outer first-order skeleton of a formula into a clause and use the axiomatized Boolean type and operations within the terms. It would also include the proxy axioms in the clausal problem.

3 The Calculus

Our superposition calculus for clausal higher-order logic is inspired by the λ-free *extensional nonpurifying* calculus described by Bentkamp et al. [11]. The text of this and the next section is partly based on that paper and the associated technical report [12] (with Cruanes's permission). The central idea is that superposition inferences are restricted to unapplied subterms occurring in the "first-order outer skeleton" of the superterm—that is, outside λ-expressions and outside the arguments of applied variables. We call these "green subterms." Thus, an equation $\mathsf{g} \approx (\lambda x.\, \mathsf{f}\, x\, x)$ cannot be used directly to rewrite $\mathsf{g}\, \mathsf{a}$ to $\mathsf{f}\, \mathsf{a}\, \mathsf{a}$, because g is applied in $\mathsf{g}\, \mathsf{a}$. A separate inference rule, ArgCong, takes care of deriving $\mathsf{g}\, x \approx \mathsf{f}\, x\, x$, which can be oriented independently of its parent clause and used to rewrite $\mathsf{g}\, \mathsf{a}$ or $\mathsf{f}\, \mathsf{a}\, \mathsf{a}$.

A term (i.e., a $\beta\eta$-equivalence class) t is defined to be a *green subterm* of a term s if either $s = t$ or $s = \mathsf{f}\langle\bar{\tau}\rangle\, \bar{s}$ for some function symbol f, types $\bar{\tau}$ and terms $\bar{s}$, where t is a green subterm of s_i for some i. In $\mathsf{f}\, (\mathsf{g}\, \mathsf{a})\, (y\, \mathsf{b})\, (\lambda x.\, \mathsf{h}\, \mathsf{c}\, (\mathsf{g}\, x))$, the green subterms are a, $\mathsf{g}\, \mathsf{a}$, $y\, \mathsf{b}$, $\lambda x.\, \mathsf{h}\, \mathsf{c}\, (\mathsf{g}\, x)$, and the entire term. The set of green positions of a term is defined analogously to the set of position of a first-order

term: ε is a green position of t, and if $t = \mathsf{f}\langle\bar{\tau}\rangle\ \bar{s}$ and p is a green position of s_i, then $i.p$ is a green position of t. We denote the green subterm of s at the green position p by $s|_p$. We write $t = s\langle u\rangle_p$ to express that u is a green subterm of t at the green position p and call $s\langle\,\rangle_p$ a *green context*; we leave of the subscript p if there are no ambiguities.

Another key notion is that of a "fluid" term. A subterm t of $s[t]$ is called *fluid* if (1) $t{\downarrow_{\beta\eta}}$ is of the form $y\ \bar{u}_n$, where y is not bound in $s[t]$ and $n \geq 1$, or (2) $t{\downarrow_{\beta\eta}}$ is a λ-expression and there exists a substitution σ such that $t\sigma{\downarrow_{\beta\eta}}$ is not a λ-expression (due to η-reduction). A necessary condition for case (2) is that $t{\downarrow_{\beta\eta}}$ contains an applied variable that is not bound in $s[t]$. Intuitively, fluid subterms are terms whose η-short β-normal form can change radically as a result of instantiation. For example, applying the substitution $\{z \mapsto (\lambda x.\, x)\}$ to the fluid term $\lambda x.\, y\ \mathsf{a}\ (z\ x)$ makes the λ-expression vanish: $(\lambda x.\, y\ \mathsf{a}\ x) = y\ \mathsf{a}$ and similarly $(\lambda x.\, \mathsf{f}\ (y\ x)\ x)\{y \mapsto (\lambda x.\, \mathsf{a})\} = (\lambda x.\, \mathsf{f}\ \mathsf{a}\ x) = \mathsf{f}\ \mathsf{a}$.

3.1 Term Order

The calculus is parameterized by a well-founded strict total order $\succ$ on ground terms satisfying the following properties:

- *green subterm property*: $t\langle s\rangle \succeq s$ (i.e., $t\langle s\rangle \succ s$ or $t\langle s\rangle = s$);
- *compatibility with green contexts*: $s' \succ s$ implies $t\langle s'\rangle \succ t\langle s\rangle$.

The literal and clause orders are defined as multiset extensions in the standard way [5]. Two properties that are not required are *compatibility with λ-expressions* ($s' \succ s$ implies $(\lambda x.\, s') \succ (\lambda x.\, s)$) and *compatibility with arguments* ($s' \succ s$ implies $s'\,t \succ s\,t$). The latter would even be inconsistent with totality. To see why, consider the symbols $\mathsf{c} \succ \mathsf{b} \succ \mathsf{a}$ and the terms $\lambda x.\, \mathsf{b}$ and $\lambda x.\, x$. Owing to totality, one of the terms must be larger than the other, say, $(\lambda x.\, \mathsf{b}) \succ (\lambda x.\, x)$. By compatibility with arguments, we get $(\lambda x.\, \mathsf{b})\,\mathsf{c} \succ (\lambda x.\, x)\,\mathsf{c}$, i.e., $\mathsf{b} \succ \mathsf{c}$, a contradiction. A similar line of reasoning applies if $(\lambda x.\, \mathsf{b}) \prec (\lambda x.\, x)$, using a instead of c.

For nonground terms, $\succ$ is extended to a strict partial order so that $t \succ s$ if and only if $t\theta \succ s\theta$ for all grounding substitutions θ. We also introduce a quasiorder $\succsim$ such that $t \succsim s$ if and only if $t\theta \succeq s\theta$ for all grounding substitutions θ, and similarly for literals and clauses. The quasiorder $\succsim$ is more precise than the strict order $\succeq$. For example, we have $x\ \mathsf{b} \not\succeq x\ \mathsf{a}$ because $x\ \mathsf{b} \neq x\ \mathsf{a}$ and $x\ \mathsf{b} \not\succ x\ \mathsf{a}$ by stability under substitutions with $\{x \mapsto \lambda y.\, c\}$. But we can have we can have $x\ \mathsf{b} \succsim x\ \mathsf{a}$.

Our approach to derive a suitable order is to encode η-short β-normal forms into untyped λ-free higher-order terms and apply an order $\succ_{\mathsf{base}}$ such as the λ-free Knuth–Bendix order (KBO) [9], the λ-free lexicographic path order (LPO) [21], or the embedding path order (EPO) [10]. The encoding, denoted by $\lceil\ \rceil$, translates $\lambda x : \tau.\, t$ to $\mathsf{lam}\ \lceil\tau\rceil\ \lceil t\rceil$ and uses De Bruijn symbols db_i to represent bound variables x [25]. It replaces fluid terms t by fresh variables z_t and maps type arguments to term arguments, while erasing any other type information; thus, $\lceil\lambda x : \iota.\, \lambda y : \iota.\, x\rceil = \mathsf{lam}\ \iota\ (\mathsf{lam}\ \iota\ (\mathsf{db}_1\ \iota))$ and $\lceil\mathsf{f}\langle\iota\rangle\ (y\ \mathsf{a})\rceil = \mathsf{f}\ \iota\ z_{y\,\mathsf{a}}$. The use of De

Bruijn indices and the monolithic encoding of fluid terms ensure stability under α-renaming and under substitution.

More precisely, the encoding $[\,]$ is composed of two steps $[\,]_{\mathsf{db}}$ and $[\,]_{\mathsf{lam}}$. Given the higher-order signature $(\Sigma_{\mathsf{ty}}, \mathcal{V}_{\mathsf{ty}}, \Sigma, \mathcal{V})$, $[\,]_{\mathsf{db}}$ encodes terms into the signature $(\Sigma_{\mathsf{ty}}, \mathcal{V}_{\mathsf{ty}}, \Sigma \uplus \{\mathsf{db}_i \mid i \in \mathbb{N}\}, \mathcal{V})$ by replacing each occurrence of a bound variable by db_i, where i is the number of λs in the term structure between the binding λ and the bound variable. For types, we simply define $[\tau]_{\mathsf{db}} = \tau$. Then, $[\,]_{\mathsf{lam}}$ encodes these types and terms further as terms over the untyped λ-free signature $(\Sigma_{\mathsf{ty}} \uplus \Sigma \uplus \{\mathsf{lam}\} \uplus \{\mathsf{db}_i \mid i \in \mathbb{N}\}, \{z_t \mid t \text{ is a term}\})$. The type-to-term version of $[\,]_{\mathsf{lam}}$ is defined as $[\alpha]_{\mathsf{lam}} = \alpha$ and $[\kappa(\bar{\tau})]_{\mathsf{lam}} = \kappa\,[\bar{\tau}]_{\mathsf{lam}}$. The term-to-term version is defined as follows. Here, *fluid-like* means that the λ-term t such that $[t]_{\mathsf{db}} = t_{\mathsf{db}}$ is fluid.

$$[t_{\mathsf{db}}]_{\mathsf{lam}} = \begin{cases} z_{t_{\mathsf{db}}} & \text{if } t_{\mathsf{db}} = x \text{ or if } t_{\mathsf{db}} \text{ is fluid-like} \\ \mathsf{f}\,[\bar{\tau}]_{\mathsf{lam}}\,[\bar{u}_{\mathsf{db}}]_{\mathsf{lam}} & \text{if } t = \mathsf{f}\langle\bar{\tau}\rangle\,\bar{u}_{\mathsf{db}} \\ \mathsf{lam}\,[\tau]_{\mathsf{lam}}\,[u_{\mathsf{db}}]_{\mathsf{lam}} & \text{if } t = (\lambda x : \tau.\, u_{\mathsf{db}}) \text{ and } t_{\mathsf{db}} \text{ is not fluid-like} \end{cases}$$

For any λ-terms t and s, let $[t] = [[t]_{\mathsf{db}}]_{\mathsf{lam}}$ and let $t \succ_{\mathsf{meta}} s$ be $[t] \succ_{\mathsf{base}} [s]$.

Lemma 1. *Let $\succ_{\mathsf{base}}$ be a strict partial order on λ-free terms. If $\succ_{\mathsf{base}}$'s restriction to ground terms enjoys well-foundedness, totality, the green subterm property, and compatibility with green contexts, the restriction to ground terms of the induced order $\succ_{\mathsf{meta}}$ enjoys the same properties.*

Proof. Transitivity and irreflexivity of $\succ_{\mathsf{meta}}$ follow directly from transitivity and irreflexivity of $\succ_{\mathsf{base}}$.

Well-foundedness: If there was an infinite descending chain of ground terms $t_1 \succ_{\mathsf{meta}} t_2 \succ_{\mathsf{meta}} \ldots$, there would be the infinite descending chain $[t_1] \succ_{\mathsf{base}} [t_2] \succ_{\mathsf{base}} \ldots$, in contradiction to the well-foundedness of $\succ_{\mathsf{base}}$ on ground terms.

Totality: By totality of $\succ_{\mathsf{base}}$, for any ground terms t and s, we have $[t] \succ_{\mathsf{base}} [s]$, $[t] \prec_{\mathsf{base}} [s]$, or $[t] = [s]$. In the first two cases it follows $t \succ_{\mathsf{meta}} s$ or $t \prec_{\mathsf{meta}} s$. In the last case, it follows $t = s$ because the encoding $[\,]$ is clearly injective.

Green subterm property: Let s be a term. We show that $s \succeq_{\mathsf{meta}} s|_p$ by induction on p, where $s|_p$ denotes the green subterm at position p. If $p = \varepsilon$, this is trivial. If $p = p'.i$, we have $s \succeq_{\mathsf{meta}} s|_{p'}$ by the induction hypothesis. Hence, it suffices to show that $s|_{p'} \succeq_{\mathsf{meta}} s|_{p'.i}$. From the existence of the position $p'.i$, we know that $s|_{p'}$ must be of the form $s|_{p'} = \mathsf{f}\langle\bar{\tau}\rangle\,\bar{u}$. Then $s|_{p'.i} = u_i$. The encoding yields $[s|_{p'}] = \mathsf{f}\,[\bar{\tau}]\,[\bar{u}]$ and hence $[s|_{p'}] \succeq_{\mathsf{base}} [s|_{p'.i}]$ by the green subterm property of $\succ_{\mathsf{base}}$. It follows that $s|_{p'} \succeq_{\mathsf{meta}} s|_{p'.i}$ and hence $s \succeq_{\mathsf{meta}} s|_p$.

Compatibility with green contexts: By induction on the depth of the context, it suffices to show that $t \succ_{\mathsf{meta}} s$ implies $\mathsf{f}\langle\bar{\tau}\rangle\,\bar{u}\,t\,\bar{v} \succ_{\mathsf{meta}} \mathsf{f}\langle\bar{\tau}\rangle\,\bar{u}\,s\,\bar{v}$ for all t, s, f, $\bar{\tau}$, $\bar{u}$, and $\bar{v}$. This is equivalent to showing that $[t] \succ_{\mathsf{base}} [s]$ implies

$$[\mathsf{f}\langle\bar{\tau}\rangle\,\bar{u}\,t\,\bar{v}] = \mathsf{f}\,[\bar{\tau}]\,[\bar{u}]\,[t]\,[\bar{v}] \succ_{\mathsf{base}} \mathsf{f}\,[\bar{\tau}]\,[\bar{u}]\,[s]\,[\bar{v}] = [\mathsf{f}\langle\bar{\tau}\rangle\,\bar{u}\,s\,\bar{v}]$$

which follows directly from compatibility with green contexts of $\succ_{\mathsf{base}}$.

Lemma 2. *Let $\succ_{\mathsf{base}}$ be a strict partial order on λ-free terms. If $\succ_{\mathsf{base}}$ enjoys stability under substitution (with respect to λ-free terms), $\succ_{\mathsf{meta}}$ enjoys stability under substitution (with respect to $\beta\eta$-equivalence classes).*

Proof. Let t be a λ-term and t_{db} be a $[]_{\mathsf{db}}$-encoded λ-term such that $t_{\mathsf{db}} = [t]_{\mathsf{db}}$. Let θ be a substitution. Then define a λ-free substitution ρ by $z_{t_{\mathsf{db}}}\rho = [t\theta]$.

Then $[t_{\mathsf{db}}]_{\mathsf{lam}}\rho = [t\theta]$ for all t_{db} by induction on the encoding rules: If $t_{\mathsf{db}} = x$ or if t_{db} is fluid-like, $[t_{\mathsf{db}}]_{\mathsf{lam}}\rho = z_{t_{\mathsf{db}}}\rho = [t\theta]$. If $t_{\mathsf{db}} = \mathsf{f}\langle\bar{\tau}\rangle\ \bar{u}_{\mathsf{db}}$ where $\bar{u}_{\mathsf{db}} = [\bar{u}]_{\mathsf{db}}$ for some tuple $\bar{u}$ of λ-terms, then $[t_{\mathsf{db}}]_{\mathsf{lam}}\rho = \mathsf{f}([\bar{\tau}]_{\mathsf{lam}}\rho)([\bar{u}_{\mathsf{db}}]_{\mathsf{lam}}\rho) \overset{\text{IH}}{=} \mathsf{f}\ [\bar{\tau}\theta]\ [\bar{u}\theta] = [\mathsf{f}\ (\bar{\tau}\theta)\ (\bar{u}\theta)] = [t\theta]$. If $t_{\mathsf{db}} = (\lambda x : \tau.\ u_{\mathsf{db}})$ where $u_{\mathsf{db}} = [u]_{\mathsf{db}}$ for some λ-term u and t_{db} is not fluid-like, $[t_{\mathsf{db}}]_{\mathsf{lam}}\rho = \mathsf{lam}\ ([\tau]_{\mathsf{lam}}\rho)\ ([u_{\mathsf{db}}]_{\mathsf{lam}}\rho) \overset{\text{IH}}{=} \mathsf{lam}\ [\tau\theta]\ [u\theta] = [\lambda x : \tau\theta.\ [u\theta]_{\mathsf{db}}]_{\mathsf{lam}} = [\lambda x : \tau\theta.\ u\theta] = [(\lambda x : \tau.\ u)\theta] = [t\theta]$. Here, x is a fresh variable. In the last step, it is crucial that t_{db} is not fluid-like because otherwise an η-reduction might be triggered.

From $[t_{\mathsf{db}}]_{\mathsf{lam}}\rho = [t\theta]$, we derive $[[t]_{\mathsf{db}}]_{\mathsf{lam}}\rho = [t\theta]$ and hence $[t]\rho = [t\theta]$.

Now we assume $t \succ_{\mathsf{meta}} s$ for some terms t and s. Then $[t] \succ_{\mathsf{base}} [s]$ and by stability under substitutions, $[t]\rho \succ_{\mathsf{base}} [s]\rho$. With the above observation, it follows that $[t\theta] \succ_{\mathsf{base}} [s\theta]$ and hence $t\theta \succ_{\mathsf{meta}} s\theta$.

3.2 The Inference Rules

In addition to $\succ$, the calculus is parameterized by a selection function, which maps each clause to a subclause consisting of negative literals. A literal $L\langle y\rangle$ must not be selected if $y\ \bar{u}_n$, with $n > 0$, is a $\succsim$-maximal term of the clause.

A literal L is (*strictly*) *eligible* in C if it is selected in C or if there are no selected literals in C and L is (strictly) maximal in C. A variable is *deep* in a clause C if it occurs inside a λ-expression or inside an argument of an applied variable; these cover all occurrences that may correspond to positions inside λ-expressions after applying a substitution. A variable that is not deep is said to be *shallow*.

We regard positive and negative superposition as two cases of a single rule

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

with the following side conditions:

1. u is not a fluid subterm;
2. u is not a deep variable in C;
3. *variable condition*: if u is a variable y, there must exist a grounding substitution θ such that $t\sigma\theta \succ t'\sigma\theta$ and $C\sigma\theta \prec C''\sigma\theta$, where $C'' = C\{y \mapsto t'\}$;
4. $\sigma \in \mathrm{CSU}(t, u)$;
5. $t\sigma \not\precsim t'\sigma$;
6. $s\langle u\rangle\sigma \not\precsim s'\sigma$;
7. $C\sigma \not\precsim D\sigma$;
8. $(t \approx t')\sigma$ is strictly eligible in $D\sigma$;
9. $([\neg]\, s\langle u\rangle \approx s')\sigma$ is eligible in $C\sigma$, and strictly eligible if it is positive.

There are four main differences with the statement of the standard superposition rule: Contexts $s[\,]$ are replaced by green contexts $s\langle\,\rangle$. The standard condition $u \notin \mathcal{V}$ is generalized by conditions 2 and 3. Most general unifiers are replaced by complete sets of unifiers. And $\not\preceq$ is replaced by the more restrictive $\not\precsim$.

The second rule is a variant of Sup that focuses on fluid subterms occurring in green contexts. Its statement is

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

with the following side conditions, in addition to Sup's conditions 5 to 9:

1. u is either a deep variable in C or a fluid subterm;
2. z is a fresh variable; 3. $\sigma \in \mathrm{CSU}(z\, t, u)$; 4. $z\, t' \neq z\, t$.

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

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

For EqRes: $\sigma \in \mathrm{CSU}(u, u')$ and $(u \not\approx u')\sigma$ is eligible in the premise. For EqFact: $\sigma \in \mathrm{CSU}(u, u')$, $u'\sigma \not\precsim v'\sigma$, $u\sigma \not\precsim v\sigma$, and $(u \approx v)\sigma$ is eligible in the premise.

Argument congruence, a higher-order concern, is embodied by the rule

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

where σ is the most general type substitution that ensures well-typedness of the conclusion. In particular, if the result type of s is not a type variable, σ is the identity substitution; and if the result type is a type variable, it is instantiated with $\bar{\alpha}_n \to \beta$, where $\bar{\alpha}_n$ and β are for fresh type variables, yielding infinitely many conclusions, one for each n. The literal $s\sigma \approx s'\sigma$ must be strictly eligible in $(C' \vee s \approx s')\sigma$, and $\bar{x}_n$ is a nonempty tuple of distinct fresh variables.

Finally, the rules are complemented by the polymorphic functional extensionality axiom Ext.

3.3 Rationale for the Rules

The calculus realizes the following division of labor: Sup and FluidSup are responsible for green subterms, which are outside λs, ArgCong indirectly gives access to the remaining positions outside λs, and the extensionality axiom takes care of subterms occurring inside λs.

Example 3. Prefix subterms such as g in the term $\mathsf{g}\,\mathsf{a}$ are not green subterms and thus cannot be superposed into. ArgCong gives us access to those positions. Consider the clauses

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

An ArgCong inference from the second clause results in $\mathsf{g}\,x \approx \mathsf{f}\,x$. This clause can be used for a Sup inference into the first clause, yielding $\mathsf{f}\,\mathsf{a} \not\approx \mathsf{f}\,\mathsf{a}$ and thus $\bot$ by EqRes.

Example 4. Applied variables give rise to subtle situations with no counterparts in first-order logic. Consider the clauses

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

where $\mathsf{f}\,\mathsf{a} \succ \mathsf{c}$. It is easy to see that the clause set is unsatisfiable, by grounding the second clause with $\theta = \{y \mapsto (\lambda x.\, \mathsf{g}\,(\mathsf{f}\,x))\}$. However, to mimic the superposition inference that can be performed at the ground level, it is necessary to superpose at an imaginary position *below* the applied variable y and yet *above* its argument a, namely, into the subterm $\mathsf{f}\,\mathsf{a}$ of $\mathsf{g}\,(\mathsf{f}\,\mathsf{a}) = (\lambda x.\, \mathsf{g}\,(\mathsf{f}\,x))\,\mathsf{a} = (y\,\mathsf{a})\theta$. FluidSup's z variable effectively transforms $\mathsf{f}\,\mathsf{a} \approx \mathsf{c}$ into $z\,(\mathsf{f}\,\mathsf{a}) \approx z\,\mathsf{c}$, whose left-hand side can be unified with $y\,\mathsf{a}$ by taking $\{y \mapsto (\lambda x.\, z\,(\mathsf{f}\,x))\}$. The resulting clause is $\mathsf{h}\,(z\,(\mathsf{f}\,\mathsf{b}))\,(z\,\mathsf{c}) \not\approx \mathsf{h}\,(\mathsf{g}\,(\mathsf{f}\,\mathsf{b}))\,(\mathsf{g}\,\mathsf{c})$, which has the right form for EqRes.

Example 5. The following clause set has a similar flavor:

$$\mathsf{f}\,\mathsf{a} \approx \mathsf{c} \qquad\qquad \mathsf{f}\,\mathsf{b} \approx \mathsf{d} \qquad\qquad \mathsf{g}\,\mathsf{c} \not\approx y\,\mathsf{a} \vee \mathsf{g}\,\mathsf{d} \not\approx y\,\mathsf{b}$$

EqRes is applicable on either literal of the third clause, but the computed unifier, $\{y \mapsto \lambda x.\, \mathsf{g}\,\mathsf{c}\}$ or $\{y \mapsto \lambda x.\, \mathsf{g}\,\mathsf{d}\}$, is not the right one. Again, we need FluidSup.

Example 6. Third-order clauses in which variables are applied to λ-expressions can be even more stupefying. The clause set

$$\mathsf{f}\,\mathsf{a} \approx \mathsf{c} \qquad\qquad \mathsf{h}\,(y\,(\lambda x.\, \mathsf{g}\,(\mathsf{f}\,x))\,\mathsf{a})\,y \not\approx \mathsf{h}\,(\mathsf{g}\,\mathsf{c})\,(\lambda w\,x.\, w\,x)$$

is unsatisfiable. To see this, apply $\theta = \{y \mapsto (\lambda w\,x.\, w\,x)\}$ to the second clause: $\mathsf{h}\,(\mathsf{g}\,(\mathsf{f}\,\mathsf{a}))\,(\lambda w\,x.\, w\,x) \not\approx \mathsf{h}\,(\mathsf{g}\,\mathsf{c})\,(\lambda w\,x.\, w\,x)$. Let $\mathsf{f}\,\mathsf{a} \succ \mathsf{c}$. A Sup inference is possible between the first clause and this ground instance of the second one. But at the nonground level, the subterm $\mathsf{f}\,\mathsf{a}$ is not clearly localized: $\mathsf{g}\,(\mathsf{f}\,\mathsf{a}) = (\lambda x.\, \mathsf{g}\,(\mathsf{f}\,x))\,\mathsf{a} = (\lambda w\,x.\, w\,x)\,(\lambda x.\, \mathsf{g}\,(\mathsf{f}\,x))\,\mathsf{a} = (y\,(\lambda x.\, \mathsf{g}\,(\mathsf{f}\,x))\,\mathsf{a})\theta$. FluidSup can cope with this. One of the unifiers of $z\,(\mathsf{f}\,\mathsf{a})$ and $y\,(\lambda x.\, \mathsf{g}\,(\mathsf{f}\,x))\,\mathsf{a}$ will be $\{y \mapsto (\lambda w\,x.\, w\,x),\, z \mapsto \mathsf{g}\}$, yielding $\mathsf{h}\,(\mathsf{g}\,\mathsf{c})\,(\lambda w\,x.\, w\,x) \not\approx \mathsf{h}\,(\mathsf{g}\,\mathsf{c})\,(\lambda w\,x.\, w\,x)$.

Example 7. FluidSup is concerned not only with applied variables but also with λ-expressions that, after substitution, may be η-reduced to reveal new applied variables or green subterms. Consider the clause set

$$\mathsf{g}\,\mathsf{a} \approx \mathsf{b} \qquad\qquad \mathsf{h}\,(\lambda y.\, x\,y\,\mathsf{g}\,z) \approx \mathsf{c} \qquad\qquad \mathsf{h}\,(\mathsf{f}\,\mathsf{b}) \not\approx \mathsf{c}$$

Applying $\theta = \{x \mapsto \lambda y'\ w\ z'.\ \mathsf{f}\ (w\ \mathsf{a})\ y',\ z \mapsto \mathsf{b}\}$ to the second clause yields (note that the value z is mapped to does not matter):

$$\begin{aligned} & \mathsf{h}\ (\lambda y.\ (\lambda y'\ w\ z'.\ \mathsf{f}\ (w\ \mathsf{a})\ y')\ y\ \mathsf{g}\ \mathsf{b}) \approx \mathsf{c} \\ = {} & \mathsf{h}\ (\lambda y.\ \mathsf{f}\ (\mathsf{g}\ \mathsf{a})\ y) \approx \mathsf{c} \\ = {} & \mathsf{h}\ (\mathsf{f}\ (\mathsf{g}\ \mathsf{a})) \approx \mathsf{c} \end{aligned}$$

A Sup inference is possible between the first clause of the considered set and this new ground clause, producing the clause $\mathsf{h}\ (\mathsf{f}\ \mathsf{b}) \approx \mathsf{c}$. By also considering λ-expressions, the FluidSup rule is applicable at the nonground level to derive this clause.

The side condition on FluidSup could be tightened by analyzing syntactically whether terms can be subject to η-normalization after variable instantiation. For example, there is no instantiation of y that would make it possible to eliminate the λ in $\lambda x.\ y\ (\mathsf{f}\ x\ \mathsf{a})$.

Because it gives rise to flex–flex pairs (unification constraints where both sides are applied variables), FluidSup can be very prolific. With applied variables on both sides of its maximal literal, the extensionality axiom is another prime source of flex–flex pairs. Flex–flex pairs can also arise in the other rules (Sup, EqRes, and EqFact).

Due to order restrictions and fairness, we cannot postpone solving flex–flex pairs indefinitely. Thus, we cannot use Huet's pre-unification procedure [38] and must instead choose a complete procedure such as Jensen and Pietrzykowski's [39] or Snyder and Gallier's [61]. On the positive side, optional inference rules can efficiently cover many cases where FluidSup or the extensionality axiom would otherwise be needed (Section 5), and heuristics can help keep the explosion under control. Moreover, flex–flex pairs are not always as bad as their reputation; for example, $y\ \mathsf{a}\ \mathsf{b} \stackrel{?}{=} z\ \mathsf{c}\ \mathsf{d}$ admits a most general unifier: $\{y \mapsto (\lambda w\ x.\ y'\ w\ x\ \mathsf{c}\ \mathsf{d}),\ z \mapsto y'\ \mathsf{a}\ \mathsf{b}\}$.

The calculus is a graceful generalization of standard superposition, except for the extensionality axiom. From $\mathsf{g}\ x \approx \mathsf{f}\ x\ x$, the axiom can be used to derive clauses such as $(\lambda x.\ y\ x\ (\mathsf{g}\ x)) \approx (\lambda x.\ y\ x\ (\mathsf{f}\ x\ x))$, which are useless if the problem is first-order. This could be avoided if we could find a way to make the positive literal, $y \approx z$, maximal, or to select it without losing refutational completeness. This literal interacts only with green subterms of function type, which cannot arise in first-order clauses.

3.4 Redundancy Criterion

A redundant (or composite) clause is usually defined as a clause whose ground instances are entailed by smaller ($\prec$) ground instances of existing clauses. This would be too strong for our calculus; for example, it would make ArgCong inferences redundant. Our solution is to base the redundancy criterion on a weaker ground logic in which argument congruence and extensionality are not guaranteed to hold.

The weaker logic is defined via an encoding $\lfloor\ \rfloor$ of ground λ-terms into 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.* The $\lfloor\ \rfloor$ 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 $\lfloor \mathsf{g}\langle\iota\rangle \rfloor = \mathsf{g}^\iota_0$. In addition, it conceals λs by replacing them with fresh symbols. These measures effectively disable argument congruence and extensionality. For example, the clause sets $\{\mathsf{g}_0 \approx \mathsf{f}_0,\ \mathsf{g}_1(\mathsf{a}_0) \not\approx \mathsf{f}_1(\mathsf{a}_0)\}$ and $\{\mathsf{b}_0 \approx \mathsf{a}_0,\ \mathsf{c}_0 \not\approx \mathsf{d}_0\}$ are satisfiable, even though $\{\mathsf{g} \approx \mathsf{f},\ \mathsf{g}\ \mathsf{a} \not\approx \mathsf{f}\ \mathsf{a}\}$ and $\{\mathsf{b} \approx \mathsf{a},\ (\lambda x.\ \mathsf{b}) \not\approx (\lambda x.\ \mathsf{a})\}$ are unsatisfiable.

Given a ground higher-order signature $(\Sigma_{\mathsf{ty}}, \{\}, \Sigma, \{\})$, we define a first-order signature $(\Sigma_{\mathsf{ty}}, \{\}, \Sigma^{\downarrow}, \{\})$ as follows. The type constructors Σ_{ty} are the same in both signatures, but $\to$ is uninterpreted in first-order logic. For each ground instance $\mathsf{f}\langle\bar{\upsilon}\rangle : \tau_1 \to \cdots \to \tau_n \to \tau$ of a symbol $\mathsf{f} \in \Sigma$, we introduce a first-order symbol $\mathsf{f}^{\bar{\upsilon}}_j \in \Sigma^{\downarrow}$ with argument types $\bar{\tau}_j$ and result type $\tau_{j+1} \to \cdots \to \tau_n \to \tau$, for each j. Moreover, for each ground term $\lambda x.\, t$, we introduce a symbol $\lfloor \lambda x.\, t \rfloor \in \Sigma^{\downarrow}$ of the same type.

The $\lfloor\ \rfloor$ encoding is defined on ground η-short β-normal forms so that $\lambda x.\, t$ is mapped to the symbol $\lfloor \lambda x.\, t \rfloor$ and $\lfloor \mathsf{f}\langle\bar{\upsilon}\rangle\ \bar{s}_j \rfloor = \mathsf{f}^{\bar{\upsilon}}_j(\lfloor \bar{s}_j \rfloor)$ recursively. The encoding is extended to literals and clauses elementwise. Using the inverse mapping $\lceil\ \rceil$, the order $\succ$ can be transferred to the floor level by defining $t \succ s$ as $\lceil t \rceil \succ \lceil s \rceil$. 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.

More precisely, the $\lfloor\ \rfloor$ mapping is extended to ground literals and ground clauses as follows:

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

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

$$\begin{aligned} \lceil \lfloor \lambda x.\, s \rfloor \rceil &= \lambda x.\, s \\ \lceil \mathsf{f}^{\bar{\tau}}_i\ \bar{t}_i \rceil &= \mathsf{f}\langle\bar{\tau}\rangle\ \lceil \bar{t}_i \rceil \\ \lceil s \approx t \rceil &= \lceil s \rceil \approx \lceil t \rceil \\ \lceil s \not\approx t \rceil &= \lceil s \rceil \not\approx \lceil t \rceil \\ \lceil L_1 \vee \cdots \vee L_n \rceil &= \lceil L_1 \rceil \vee \cdots \vee \lceil L_n \rceil \end{aligned}$$

A crucial property of $\lfloor\ \rfloor$ is that green subterms in the ceiling logic correspond to subterms in the floor logic. Thus, the subterms considered by Sup and FluidSup coincide with the the subterms exposed to the redundancy criterion.

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

$$\mathsf{f}\langle\lceil\bar{\tau}\rceil\rangle\ \lceil u_1\rceil \ldots \lceil u_{i-1}\rceil\ \lceil u_i\rceil\langle\lceil s\rceil\rangle_{p'}\ \lceil u_{i+1}\rceil \ldots \lceil u_n\rceil$$

It follows that $\lceil t[s]_p \rceil = \lceil t\rceil\langle\lceil s\rceil\rangle_p$. □

Lemma 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 green 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 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 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 employs the entailment relation $\models$ on ground clauses. We define redundancy of ceiling clauses in the same way, but using the floor logic's $\models$ on the $\lfloor\ \rfloor$-encoded clauses. This definition gracefully generalizes the standard first-order notion of redundancy. For Sup, FluidSup, EqRes, and EqFact, 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, whose conclusion is too large, we use redundancy of clauses.

More precisely we define redundancy as follows: As usual, an inference is a ground instance of an inference I if it is the result of applying a grounding substitution to I and if it is itself an inference (satisfying all the conditions of the inference rules).

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, a ground inference with conclusion E and right (or only) premise C is *redundant with respect to a set of ground clauses* N if one of its premises is redundant with respect to N, or if $\lfloor E\rfloor$ is

entailed by clauses in $\lfloor N \rfloor$ that are smaller than $\lfloor C \rfloor$. A nonground inference is *redundant with respect to a clause set* N if all its ground instances are redundant with respect to $\mathcal{G}_\Sigma(N)$.

An ArgCong 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.

The saturation procedures of superposition-based provers aggressively delete clauses that are strictly subsumed by other clauses. A clause C *subsumes* D if there exists a substitution σ such that $C\sigma \subseteq D$. A clause C *strictly subsumes* D if C subsumes D but D does not subsume C. For example, $x \approx \mathsf{c}$ strictly subsumes both $\mathsf{a} \approx \mathsf{c}$ and $\mathsf{b} \not\approx \mathsf{a} \vee x \approx \mathsf{c}$. The proof of refutational completeness of resolution and superposition provers relies on the well-foundedness of the strict subsumption relation [58, Section 7]. Unfortunately, this property does not hold for higher-order logic, where $\mathsf{f}\ x\ x \approx \mathsf{c}$ is strictly subsumed by $\mathsf{f}\ (x\ \mathsf{a})\ (x\ \mathsf{b}) \approx \mathsf{c}$, which is strictly subsumed by $\mathsf{f}\ (x\ \mathsf{a}\ \mathsf{a})\ (x\ \mathsf{b}\ \mathsf{b}') \approx \mathsf{c}$, and so on. Subsumption must be restricted to prevent such infinite chains—for example, by requiring that the subsumer is syntactically smaller than or of the same size as the subsumee.

4 Refutational Completeness

Besides soundness, the most important property of the higher-order superposition calculus introduced in Section 3 is refutational completeness. The proof is adapted from Bentkamp et al. [11]. We use the structure and notation of Waldmann's version [67], which is essentially the completeness proof for superposition without constraints [5] presented in the style of the proof for superposition with constraints by Nieuwenhuis and Rubio [53].

4.1 Outline of the Proof

For the rest of Section 4, let $N \not\ni \bot$ be a higher-order clause set saturated up to redundancy with respect to the inference rules and that contains the extensionality axiom. To avoid empty Herbrand domains, we assume that the signature Σ contains a polymorphic symbol of type $\Pi\alpha.\ \alpha$. The proof proceeds in two steps:

1. Construct a model of the first-order grounded clause set $\lfloor \mathcal{G}_\Sigma(N) \rfloor$, where $\lfloor\ \rfloor$ is the encoding of ground terms used to define redundancy (Section 3.4).

2. Lift this first-order model to a proper higher-order interpretation and show that it is a model of $\mathcal{G}_\Sigma(N)$ and hence of N.

The first step follows the same general idea as the completeness proof for standard superposition [5, 54, 67]. We construct a term rewriting system R_∞ and use it to define a candidate interpretation that equates all terms that share the same normal form with respect to R_∞. At this level, expressions $\lambda x.\, t$ are regarded as uninterpreted symbols $\lfloor \lambda x.\, t \rfloor$.

As in the standard proof, it is the set N, and not its grounding $\mathcal{G}_\Sigma(N)$, that is saturated. We must show that there exist nonground inferences corresponding to all necessary ground Sup, EqRes, and EqFact inferences. We face two specifically higher-order difficulties. First, in standard superposition, we can avoid Sup inferences into variables x by exploiting the order's compatibility with contexts: If $t' \prec t$, we have $C\{x \mapsto t'\} \prec C\{x \mapsto t\}$, which allows us to invoke the induction hypothesis at a key point in the argument to establish the truth of $C\{x \mapsto t'\}$. This technique fails for higher-order variables x that occur applied in C, because the order lacks compatibility with arguments. Hence, our Sup rule must perform some inferences into variables. The other difficulty also concerns applied variables. We must show that any necessary ground Sup inference into a position corresponding to a fluid term or a deep variable on the nonground level can be lifted to a FluidSup inference. This involves showing that the z variable in FluidSup can represent arbitrary contexts around a term t.

For the first-order model construction, $\beta\eta$-normalization is the proverbial dog that did not bark. At the ground level, the rules Sup, EqRes, and EqFact preserve η-short β-normal form, and so does first-order term rewriting. Thus, we can completely ignore $\to_\beta$ and $\to_\eta$. At the nonground level, β- and η-reduction can arise only through instantiation. This poses no difficulties thanks to the order's stability under substitution.

The second step of the completeness proof consists of constructing a higher-order interpretation and proving that it is a model of $\mathcal{G}_\Sigma(N)$, and hence of N. The difficulty is to show that the symbols representing λ-expressions behave like the λ-expressions they represent. This step relies on saturation with respect to the ArgCong rule—which connects a λ-expression with its value when applied to an argument x—and on the presence of the extensionality axiom.

4.2 Candidate Interpretation

The construction of the candidate interpretation is as in the first-order proof, except that it is based on $\lfloor \mathcal{G}_\Sigma(N) \rfloor$. We first define sets of rewrite rules E_C and R_C for all $C \in \lfloor \mathcal{G}_\Sigma(N) \rfloor$ by induction on the clause order. Assume that E_D has already been defined for all $D \in \lfloor \mathcal{G}_\Sigma(N) \rfloor$ with $D \prec C$. Then $R_C = \bigcup_{D \prec C} E_D$. Let $E_C = \{s \to t\}$ if the following conditions are met:

(a) $C = C' \vee s \approx t$;
(b) $s \approx t$ is strictly maximal in C;
(c) $s \succ t$;
(d) C is false in R_C;
(e) C' is false in $R_C \cup \{s \to t\}$; and
(f) s is irreducible with respect to R_C.

Then C is *productive*. Otherwise, $E_C = \emptyset$. Finally, $R_\infty = \bigcup_D E_D$.

A rewrite system R defines an interpretation $\mathcal{T}_\Sigma^\emptyset/R$ such that for every *ground* equation $s \approx t$, we have $\mathcal{T}_\Sigma^\emptyset/R \models s \approx t$ if and only if $s \leftrightarrow_R^* t$. Formally, $\mathcal{T}_\Sigma^\emptyset/R$ denotes the monomorphic first-order interpretation whose universes $\mathcal{U}_\tau$ consist of the R-equivalence classes of the terms of type τ over the signature Σ and the empty set of variables, denoted as $T_\Sigma(\emptyset)/R$ by Waldmann [67] and as I by Bachmair and Ganzinger [5]. The interpretation $\mathcal{T}_\Sigma^\emptyset/R$ is term-generated—that is, for every element a of the universe of this interpretation and for any valuation ξ, there exists a ground term t such that $[\![t]\!]^\xi_{\mathcal{T}_\Sigma^\emptyset/R} = a$. To lighten notation, we will write R to refer to both the term rewriting system R and the interpretation $\mathcal{T}_\Sigma^\emptyset/R$.

The following properties of the candidate interpretations can be shown exactly as in Waldmann's version of the first-order proof [67].

Lemma 10. *The rewrite systems R_C and R_∞ are confluent and terminating.*

Lemma 11. *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 12. *If $D = D' \vee u \approx v$ is productive, then D' is false and D is true in R_∞ and R_C for all $C \succ D$.*

4.3 Lifting Lemmas

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 [6, p. 45], let S be the selection function with respect to which N is saturated up to redundancy. We introduce another selection function S_N such that each clause $C \in \mathcal{G}_\Sigma(N)$ is a ground instance of a clause $D \in N$ such that the selections $S(D)$ and $S_N(C)$ coincide. In the remainder of the proof, we adhere to the following convention:

Convention 13. When we speak about selected literals of clauses in N, it is with respect to the selection function S. When we speak about selected literals of clauses in $\mathcal{G}_\Sigma(N)$, it is with respect to the selection function S_N.

Lemma 14 (Lifting of EqRes and EqFact inferences). *Let $C\theta \in \mathcal{G}_\Sigma(N)$, where θ is a substitution and the selected literals in $C \in N$ correspond to those in $C\theta$ (using S for C and S_N for $C\theta$ as mentioned in Convention 13). Then every* EqRes *or* EqFact *inference from $C\theta$ is a ground instance of an inference from C.*

Proof. EqRes: If there is an EqRes inference from $C\theta$, then $C\theta$ is of the form $C\theta = C'\theta \vee s\theta \not\approx s'\theta$ where $C = C' \vee s \not\approx s'$, and $s\theta \not\approx s'\theta$ is selected or no

literal of $C\theta$ is selected and $s\theta \not\approx s'\theta$ is maximal. The ground inference is

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

where $s\theta$ and $s'\theta$ are unifiable and ground; hence $s\theta = s'\theta$. Since $s\theta \not\approx s'\theta$ is maximal (and nothing is selected) or selected in $C\theta$, We know that $s \not\approx s'$ is maximal (and nothing is selected) or selected in C. Since s and s' are unifiable, there exists some idempotent $\sigma \in \mathrm{CSU}(s, s')$ such that $\sigma\rho = \theta$ for some substitution ρ. Hence there is an inference

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

By idempotence of σ, we have $C'\sigma\theta = C'\theta$. Thus the ground inference is the θ-ground instance of the nonground inference.

EqFact: Analogously, if there is an EqFact inference from $C\theta$, then $C\theta$ is of the form $C\theta = C'\theta \vee s'\theta \approx t'\theta \vee s\theta \approx t\theta$ where $s\theta \approx t\theta$ is maximal, no literal is selected in $C\theta$, $s\theta \not\prec t\theta$, and $C = C' \vee s' \approx t' \vee s \approx t$. The ground inference is

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

where $s\theta$ and $s'\theta$ are unifiable and ground; hence $s\theta = s'\theta$. Since $s\theta \approx t\theta$ is maximal in $C\theta$, nothing is selected in $C\theta$, and $s\theta \not\prec t\theta$, we know that $s \approx t$ is maximal in C, nothing is selected in C, and $s \not\prec t$. Since s and s' are unifiable, there exists some idempotent $\sigma \in \mathrm{CSU}(s, s')$ such that $\sigma\rho = \theta$ for some substitution ρ. Hence there is an inference

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

By idempotence of σ, 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. □

To lift Superposition inferences, we need the following auxiliary lemma:

Lemma 15 (Instances of green subterms). *Let s be a λ-term in η-short β-normal form, let σ be a substitution, let p be a green position of both s and $s\sigma{\downarrow_{\beta\eta}}$. Then $(s|_p)\sigma{\downarrow_{\beta\eta}} = (s\sigma{\downarrow_{\beta\eta}})|_p$.*

Proof. By induction on p.

If $p = \varepsilon$, then $(s|_p)\sigma{\downarrow_{\beta\eta}} = s\sigma{\downarrow_{\beta\eta}} = (s\sigma{\downarrow_{\beta\eta}})|_p$.

If $p = i.p'$, then $s = \mathsf{f}\langle\bar{\tau}\rangle\, s_1 \ldots s_n$ and $s\sigma = \mathsf{f}\langle\bar{\tau}\sigma\rangle\, (s_1\sigma) \ldots (s_n\sigma)$, where $1 \le i \le n$ and p' is a green position of s_i. Clearly, $\beta\eta$-normalization steps of $s\sigma$ can take place only in proper subterms. So $s\sigma{\downarrow_{\beta\eta}} = \mathsf{f}\langle\bar{\tau}\sigma\rangle\, (s_1\sigma{\downarrow_{\beta\eta}}) \ldots (s_n\sigma{\downarrow_{\beta\eta}})$.

Since $p = i.p'$ is a green position of $s\sigma{\downarrow_{\beta\eta}}$, p' must be a green position of $(s_i\sigma){\downarrow_{\beta\eta}}$. By induction, $(s_i|_{p'})\sigma{\downarrow_{\beta\eta}} = (s_i\sigma{\downarrow_{\beta\eta}})|_{p'}$, therefore

$$(s|_p)\sigma{\downarrow_{\beta\eta}} = (s|_{i.p'})\sigma{\downarrow_{\beta\eta}} = (s_i|_{p'})\sigma{\downarrow_{\beta\eta}} = (s_i\sigma{\downarrow_{\beta\eta}})|_{p'} = (s\sigma{\downarrow_{\beta\eta}})|_p.$$

□

Definition 16. Let I be a ground Sup inference from $D\theta = D'\theta \vee t\theta \approx t'\theta$ and $C\theta = C'\theta \vee [\neg]\, s\theta\langle t\theta\rangle_p \approx s'\theta$, where we assume that s, t, $s\theta$ and $t\theta$ are represented by λ-terms in η-short β-normal form. Let p' be the longest prefix of p that is a green position of s.[1] We call the ground inference *liftable*, if $s|_{p'}$ is a deep variable in C, or if $p = p'$ and the variable condition holds for D and C, or if $p \neq p'$ and $s|_{p'}$ is not a variable.

Lemma 17 (Lifting of Sup inferences). *Let $D\theta, C\theta \in \mathcal{G}_\Sigma(N)$, where θ is a substitution and the selected literals in $D \in N$ and $C \in N$ correspond to those in $D\theta$ and $C\theta$ (possibly using different selection functions for $\mathcal{G}_\Sigma(N)$ and N). Then every liftable* Sup *inference between $D\theta$ and $C\theta$ is a ground instance of a* Sup *or* FluidSup *inference from D and C.*

Proof. We may assume that $D = D' \vee t \approx t'$ and $C = C' \vee [\neg]\, s \approx s'$, and that the inference has the form

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

where $t\theta \approx t'\theta$ is strictly eligible, $[\neg]\, s\theta \approx s'\theta$ is strictly eligible if positive and eligible if negative, $D\theta \not\asymp C\theta$, $t\theta \not\asymp t'\theta$, and $s\theta \not\asymp s'\theta$. Furthermore, we assume that s, t, $s\theta$ and $t\theta$ are represented by λ-terms in η-short β-normal form.

Let p' be the longest prefix of p that is a green position of s; let $u = s|_{p'}$. By Lemma 15, $u\theta$ agrees with $s\theta|_{p'}$ (considering both as terms rather than as λ-terms).

Case 1: $p = p'$; u is not fluid; and if u is a variable, it is not deep in C. Then $u\theta = s\theta|_{p'} = s\theta|_p = t\theta$. As θ is a unifier of u and t, there exists some idempotent $\sigma \in \mathrm{CSU}(t, u)$ such that $\sigma\rho = \theta$ for some substitution ρ. The inference conditions can be lifted: (Strict) eligibility of $t\theta \approx t'\theta$ and $[\neg]\, s\theta \approx s'\theta$ implies (strict) eligibility of $t \approx t'$ and $[\neg]\, s \approx s'$; $D\theta \not\asymp C\theta$ implies $D \not\asymp C$; $t\theta \not\asymp t'\theta$ implies $t \not\asymp t'$; and $s\theta \not\asymp s'\theta$ implies $s \not\asymp s'$. Moreover, by liftability, the variable condition holds for D and C. So there is a Superposition inference

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

where $\sigma = \mathrm{mgu}(t, u)$. Since σ is idempotent, $\theta = \sigma\rho = \sigma\sigma\rho = \sigma\theta$, so $(D' \vee C' \vee [\neg]\, s\langle t'\rangle_p \approx s')\sigma\theta = D'\theta \vee C'\theta \vee [\neg]\, s\theta\langle t'\theta\rangle_p \approx s'\theta$, which implies that the ground inference is the θ-ground instance of the nonground inference.

[1] Since ε is a green position of s, the longest prefix always exists.

Case 2: (a) $p \neq p'$; or (b) u is fluid; or (c) u is a deep variable in C. We will first show that u is either fluid or a deep variable in C, even in case (a). Suppose it is neither. By liftability, u cannot be a shallow variable either. Moreover, since u is not fluid, it cannot have the form $y\,\bar{u}_n$ for a variable y and $n \geq 1$. If u were an application $\mathsf{f}\langle\bar{\tau}\rangle\, s_1 \ldots s_n$ with $n \geq 0$, $u\theta$ would have the form $\mathsf{f}\langle\bar{\tau}\theta\rangle\, s_1\theta \ldots s_n\theta$, but then there is some $1 \leq i \leq n$ such that $p'.i$ is a prefix of p and $s|_{p'.i}$ is a green subterm of s, contradicting the fact that p' is the longest prefix of p with this property. So u must be a λ-expression, but since $t\theta$ is a proper green subterm of $u\theta$, $u\theta$ cannot be a λ-expression, contradicting the assumption that u is neither fluid nor a deep variable in C.

Let $p = p'.p''$. Let z be a new variable. Define a substitution θ' that maps z to $\lambda y.(s\theta|_{p'})\langle y\rangle_{p''}$ and any other variable w to $w\theta$. Clearly, $(z\,t)\theta' = (s\theta|_{p'})\langle t\theta\rangle_{p''} = s\theta|_{p'} = u\theta = u\theta'$. As θ' is a unifier of u and $z\,t$, there exists some idempotent $\sigma \in \mathrm{CSU}(z\,t, u)$ such that $\sigma\rho = \theta'$ for some substitution ρ. As in Case 1, (strict) eligibility of the ground literals implies (strict) eligibility of the nonground literals. Moreover, by construction of θ', $t\theta' = t\theta \neq t'\theta = t'\theta'$ implies $(z\,t)\theta' \neq (z\,t')\theta'$, and thus $(z\,t)\sigma \neq (z\,t')\sigma$. So there is an inference

$$\frac{D' \vee t \approx t' \qquad C' \vee [\neg]\, s\langle u\rangle_{p'} \approx s'}{(D' \vee C' \vee [\neg]\, s\langle z\,t'\rangle_{p'} \approx s')\sigma}\,\textsc{FluidSup}$$

where $\sigma \in \mathrm{CSU}(z\,t, u)$. Since σ is idempotent, $\theta' = \sigma\rho = \sigma\sigma\rho = \sigma\theta'$, so $(D' \vee C' \vee [\neg]\, s\langle z\,t'\rangle_{p'} \approx s')\sigma\theta' = D'\theta' \vee C'\theta' \vee [\neg]\, s\theta'\langle z\theta'\; t'\theta'\rangle_{p'} \approx s'\theta' = D'\theta' \vee C'\theta' \vee [\neg]\, s\theta\langle s\theta|_{p'}\langle t'\theta\rangle_{p''}\rangle_{p'} \approx s'\theta' = D'\theta \vee C'\theta \vee [\neg]\, s\theta\langle t'\theta\rangle_{p} \approx s'\theta$, which implies that the ground inference is the θ'-ground instance of the nonground inference. □

4.4 Construction of the First-Order Model

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

The next two lemmas are slightly adapted from the first-order proof. The justification for Lemma 18, about liftable inferences, is essentially as in the first-order case. The proof of Lemma 19, about nonliftable inferences, is more problematic. The standard argument involves defining a substitution θ' such that $\lfloor C\theta' \rfloor$ and $\lfloor C\theta \rfloor$ 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 condition and the ArgCong rule come into play.

Lemma 18. *Let $D\theta, C\theta \in \mathcal{G}_\Sigma(N)$, where the selected literals in $D \in N$ and in $C \in N$ correspond to those in $D\theta$ and $C\theta$, respectively. We consider a liftable* Sup *inference from $D\theta$ and $C\theta$ or an* EqRes *or* EqFact *inference from $C\theta$. Let E be*

the conclusion. Assume that $C\theta$ and $D\theta$ are nonredundant with respect to $\mathcal{G}_\Sigma(N)$. Then $\lfloor E \rfloor$ is entailed by clauses from $\lfloor \mathcal{G}_\Sigma(N) \rfloor$ that are smaller than $\lfloor C\theta \rfloor$.

Proof. We have a liftable Sup inference from $D\theta$ and $C\theta$ or an EqRes or EqFact inference from $C\theta$. As shown in the lifting lemmas (Lemmas 14 and 17) this inference is an instance of an inference from C (or from D and C for Sup inferences). Let $\tilde{E}$ be its conclusion, then $\tilde{E}\theta = E$. Since N is saturated up to redundancy, this inference is redundant with respect to N and hence the θ-ground instance of this inference is redundant with respect to $\mathcal{G}_\Sigma(N)$. By definition of inference redundancy, since none of $D\theta$ and $C\theta$ are redundant with respect to $\mathcal{G}_\Sigma(N)$, $\lfloor E \rfloor$ is entailed by clauses from $\lfloor \mathcal{G}_\Sigma(N) \rfloor$ that are smaller than $\lfloor C\theta \rfloor$. □

Lemma 19. *Let $D\theta, C\theta \in \mathcal{G}_\Sigma(N)$, where the selected literals in $D \in N$ and in $C \in N$ correspond to those in $D\theta$ and $C\theta$, respectively. We consider a nonliftable* Sup *inference from $D\theta$ and $C\theta$. Assume that $C\theta$ and $D\theta$ are nonredundant with respect to $\mathcal{G}_\Sigma(N)$. Let $D'\theta$ be the clause $D\theta$ without the literal involved in the inference. Then $\lfloor C\theta \rfloor$ is entailed by $\neg\lfloor D'\theta \rfloor$ and the clauses in $\lfloor \mathcal{G}_\Sigma(N) \rfloor$ that are smaller than $\lfloor C\theta \rfloor$.*

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

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

Let p the position in $s\theta$ where the inference takes place and p' the longest prefix of p that is a green subterm of s. Since the inference is not liftable, $s|_{p'}$ is not a deep variable; if $p = p'$, the variable condition does not hold for D and C; and if $p \neq p'$, $s|_{p'}$ is a variable. That means the inference is either at the position of an unapplied variable shallow in C, for which the variable condition does not hold, or below an unapplied variable shallow in C.

Case 1: We assume that it is a superposition below a variable shallow in C, say, x. Let $t\theta \approx t'\theta$ be the strictly maximal literal of $D\theta$, where $t\theta \succ t'\theta$. Then $t\theta$ is an argument subterm of $x\theta$ and hence an argument subterm of $x\theta\, \bar{w}$ for any arguments $\bar{w}$. Let v be the term that we obtain by replacing $t\theta$ by $t'\theta$ in $x\theta$ at the relevant position. It follows from the definition of R that $R \models \lfloor t\theta \approx t'\theta \rfloor$ and by congruence, $R \models \lfloor x\theta\, \bar{w} \approx v\, \bar{w} \rfloor$ for any arguments $\bar{w}$. Hence, $R \models \lfloor C\theta \rfloor$ if and only if $R \models \lfloor C\{x \mapsto v\}\theta \rfloor$ by congruence. Here, it is crucial that the variable is shallow in C because congruence does not hold on the floor level below λ-expressions. 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 green contexts. Therefore, by the assumption about R, we have $R \models \lfloor C\{x \mapsto v\}\theta \rfloor$ and hence $R \models \lfloor C\theta \rfloor$.

Case 2: The superposition takes place at the position of a variable x shallow in C, but the variable condition does not hold. Since the variable condition does not hold, we know that $C\theta \succeq C''\theta$, where $C'' = C\{x \mapsto t'\}$. We cannot have $C\theta = C''\theta$ because $x\theta = t\theta \neq t'\theta$ and x occurs in C. Hence, we have $C\theta \succ C''\theta$.

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

By saturation up to redundancy, for any type-correct m-tuple of fresh variables $\bar{z}$, we can use an ArgCong inference with premise D (if $m > 0$) or D itself (if $m = 0$) to show that up to variable renaming $D' \vee t\ \bar{z} \approx t'\ \bar{z}$ is in $N \cup \mathrm{Red}(N)$. Hence, $D'\theta \vee t\theta\ \bar{u} \approx t'\theta\ \bar{u}$ is in $\mathcal{G}_\Sigma(N \cup \mathrm{Red}(N))$ for any type-correct ground arguments $\bar{u}$.

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

$$R \models \lfloor t\theta\ \bar{u} \rfloor \approx \lfloor t'\theta\ \bar{u} \rfloor \tag{$\dagger$}$$

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

We proceed to show the equivalence of $C\theta$ and $C''\theta$ via rewriting with equations of the form ($\dagger$) where $t\theta\bar{u}$ and $t'\theta\bar{u}$ are smaller than the maximal term of $C\theta$. Since x is shallow in C, every occurrence of x in C is not inside a λ-expression and not inside an argument of an applied variable. Therefore, all occurrences of x in C are in a green subterm of the form $x\ \bar{v}$ for some terms $\bar{v}$ that do not contain x. Hence, every occurrence of x in C corresponds to a subterm $\lfloor (x\ \bar{v})\theta \rfloor = \lfloor t\theta\ \bar{v}\theta \rfloor$ in $\lfloor C\theta \rfloor$ and to a subterm $\lfloor (x\ \bar{v})\{x \mapsto t'\}\theta \rfloor = \lfloor t'\theta\ \bar{v}\{x \mapsto t'\}\theta \rfloor = \lfloor t'\theta\ \bar{v}\theta \rfloor$ in $\lfloor C''\theta \rfloor$. These are the only places where $C\theta$ and $C''\theta$ differ.

In order to justify the necessary rewrites from $\lfloor t\theta\ \bar{v}\theta \rfloor$ into $\lfloor t'\theta\ \bar{v}\theta \rfloor$ using ($\dagger$), we need to show that $\lfloor t\theta\ \bar{v}\theta \rfloor$ and $\lfloor t'\theta\ \bar{v}\theta \rfloor$ are smaller than the maximal term in $\lfloor C\theta \rfloor$ for the relevant $\bar{v}$. If $\bar{v}$ is the empty tuple, we do not need to show this because $R \models \lfloor t\theta \approx t'\theta \rfloor$ follows from $\lfloor D\theta \rfloor$ being true and $\lfloor D'\theta \rfloor$ being false. If $\bar{v}$ is non-empty, it suffices to show that $x\ \bar{v}$ is not a maximal term in C. Then $\lfloor t\theta\ \bar{v}\theta \rfloor$ and $\lfloor t'\theta\ \bar{v}\theta \rfloor$, which correspond to the term $x\ \bar{v}$ in C, cannot be maximal in $\lfloor C\theta \rfloor$ and $\lfloor C''\theta \rfloor$ either. Hence they must be smaller than the maximal term in $\lfloor C\theta \rfloor$ because they are subterms of $\lfloor C\theta \rfloor$ and $\lfloor C''\theta \rfloor \prec \lfloor C\theta \rfloor$, respectively.

To show that $x\ \bar{v}$ is not a maximal term in C, we make a case distinction on whether $[\neg]s\theta \approx s'\theta$ is selected in $C\theta$ or $s\theta$ is the maximal term in $C\theta$. One of these must hold because $[\neg]s\theta \approx s'\theta$ is eligible in $C\theta$. If it is selected, by the selection restrictions, x cannot be the head of a maximal term of C. If $s\theta$ is the maximal term in $C\theta$, we can argue that x is a green subterm of s and, since x is shallow, s cannot be of the form $x\ \bar{v}$ for a non-empty $\bar{v}$.

This justifies the necessary rewrites between $\lfloor C\theta \rfloor$ and $\lfloor C''\theta \rfloor$ and it follows that $R \models \lfloor C\theta \rfloor$. □

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

Lemma 20 (Model construction). *Let $\lfloor C\theta \rfloor \in \lfloor \mathcal{G}_\Sigma(N) \rfloor$. We have*

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

Proof. We use induction of the clause order $\succ$ on floor logic ground clauses and assume that (i)–(iv) are already satisfied for all clauses in $\lfloor \mathcal{G}_\Sigma(N) \rfloor$ that are smaller than $\lfloor C\theta \rfloor$. The 'if' part of (i) is obvious from the construction and that condition (iii) follows from (i) by Lemmas 11 and 12. So it remains to show (ii), (iv), and the 'only if' part of (i), i.e., we show the following: If $E_{\lfloor C\theta \rfloor} = \emptyset$ or $C\theta$ is redundant with respect to $\mathcal{G}_\Sigma(N)$ or $C\theta$ has selected literals, then $R_{\lfloor C\theta \rfloor} \models \lfloor C\theta \rfloor$. Without loss of generality, we assume that the selected literals in $C \in N$ correspond to those in $C\theta$.

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

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

Case 2.1: $s\theta = s'\theta$. Then there is an EqRes inference:

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

By Lemma 18, $\lfloor C'\theta \rfloor$ is entailed by clauses from $\lfloor \mathcal{G}_\Sigma(N) \rfloor$ that are smaller than $\lfloor C\theta \rfloor$. By part (iii) of the induction hypothesis, these clauses are true in $R_{\lfloor C\theta \rfloor}$, which implies that $\lfloor C'\theta \rfloor$ and hence $\lfloor C\theta \rfloor$ are true in $R_{\lfloor C\theta \rfloor}$.

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

There is a Sup inference

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

If this inference is not liftable, by Lemma 19, $\neg\lfloor D'\theta \rfloor$ 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 12. 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 18, $\lfloor D'\theta \vee C'\theta \vee s\theta\langle t'\theta\rangle \not\approx s'\theta \rfloor$ is entailed by clauses from $\lfloor \mathcal{G}_\Sigma(N) \rfloor$ that are smaller than $\lfloor C\theta \rfloor$. By part (iii) of the induction hypothesis, $\lfloor D'\theta \vee C'\theta \vee s\theta\langle t'\theta\rangle \not\approx s'\theta \rfloor$ is then true in $R_{\lfloor C\theta \rfloor}$. Since $\lfloor D'\theta \rfloor$ is false in $R_{\lfloor C\theta \rfloor}$, it follows that $R_{\lfloor C\theta \rfloor} \models C'\theta$ or $R_{\lfloor C\theta \rfloor} \models s\theta\langle t'\theta\rangle \not\approx s'\theta$. In the latter case, we have $R_{\lfloor C\theta \rfloor} \models s\theta\langle t\theta\rangle \not\approx s'\theta$ because $t\theta \to t\theta' \in R_{\lfloor C\theta \rfloor}$. Hence, in both cases, $R_{\lfloor C\theta \rfloor} \models C\theta$.

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

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

$$\frac{C\theta \;=\; C''\theta \vee t\theta \approx t'\theta \vee s\theta \approx s'\theta}{C''\theta \vee t'\theta \not\approx s'\theta \vee t\theta \approx t'\theta}\,\text{EqFact}$$

By Lemma 18, its conclusion is entailed by clauses from $\lfloor \mathcal{G}_\Sigma(N) \rfloor$ that are smaller than $\lfloor C\theta \rfloor$. By part (iii) of the induction hypothesis, these clauses are true in $R_{\lfloor C\theta \rfloor}$, which implies that $\lfloor C''\theta \vee t'\theta \not\approx s'\theta \vee t\theta \approx t'\theta \rfloor$ is true in $R_{\lfloor C\theta \rfloor}$. Since $t'\theta = s'\theta$ and hence $\lfloor t'\theta \not\approx s'\theta \rfloor$ is false in $R_{\lfloor C\theta \rfloor}$, this implies that $R_{\lfloor C\theta \rfloor} \models \lfloor C\theta \rfloor$.

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

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

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

If this inference is not liftable, by Lemma 19, $\neg\lfloor D'\theta \rfloor$ 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 12. 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 18, $\lfloor D'\theta \vee C'\theta \vee s\theta\langle t'\theta\rangle \approx s'\theta \rfloor$ 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

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

By Lemma 18, its conclusion is entailed by clauses from $\lfloor \mathcal{G}_\Sigma(N) \rfloor$ that are smaller than $\lfloor C\theta \rfloor$. By part (iii) of the induction hypothesis, these clauses are true in $R_{\lfloor C\theta \rfloor}$, which implies that $\lfloor C''\theta \vee t'\theta \not\approx s'\theta \vee t\theta \approx t'\theta \rfloor$ is true in $R_{\lfloor C\theta \rfloor}$. Since the literal $\lfloor t'\theta \not\approx s'\theta \rfloor$ must be false in $R_{\lfloor C\theta \rfloor}$, this implies that $R_{\lfloor C\theta \rfloor} \models \lfloor C\theta \rfloor$, contradicting our assumption. □

4.5 Construction of the Higher-Order Model

Notation: We write $s \sim t$ for ground ceiling terms s and t as an abbreviation for $R_\infty \models \lfloor s \rfloor \approx \lfloor t \rfloor$, which is equivalent to $[\![\lfloor s \rfloor]\!]_{R_\infty} = [\![\lfloor t \rfloor]\!]_{R_\infty}$.

Lemma 21. *For all ceiling logic ground terms $t, s : \tau \to \upsilon$, the following statements are equivalent:*

1. $t \sim s$
2. $t\,(\mathsf{diff}\langle \tau, \upsilon \rangle(t, s)) \sim s\,(\mathsf{diff}\langle \tau, \upsilon \rangle(t, s))$
3. $t\,u \sim s\,u$ *for all ceiling logic ground terms* u

Proof. (3)⇒(2): Let $u = \mathsf{diff}\langle \tau, \upsilon \rangle(t, s)$.

(2)⇒(1): As a ground instance of the extensionality axiom, we have

$$R_\infty \models \lfloor t\,(\mathsf{diff}\langle \tau, \upsilon \rangle(t, s)) \not\approx s\,(\mathsf{diff}\langle \tau, \upsilon \rangle(t, s)) \vee t \approx s \rfloor$$

Hence, it follows from $t\,(\mathsf{diff}\langle \tau, \upsilon \rangle(t, s)) \sim s\,(\mathsf{diff}\langle \tau, \upsilon \rangle(t, s))$ that $t \sim s$.

(1)⇒(3): We assume that $t \sim s$, i.e., $\lfloor t \rfloor \leftrightarrow^*_{R_\infty} \lfloor s \rfloor$. By induction on the number of rewriting steps between $\lfloor t \rfloor$ and $\lfloor s \rfloor$ and by transitivity of $\sim$, it suffices to show that

$$\lfloor t \rfloor \to_{R_\infty} \lfloor s \rfloor \text{ implies } t\,u \sim s\,u$$

If the rewrite step $\lfloor t \rfloor \to_{R_\infty} \lfloor s \rfloor$ is not at the top level, then neither $s{\downarrow_{\beta\eta}}$ nor $t{\downarrow_{\beta\eta}}$ can be λ-expressions. Therefore, $s{\downarrow_{\beta\eta}}\, u$ and $t{\downarrow_{\beta\eta}}\, u$ are in $\beta\eta$-normal form and there is an analogous rewrite step $\lfloor t\, u \rfloor \to_{R_\infty} \lfloor s\, u \rfloor$ using the same rewrite rule. It follows that $t\, u \sim s\, u$.

If the rewrite step $\lfloor t \rfloor \to_{R_\infty} \lfloor s \rfloor$ is at the top level, $\lfloor t \rfloor \to \lfloor s \rfloor$ must be a rule of R_∞. This rule must come from a productive clause of the form $\lfloor C\theta \rfloor = \lfloor C'\theta \vee t'\theta \approx s'\theta \rfloor$ where $t'\theta = t$ and $s'\theta = s$. Without loss of generality, we assume that the selected literals in $C = C' \vee t' \approx s' \in N$ correspond to those in $C\theta$. By the definition of productive clauses and by Lemma 20 (i) and (iv), $t \approx s$ is strictly eligible in $\lfloor C\theta \rfloor$, and so is $t' \approx s'$ in C. Moreover, t and s have a functional type, and hence t' and s' have either a functional or a polymorphic type. Hence, there is this ArgCong inference:

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

where x is a fresh variable and σ is at least as general as θ.

By part (ii) of Lemma 20, a productive clause is never redundant; hence $C\theta$ is not redundant with respect to $\mathcal{G}_\Sigma(N)$ and therefore C is not redundant with respect to N. Hence, the conclusion $(C' \vee t'\, x \approx s'\, x)\sigma$ of the above inference from C is in $N \cup \mathrm{Red}(N)$. Therefore, the ground instance $\lfloor C'\theta \vee t\, u \approx s\, u \rfloor$ of $\lfloor (C' \vee t'\, x \approx s'\, x)\sigma \rfloor$ is either contained in $\lfloor \mathcal{G}_\Sigma(N) \rfloor$ or it is entailed by clauses in $\lfloor \mathcal{G}_\Sigma(N) \rfloor$. Thus, it is true in R_∞, because R_∞ is a model of $\lfloor \mathcal{G}_\Sigma(N) \rfloor$. By Lemma 12, $\lfloor C'\theta \rfloor$ is false in R_∞. So, $\lfloor t\, u \rfloor \approx \lfloor s\, u \rfloor$ must be true in R_∞. □

Lemma 22. *Let s be a ceiling logic term and θ, θ' ground substitutions, such that $x\theta \sim x\theta'$ for all variables x and $\alpha\theta = \alpha\theta'$ for all type variables α. Then $s\theta \sim s\theta'$.*

Proof. In this proof, we consider all terms to be λ-terms, not $\beta\eta$-equivalence classes. The lemma however holds for $\beta\eta$-equivalence classes as well because for the relation $\sim$ only the $\beta\eta$-normal form matters.

Definition ($\oplus$). We extend the syntax of terms with a new polymorphic function symbol $\oplus : \Pi\alpha.\ \alpha \to \alpha \to \alpha$. We will omit the type argument of $\oplus$ for brevity. We have two reduction rules for $\oplus$: $\oplus\, t\, s \to t$ and $\oplus\, t\, s \to s$.

The computability path order ($\prec_{\mathsf{CPO}}$) [21] guarantees that

- $f(\bar{t}) \succ_{\mathsf{CPO}} v$ if $f \in \Sigma$ and for some i, $t_i \succeq_{\mathsf{CPO}} v$ due to rule $\mathcal{F}_b\triangleright$, and
- $(\lambda x.\, t)\, u \succ_{\mathsf{CPO}} v$ if $t[x \mapsto u] \succeq_{\mathsf{CPO}} v$ due to rule $@\beta$,

thus this order is compatible with the $\oplus$-reduction rules and β-reduction. Since this order is moreover well-founded, these reduction rules terminate. Since the reduction rules describe a finitely branching term rewrite system, for any term there is a maximal number of reduction steps possible (Kőnigs Lemma).

Definition (term-ground). A term is *term-ground* if it does not contain free (term) variables. It may contain polymorphic type arguments.

Definition ($\mathcal{S}$). We define an auxiliary function $\mathcal{S}$ that essentially measures the size of a term, but assigns size 1 to term-ground terms.

$$
\begin{array}{rl}
\mathcal{S}(x) = 1 & \text{for (free or bound) variables } x \\
\mathcal{S}(t) = 1 & \text{for term-ground terms } t \\
\mathcal{S}(\mathsf{f}\langle\bar{\tau}\rangle) = 1 & \text{for a non-term-ground expression } \mathsf{f}\langle\bar{\tau}\rangle \\
\mathcal{S}(\lambda x.\, t) = 1 + \mathcal{S}(t) & \text{for non-term-ground } \lambda\text{-expressions } \lambda x.\, t \\
\mathcal{S}(t_1\, t_2) = \mathcal{S}(t_1) + \mathcal{S}(t_2) & \text{for non-term-ground applications } t_1\, t_2
\end{array}
$$

We prove $s\theta \sim s\theta'$ by induction on s, θ, and θ' using the left-to-right lexicographic order on the triple $(n_1(s), n_2(s), n_3(s)) \in \mathbb{N}^3$, where

- $n_1(s)$ is the maximal number of $\beta\oplus$-reduction steps starting from $s\sigma_{\theta,\theta'}$, where $\sigma_{\theta,\theta'}$ is the substitution mapping each term variable x to $\oplus\, x\theta\, x\theta'$.
- $n_2(s)$ is the number of term variables occurring more than once in s.
- $n_3(s) = \mathcal{S}(s)$.

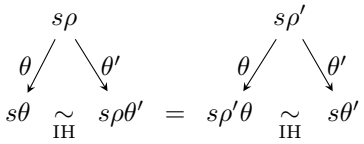
Strictly speaking, n_1 also depends on θ and θ', but we write $n_1(s)$ instead of $n_1(s, \theta, \theta')$ for brevity.

Case 1: s is term-ground. Then the lemma is trivial.

Case 2: s contains $k \geq 2$ free term variables. Then we can apply the induction hypothesis twice and use the transitivity of $\sim$ as follows. Let x be one of the free term variables in s. Let $\rho = \{x \mapsto x\theta\}$ the substitution that maps x to $x\theta$ and ignores all other variables. Let $\rho' = \theta'[x \mapsto x]$.

We want to invoke the induction hypothesis on $s\rho$ and $s\rho'$. This is justified because $s\sigma_{\theta,\theta'}$ $\oplus$-reduces to $s\rho\sigma_{\theta,\theta'}$ and to $s\rho'\sigma_{\theta,\theta'}$. Hence, $n_1(s) > n_1(s\rho)$ and $n_1(s) > n_1(s\rho')$.

This application of the induction hypothesis gives us $s\rho\theta \sim s\rho\theta'$ and $s\rho'\theta \sim s\rho'\theta'$. Since $s\theta = s\rho\theta$, $s\rho\theta' = s\rho'\theta$, and $s\rho'\theta' = s\theta'$, this implies $s\theta \sim s\theta'$ by transitivity of $\sim$, as illustrated in this diagram:



Case 3: s contains a free term variable that occurs more than once. Then we rename variable occurrences apart by replacing each occurrence of each term variable x by a fresh variable x_i, for which we define $x_i\theta = x\theta$ and $x_i\theta' = x\theta'$. Let s' be the resulting term. Since $s\sigma_{\theta,\theta'} = s'\sigma_{\theta,\theta'}$, we have $n_1(s) = n_1(s')$. All term variables occur only once in s'. Hence, $n_2(s) > 0 = n_2(s')$. Therefore, we

can invoke the induction hypothesis on s' to obtain $s'\theta \sim s'\theta'$. Since $s\theta = s'\theta$ and $s\theta' = s'\theta'$, it follows that $s\theta \sim s\theta'$.

Case 4: s contains only one free term variable x, which occurs exactly once.

Case 4.1: s is of the form $\mathsf{f}\langle\bar{\tau}\rangle\,\bar{t}$ for some symbol f, some types $\bar{\tau}$ and some terms $\bar{t}$. Then let u be the term in $\bar{t}$ that contains x. We want to apply the induction hypothesis to u, which can be justified as follows. Consider the longest sequence of $\beta\oplus$-reductions from $u\sigma_{\theta,\theta'}$. This sequence can be replicated inside $s\sigma_{\theta,\theta'} = (\mathsf{f}\langle\bar{\tau}\rangle\bar{t})\sigma_{\theta,\theta'}$. Therefore, the longest sequence of $\beta\oplus$-reductions from $s\sigma_{\theta,\theta'}$ is at least as long, i.e., $n_1(s) \geq n_1(u)$. Since both s and u have only one term variable occurrence, we have $n_2(s) = 0 = n_2(u)$. But $n_3(s) > n_3(u)$ because u is a non-term-ground subterm of s.

Applying the induction hypothesis gives us $u\theta \sim u\theta'$. By definition of $\lfloor\ \rfloor$, we have

$$\begin{aligned}\lfloor(\mathsf{f}\langle\bar{\tau}\rangle\,\bar{t})\theta\rfloor &= \mathsf{f}_m\langle\lfloor\bar{\tau}\theta\rfloor\rangle\,\lfloor\bar{t}\theta\rfloor \text{ and}\\ \lfloor(\mathsf{f}\langle\bar{\tau}\rangle\,\bar{t})\theta'\rfloor &= \mathsf{f}_m\langle\lfloor\bar{\tau}\theta'\rfloor\rangle\,\lfloor\bar{t}\theta'\rfloor\end{aligned}$$

where m is the length of $\bar{t}$. By congruence of $\approx$ in the floor logic, it follows that $s\theta \sim s\theta'$.

Case 4.2: s is of the form $x\,\bar{t}$ for some terms $\bar{t}$. Then we observe that, by assumption $x\theta \sim x\theta'$. By applying Lemma 21 repeatedly, we have $x\theta\,\bar{t} \sim x\theta'\,\bar{t}$. Since x occurs only once, $\bar{t}$ is term-ground and hence $s\theta = x\theta\,\bar{t}$ and $s\theta' = x\theta'\,\bar{t}$. Therefore, $s\theta \sim s\theta'$.

Case 4.3: s is of the form $(\lambda z.\, u)$ for some term u. Then we observe that to prove $s\theta \sim s\theta'$, it suffices to show that $s\theta\ (\mathsf{diff}\ s\theta\ s\theta') \sim s\theta'\ (\mathsf{diff}\ s\theta\ s\theta')$ by Lemma 21. Via $\beta\eta$-conversion, this is equivalent to $u\rho\theta \sim u\rho\theta'$ where $\rho = \{z \mapsto \mathsf{diff}\langle\tau,\upsilon\rangle\,(s\theta{\downarrow_{\beta\eta}})\,(s\theta'{\downarrow_{\beta\eta}})\}$ where $\tau \to \upsilon$ is the type of $s\theta$. To prove $u\rho\theta \sim u\rho\theta'$, we apply the induction hypothesis on $u\rho$.

It remains to show that the induction hypothesis is applicable on $u\rho$. Consider the longest sequence of $\beta\oplus$-reductions from $u\rho\sigma_{\theta,\theta'}$. Since $z\rho$ starts with the diff symbol, $z\rho$ will not cause more $\beta\oplus$-reductions than z. Hence, the same sequence of $\beta\oplus$-reductions can be applied inside $s\sigma_{\theta,\theta'} = (\lambda z.\, u)\sigma_{\theta,\theta'}$, proving that $n_1(s) \geq n_1(u\rho)$. Since both s and $u\rho$ have only one term variable occurrence, $n_2(s) = 0 = n_2(u\rho)$. But $n_3(s) = \mathcal{S}(s) = 1 + \mathcal{S}(u)$ because s is non-term-ground. Moreover, $\mathcal{S}(u) \geq \mathcal{S}(u\rho) = n_3(u\rho)$ because ρ replaces a variable by a ground term. Hence, $n_3(s) > n_3(u\rho)$, which justifies the application of the induction hypothesis.

Case 4.4: s is of the form $(\lambda z.\, u)\,t_0\,\bar{t}$ for some terms u, t_0, and $\bar{t}$.

We apply the induction hypothesis on $s' = u\{z \mapsto t_0\}\,\bar{t}$. To justify it, consider the longest sequence of $\beta\oplus$-reductions from $s'\sigma_{\theta,\theta'}$. Prepending the reduction

$s\sigma_{\theta,\theta'} \to_\beta s'\sigma_{\theta,\theta'}$ to it gives us a longer sequence from $s\sigma_{\theta,\theta'}$. Hence, $n_1(s) > n_1(s')$.

The induction hypothesis gives us $s'\theta \sim s'\theta'$. Since $\sim$ is invariant under β-reductions, it follows that $s\theta \sim s\theta'$. □

Definition of the ceiling logic The interpretation R_∞ defined above is an interpretation in monomorphic first-order logic. Let $\mathcal{U}_\tau$ be its universe for type τ and and $\mathcal{I}$ its interpretation function. When defining the universe $\mathcal{U}^\uparrow$ of the ceiling model, we need to ensure that it contains subsets of function spaces, since $\mathcal{I}^\uparrow_{\mathsf{ty}}(\to)(\mathcal{D}_1, \mathcal{D}_2)$ must be a subset of the function space from $\mathcal{D}_1$ to $\mathcal{D}_2$ for all $\mathcal{D}_1, \mathcal{D}_2 \in \mathcal{U}^\uparrow$. But the floor universes $\mathcal{U}_\tau$ consist of equivalence classes of floor logic terms with respect to the rewrite system R_∞, not of functions.

Towards this end, we define a family of functions $\mathcal{E}_\tau$ that give a meaning to the elements of each floor universe $\mathcal{U}_\tau$. As in the floor model, there will be a one-to-one correspondence of ground ceiling types and ceiling domains. We write $\mathcal{D}^\uparrow_\tau$ for the domain corresponding to the ground type τ.

We define $\mathcal{E}_\tau$ and $\mathcal{D}^\uparrow_\tau$ in a mutual induction and prove that $\mathcal{E}_\tau$ is a bijection simultaneously. We start with non-function types τ (non-function meaning that τ is not of the form $\tau_1 \to \tau_2$ for any τ_1, τ_2):

$$\mathcal{D}^\uparrow_\tau = \mathcal{U}_\tau$$
$$\mathcal{E}_\tau : \mathcal{U}_\tau \longrightarrow \mathcal{D}^\uparrow_\tau \text{ is the identity}$$

We proceed by defining $\mathcal{E}_{\tau\to\upsilon}$ and $\mathcal{D}^\uparrow_{\tau\to\upsilon}$ for function types $\tau \to \upsilon$. We assume that $\mathcal{E}_\tau$, $\mathcal{E}_\upsilon$, $\mathcal{D}^\uparrow_\tau$, and $\mathcal{D}^\uparrow_\upsilon$ have already been defined and that $\mathcal{E}_\tau$, $\mathcal{E}_\upsilon$ are bijections. To ensure that $\mathcal{E}_{\tau\to\upsilon}$ will be bijective, we first define an injective function $\mathcal{E}^0_{\tau\to\upsilon} : \mathcal{U}_{\tau\to\upsilon} \longrightarrow \mathcal{D}^\uparrow_\tau \longrightarrow \mathcal{D}^\uparrow_\upsilon$, define $\mathcal{D}^\uparrow_{\tau\to\upsilon}$ as its image $\mathcal{E}^0_{\tau\to\upsilon}(\mathcal{U}_{\tau\to\upsilon})$, and finally define $\mathcal{E}_{\tau\to\upsilon}$ as $\mathcal{E}^0_{\tau\to\upsilon}$ with its codomain restricted to $\mathcal{D}^\uparrow_{\tau\to\upsilon}$.

$$\mathcal{E}^0_{\tau\to\upsilon} : \mathcal{U}_{\tau\to\upsilon} \longrightarrow \mathcal{D}^\uparrow_\tau \longrightarrow \mathcal{D}^\uparrow_\upsilon$$
$$\mathcal{E}^0_{\tau\to\upsilon}(\llbracket\lfloor s\rfloor\rrbracket_{R_\infty})\big(\mathcal{E}_\tau\big(\llbracket\lfloor u\rfloor\rrbracket_{R_\infty}\big)\big) = \mathcal{E}_\upsilon\big(\llbracket\lfloor s\ u\rfloor\rrbracket_{R_\infty}\big)$$

This is a valid definition because each element of $\mathcal{U}_{\tau\to\upsilon}$ is of the form $\llbracket\lfloor s\rfloor\rrbracket_{R_\infty}$ for some s and each element of $\mathcal{D}^\uparrow_\tau$ is of the form $\mathcal{E}_\tau\big(\llbracket\lfloor u\rfloor\rrbracket_{R_\infty}\big)$ for some u. This function is well defined if it does not depend on the choice of s and u. To show this, we assume that there are other ground terms t and v such that $\llbracket\lfloor s\rfloor\rrbracket_{R_\infty} = \llbracket\lfloor t\rfloor\rrbracket_{R_\infty}$ and $\mathcal{E}_\tau\big(\llbracket\lfloor u\rfloor\rrbracket_{R_\infty}\big) = \mathcal{E}_\tau\big(\llbracket\lfloor v\rfloor\rrbracket_{R_\infty}\big)$. Since $\mathcal{E}_\tau$ is bijective, we have $\llbracket\lfloor u\rfloor\rrbracket_{R_\infty} = \llbracket\lfloor v\rfloor\rrbracket_{R_\infty}$. Applying Lemma 22 to the term $x\ y$ and the substitutions $\{x \mapsto s, y \mapsto u\}$ and $\{x \mapsto t, y \mapsto v\}$, it follows that

$$\llbracket\lfloor s\ u\rfloor\rrbracket_{R_\infty} = \llbracket\lfloor t\ v\rfloor\rrbracket_{R_\infty}$$

indicating that $\mathcal{E}^0_{\tau\to\upsilon}$ is well defined.

It remains to show that $\mathcal{E}^0_{\tau\to\upsilon}$ is injective as a function from $\mathcal{U}_{\tau\to\upsilon}$ to $\mathcal{D}^\uparrow_\tau \longrightarrow \mathcal{D}^\uparrow_\upsilon$. Assume two wellformed ground ceiling terms s and t such that for all wellformed ground ceiling terms u, we have $\llbracket\lfloor s\ u\rfloor\rrbracket_{R_\infty} = \llbracket\lfloor t\ u\rfloor\rrbracket_{R_\infty}$. By Lemma 21,

it follows that $[\![\lfloor s \rfloor]\!]_{R_\infty} = [\![\lfloor t \rfloor]\!]_{R_\infty}$, which concludes the proof that $\mathcal{E}^0_{\tau\to\upsilon}$ is injective.

We define $\mathcal{D}^\uparrow_{\tau\to\upsilon}$ and $\mathcal{E}_{\tau\to\upsilon}$ as follows:

$$\mathcal{D}^\uparrow_{\tau\to\upsilon} = \mathcal{E}^0_{\tau\to\upsilon}(\mathcal{U}_{\tau\to\upsilon})$$
$$\mathcal{E}_{\tau\to\upsilon} : \mathcal{U}_{\tau\to\upsilon} \longrightarrow \mathcal{D}^\uparrow_{\tau\to\upsilon}, \quad a \mapsto \mathcal{E}^0_{\tau\to\upsilon}(a)$$

This ensures that $\mathcal{E}_{\tau\to\upsilon}$ is bijective and concludes the inductive definition of $\mathcal{D}^\uparrow$ and $\mathcal{E}$. In the following, we will usually write $\mathcal{E}$ instead of $\mathcal{E}_\tau$, since the type τ is determined by first argument of $\mathcal{E}_\tau$.

We define the ceiling universe as $\mathcal{U}^\uparrow = \{\mathcal{D}^\uparrow_\tau \mid \tau \text{ ground}\}$. Moreover,

$$\mathcal{J}^\uparrow_{\mathsf{ty}}(\kappa)(\mathcal{D}^\uparrow_{\bar\tau}) = \mathcal{U}_{\kappa(\bar\tau)} \text{ for all } \kappa \in \Sigma^\uparrow_{\mathsf{ty}}$$

completing the type interpretation $\mathcal{I}^\uparrow_{\mathsf{ty}} = (\mathcal{U}^\uparrow, \mathcal{J}^\uparrow_{\mathsf{ty}})$.

We define the ceiling logic interpretation function as

$$\mathcal{J}^\uparrow(\mathsf{f}, \mathcal{D}^\uparrow_{\bar\upsilon_m}) = \mathcal{E}(\mathcal{J}(\mathsf{f}_0^{\bar\upsilon_m}))$$

for all $\mathsf{f} : \Pi\bar\alpha_m.\ \tau$.

Finally, we need to define the designation function $\mathcal{L}$, which takes a ceiling logic valuation ξ and a λ-expression as arguments. Given a valuation ξ, we choose a grounding substitution θ_ξ such that

$$\mathcal{D}^\uparrow_{\alpha\theta_\xi} = \xi(\alpha) \text{ and } \mathcal{E}([\![\lfloor x\theta_\xi \rfloor]\!]_{R_\infty}) = \xi(x)$$

for all variables x and all type variables α. Such a substitution can be constructed as follows: We can fulfill the first equation in a unique way because there is a one-to-one correspondence between ground types and ceiling domains. Since $\mathcal{E}^{-1}(\xi(x))$ is an element of a floor domain and R_∞ is term-generated, there is a ground term t with $[\![t]\!]^\xi_{R_\infty} = \mathcal{E}^{-1}(\xi(x))$. Choosing one such t and defining $x\theta_\xi = \lceil t \rceil$ gives us a grounding substitution θ_ξ with the desired property.

We define

$$\mathcal{L}(\xi, (\lambda x.\ t)) = \mathcal{E}([\![\lfloor (\lambda x.\ t)\theta_\xi \rfloor]\!]_{R_\infty})$$

To prove that this is well-defined, we assume that there is another substitution θ'_ξ with the properties $\mathcal{D}^\uparrow_{\alpha\theta'_\xi} = \xi(\alpha)$ for all α and $\mathcal{E}([\![\lfloor x\theta'_\xi \rfloor]\!]_{R_\infty}) = \xi(x)$ for all x. Then we have $\alpha\theta_\xi = \alpha\theta'_\xi$ for all α due to the one-to-one correspondence of ceiling domains and types. We have $[\![\lfloor x\theta_\xi \rfloor]\!]_{R_\infty} = [\![\lfloor x\theta'_\xi \rfloor]\!]_{R_\infty}$ for all x because $\mathcal{E}$ is injective. By Lemma 22 it follows that $[\![\lfloor (\lambda x.\ t)\theta_\xi \rfloor]\!]_{R_\infty} = [\![\lfloor (\lambda x.\ t)\theta'_\xi \rfloor]\!]_{R_\infty}$, which proves that $\mathcal{L}$ is well-defined.

This concludes the definition of the ceiling interpretation $R^\uparrow_\infty = (\mathcal{U}^\uparrow, \mathcal{J}^\uparrow, \mathcal{L})$. It remains to show that $R^\uparrow_\infty$ is proper. In a proper interpretation, the denotation $[\![t]\!]_{R^\uparrow_\infty}$ of a term t does not depend on the representative of t modulo $\beta\eta$, but since we haven't shown $R^\uparrow_\infty$ to be proper yet, we cannot rely on this. That is why we use λ-terms in the following three lemmas and we mark all $\beta\eta$-reductions explicitly.

The ceiling interpretation $R^\uparrow_\infty = (\mathcal{U}^\uparrow, \mathcal{J}^\uparrow, \mathcal{L})$ relates to the floor interpretation R_∞ as follows:

Lemma 23. *For all ground ceiling logic λ-terms t:*

$$[\![t]\!]_{R^{\uparrow}_{\infty}} = \mathcal{E}([\![\lfloor t{\downarrow_{\beta\eta}}\rfloor]\!]_{R_\infty})$$

Proof. By induction on t.

Assume that $[\![s]\!]_{R^{\uparrow}_{\infty}} = \mathcal{E}([\![\lfloor s{\downarrow_{\beta\eta}}\rfloor]\!]_{R_\infty})$ for all proper subterms s of t.

If t is of the form $\mathsf{f}\langle\bar{\tau}\rangle$, then

$$\begin{aligned}
[\![t]\!]_{R^{\uparrow}_{\infty}} &= \mathcal{I}^{\uparrow}(\mathsf{f}, \mathcal{D}^{\uparrow}_{\bar{\tau}}) \\
&= \mathcal{E}(\mathcal{I}(\mathsf{f}_0, \mathcal{U}_{\lfloor\bar{\tau}\rfloor})) \\
&= \mathcal{E}([\![\mathsf{f}_0\langle\lfloor\bar{\tau}\rfloor\rangle]\!]_{R_\infty}) \\
&= \mathcal{E}([\![\lfloor\mathsf{f}\langle\bar{\tau}\rangle\rfloor]\!]_{R_\infty}) \\
&= \mathcal{E}([\![\lfloor\mathsf{f}\langle\bar{\tau}\rangle{\downarrow_{\beta\eta}}\rfloor]\!]_{R_\infty}) = \mathcal{E}([\![\lfloor t{\downarrow_{\beta\eta}}\rfloor]\!]_{R_\infty})
\end{aligned}$$

If t is an application $t = t_1\ t_2$, where t_1 is of type $\tau \to \upsilon$, then

$$\begin{aligned}
[\![t_1\ t_2]\!]_{R^{\uparrow}_{\infty}} &= [\![t_1]\!]_{R^{\uparrow}_{\infty}}([\![t_2]\!]_{R^{\uparrow}_{\infty}}) \\
&\overset{\text{IH}}{=} \mathcal{E}_{\tau\to\upsilon}([\![\lfloor t_1{\downarrow_{\beta\eta}}\rfloor]\!]_{R_\infty})(\mathcal{E}_\tau([\![\lfloor t_2{\downarrow_{\beta\eta}}\rfloor]\!]_{R_\infty})) \\
&\overset{\text{Def } \mathcal{E}}{=} \mathcal{E}_\upsilon([\![\lfloor t_1\ t_2{\downarrow_{\beta\eta}}\rfloor]\!]_{R_\infty})
\end{aligned}$$

If t is a λ-expression, then

$$\begin{aligned}
[\![\lambda x.\ u]\!]^{\xi}_{R^{\uparrow}_{\infty}} &= \mathcal{L}(\xi, (\lambda x.\ u)) \\
&= \mathcal{E}([\![\lfloor(\lambda x.\ u)\theta_\xi{\downarrow_{\beta\eta}}\rfloor]\!]_{R_\infty}) \\
&= \mathcal{E}([\![\lfloor\lambda x.\ u{\downarrow_{\beta\eta}}\rfloor]\!]_{R_\infty})
\end{aligned}$$

□

We need to show that the interpretation $R^{\uparrow}_{\infty} = (\mathcal{U}^{\uparrow}, \mathcal{I}^{\uparrow}, \mathcal{L})$ is proper. In the proof, we will need to employ the following lemma, which is essentially the well-known *substitution lemma*. This lemma holds in general for proper interpretations [32, Section I.2.6.1], but we must prove it here for our particular interpretation because we have not shown that our interpretation is proper yet.

Lemma 24 (Substitution lemma). *For all ceiling logic λ-terms t, ceiling types τ and grounding substitutions ρ:*

$$[\![\tau\rho]\!]_{\mathcal{I}^{\uparrow}_{\mathsf{ty}}} = [\![\tau]\!]^{\xi}_{\mathcal{I}^{\uparrow}_{\mathsf{ty}}} \text{ and } [\![t\rho]\!]_{R^{\uparrow}_{\infty}} = [\![t]\!]^{\xi}_{R^{\uparrow}_{\infty}}$$

where $\xi(\alpha) = [\![\alpha\rho]\!]_{\mathcal{I}^{\uparrow}_{\mathsf{ty}}}$ for all α and $\xi(x) = [\![x\rho]\!]_{R^{\uparrow}_{\infty}}$ for all x.

Proof. First, we prove that $[\![\tau\rho]\!]_{\mathcal{I}^{\uparrow}_{\mathsf{ty}}} = [\![\tau]\!]^{\xi}_{\mathcal{I}^{\uparrow}_{\mathsf{ty}}}$ by induction on the structure of τ. If $\tau = \alpha$ is a type variable,

$$[\![\alpha\rho]\!]_{\mathcal{I}^{\uparrow}_{\mathsf{ty}}} = \xi(\alpha) = [\![\alpha]\!]^{\xi}_{\mathcal{I}^{\uparrow}_{\mathsf{ty}}}$$

If $\tau = \kappa(\bar{\upsilon})$ for some type constructor κ and types $\bar{\upsilon}$,

$$[\![\kappa(\bar{\upsilon})\rho]\!]_{\mathcal{I}^{\uparrow}_{\mathsf{ty}}} = \mathcal{J}_{\mathsf{ty}}(\kappa)([\![\bar{\upsilon}\rho]\!]_{\mathcal{I}^{\uparrow}_{\mathsf{ty}}}) \overset{IH}{=} \mathcal{J}_{\mathsf{ty}}(\kappa)([\![\bar{\upsilon}]\!]^{\xi}_{\mathcal{I}^{\uparrow}_{\mathsf{ty}}}) = [\![\kappa(\bar{\upsilon})]\!]^{\xi}_{\mathcal{I}^{\uparrow}_{\mathsf{ty}}}$$

Next, we prove $[\![t\rho]\!]_{R^\uparrow_\infty} = [\![t]\!]^\xi_{R^\uparrow_\infty}$ by induction on the structure of t. If $t = y$, then by the definition of the denotation of a variable

$$[\![y\rho]\!]_{R^\uparrow_\infty} = \xi(y) = [\![y]\!]^\xi_{R^\uparrow_\infty}$$

If $t = \mathsf{f}\langle\bar{\tau}\rangle$, then by the definition of the term denotation

$$[\![\mathsf{f}\langle\bar{\tau}\rangle\rho]\!]_{R^\uparrow_\infty} = \mathcal{J}^\uparrow(\mathsf{f}, [\![\bar{\tau}\rho]\!]_{\mathcal{I}^\uparrow_{\mathsf{ty}}}) \overset{IH}{=} \mathcal{J}^\uparrow(\mathsf{f}, [\![\bar{\tau}]\!]^\xi_{\mathcal{I}^\uparrow_{\mathsf{ty}}}) = [\![\mathsf{f}\langle\bar{\tau}\rangle]\!]^\xi_{R^\uparrow_\infty}$$

If $t = u\ v$, then by the definition of the term denotation

$$[\![(u\ v)\rho]\!]_{R^\uparrow_\infty} = [\![u\rho]\!]_{R^\uparrow_\infty}([\![v\rho]\!]_{R^\uparrow_\infty}) \overset{IH}{=} [\![u]\!]^\xi_{R^\uparrow_\infty}([\![v]\!]^\xi_{R^\uparrow_\infty}) = [\![u\ v]\!]^\xi_{R^\uparrow_\infty}$$

If t is a lambda-expression:

$[\![(\lambda z.\ u)\rho]\!]_{R^\uparrow_\infty}$	
$= [\![(\lambda z.\ u\rho')]\!]_{R^\uparrow_\infty}$	where $\rho'(z) = z$ and $\rho'(x) = \rho(x)$ for $x \neq z$
$= \mathcal{L}(\xi', (\lambda z.\ u\rho'))$	by the definition of term denotation where ξ' is arbitrary
$= \mathcal{E}([\![\lfloor(\lambda z.\ u)\rho\theta_{\xi'}\downarrow_{\beta\eta}\rfloor]\!]_{R_\infty})$	by definition of $\mathcal{L}$
$= \mathcal{E}([\![\lfloor(\lambda z.\ u)\rho\downarrow_{\beta\eta}\rfloor]\!]_{R_\infty})$	because $(\lambda z.\ u)\rho$ is ground
$\overset{*}{=} \mathcal{L}(\xi, \lambda z.\ u)$	by definition of $\mathcal{L}$ and Lemma 23
$= [\![\lambda z.\ u]\!]^\xi_{R^\uparrow_\infty}$	by the definition of term denotation

The step (*) is justified as follows: By definition of $\mathcal{L}$, $\mathcal{L}(\xi, \lambda z.u) = \mathcal{E}([\![\lfloor(\lambda z.\ u)\theta_\xi\downarrow_{\beta\eta}\rfloor]\!]_{R_\infty})$, where θ_ξ is a substitution such that $\mathcal{D}^\uparrow_{\alpha\theta_\xi} = \xi(\alpha)$ for all α and $\mathcal{E}([\![\lfloor x\theta_\xi\downarrow_{\beta\eta}\rfloor]\!]_{R_\infty}) = \xi(x)$ for all x. By the definition of ξ and by Lemma 23, ρ is such a substitution. Hence, $\mathcal{L}(\xi, \lambda z.\ u) = \mathcal{E}([\![\lfloor(\lambda z.\ u)\rho\downarrow_{\beta\eta}\rfloor]\!]_{R_\infty})$. □

Lemma 25. *The interpretation $R^\uparrow_\infty = (\mathcal{U}^\uparrow, \mathcal{J}^\uparrow, \mathcal{L})$ is proper.*

Proof. We have to show that $[\![(\lambda x.\ t)]\!]^\xi_{R^\uparrow_\infty}(a) = [\![t]\!]^{\xi[x\mapsto a]}_{R^\uparrow_\infty}$ for all λ-expressions $(\lambda x.\ t)$, all substitutions ξ, and all a.

$[\![\lambda x.\ t]\!]^\xi_{R^\uparrow_\infty}(a)$	$= \mathcal{L}(\xi, \lambda x.\ t)(a)$	by definition of $[\![\ ]\!]_{R^\uparrow_\infty}$
	$= \mathcal{E}([\![\lfloor(\lambda x.\ t)\theta_\xi\downarrow_{\beta\eta}\rfloor]\!]_{R_\infty})(a)$	by definition of $\mathcal{L}$ where $\mathcal{E}([\![\lfloor z\theta_\xi\rfloor]\!]_{R_\infty}) = \xi(z)$ for all z and $\mathcal{D}^\uparrow_{\alpha\theta_\xi} = \xi(\alpha)$ for all α
	$= \mathcal{E}([\![\lfloor(\lambda x.\ t)\theta_\xi\ s\downarrow_{\beta\eta}\rfloor]\!]_{R_\infty})$	by definition of $\mathcal{E}$ where $\mathcal{E}([\![\lfloor s\rfloor]\!]_{R_\infty}) = a$
	$= \mathcal{E}([\![\lfloor t(\theta_\xi[x \mapsto s])\downarrow_{\beta\eta}\rfloor]\!]_{R_\infty})$	by β-reduction
	$= [\![t(\theta_\xi[x \mapsto s])]\!]_{R^\uparrow_\infty}$	by Lemma 23
	$= [\![t]\!]^{\xi[x\mapsto a]}_{R^\uparrow_\infty}$	by Lemma 24

□

Lemma 26. *$R_\infty^\uparrow$ is a term-generated model of $\mathcal{G}_\Sigma(N)$.*

Proof. By Lemma 23, we have $[\![t]\!]_{R_\infty^\uparrow} = \mathcal{E}([\![\lfloor t \rfloor]\!]_{R_\infty})$ for all ground ceiling logic λ-terms t. By Lemma 25, this statement does not depend on the choice of representative and hence holds also for $\beta\eta$-equivalence classes t. Since $\mathcal{E}$ is a bijection, it follows that a ground (dis)equation $[\neg]\ s \approx t$ is true in $R_\infty^\uparrow$ if and only if $\lfloor [\neg]\ s \approx t \rfloor$ 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 20, R_∞ is a model of $\lfloor \mathcal{G}_\Sigma(N) \rfloor$, i.e., all clauses $\lfloor C \rfloor \in \lfloor \mathcal{G}_\Sigma(N) \rfloor$ are true in R_∞. Hence, all clauses $C \in \mathcal{G}_\Sigma(N)$ are true in $R_\infty^\uparrow$ and therefore $R_\infty^\uparrow$ is a model of $\mathcal{G}_\Sigma(N)$.

It remains to show that $R_\infty^\uparrow$ is term-generated. Let $a \in \mathcal{D}_\tau^\uparrow$. Then $\mathcal{E}^{-1}(a) \in \mathcal{U}_{\lfloor \tau \rfloor}$. Since R_∞ is term-generated, we have a ground term t of the floor logic with $[\![t]\!]_{R_\infty} = \mathcal{E}^{-1}(a)$. By Lemma 23, we have $[\![\lceil t \rceil]\!]_{R_\infty^\uparrow} = \mathcal{E}([\![t]\!]_{R_\infty}) = a$. Hence, $R_\infty^\uparrow$ is term-generated. □

Lemma 27. *$R_\infty^\uparrow$ is a model of N.*

Proof. Let $C \in N$. We want to show that C is true in $R_\infty^\uparrow$ for all valuations ξ. Since $R_\infty^\uparrow$ is term-generated by Lemma 26, for each variable x, there is a ground term s_x such that $[\![s_x]\!]_{R_\infty^\uparrow} = \xi(x)$. Let ρ be the substitution that maps every free variable x in C to s_x. Then $\xi(x) = [\![s_x]\!]_{R_\infty^\uparrow} = [\![x\rho]\!]^\xi_{R_\infty^\uparrow}$ for all x. By treating the type variables of C in the same way, we can also achieve that $\xi(\alpha) = [\![\alpha\rho]\!]^\xi_{\mathcal{I}_{\mathsf{ty}}^\uparrow}$ for all x. By Lemma 24, $[\![t\rho]\!]_{R_\infty^\uparrow} = [\![t]\!]^\xi_{R_\infty^\uparrow}$ for all terms t and $[\![\tau\rho]\!]_{\mathcal{I}_{\mathsf{ty}}^\uparrow} = [\![\tau]\!]^\xi_{\mathcal{I}_{\mathsf{ty}}^\uparrow}$ for all types τ. Hence, $C\rho$ and C have the same truth value in $R_\infty^\uparrow$ for ξ. Since $C\rho$ is ground and therefore true in $R_\infty^\uparrow$ by Lemma 26, C is true as well. □

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

Theorem 28 (Refutational completeness). *Let N be a clause set saturated up to redundancy and containing the extensionality axiom. Then N has a model if and only if $\bot \notin N$.*

Proof. If $\bot \in N$, then obviously N does not have a model. If $\bot \notin N$, then the interpretation R_∞ (that is, $\mathcal{T}_\Sigma^\emptyset / R_\infty$) is a model of $\lfloor \mathcal{G}_\Sigma(N) \rfloor$ according to part (iii) of Lemma 20. By Lemma 26, $R_\infty^\uparrow$ is a term-generated model of $\mathcal{G}_\Sigma(N)$. By Lemma 27, it is a model of N. □

5 Extensions

The calculus can be extended to make it more practical. The familiar simplification machinery can be adapted to higher-order terms by considering green contexts instead of arbitrary contexts. Optional inference rules provide lightweight alternatives to the FluidSup rule and the extensionality axiom.

Two of the rules below are based on "orange subterms." A λ-term t is an *orange subterm* of a λ-term s if $s = t$; or if $s = \mathsf{f}\langle\bar{\tau}\rangle\ \bar{s}$ and t is an orange subterm of s_i for some i; or if $s = x\ \bar{s}$ and t is an orange subterm of s_i for some i; or if

$s = (\lambda x.\, u)$ and t is an orange subterm of u. In $\mathsf{f}\,(\mathsf{g}\,\mathsf{a})\,(y\,\mathsf{b})\,(\lambda x.\, \mathsf{h}\,\mathsf{c}\,(\mathsf{g}\,x))$, the orange subterms include b, c, x, $\mathsf{g}\,x$, $\mathsf{h}\,\mathsf{c}\,(\mathsf{g}\,x)$, and all the green subterms. Following the convention introduced in Section 2.1, this notion is lifted to $\beta\eta$-equivalence classes via representatives in η-short β-normal form. We write $t = s\langle\!\langle \bar{x}_n.\, u \rangle\!\rangle$ to indicate that u is an orange subterm of t, where $\bar{x}_n$ are the variables bound in the *orange context* around u. If $n = 0$, we write $t = s\langle\!\langle u \rangle\!\rangle$.

Once a term $s\langle\!\langle \bar{x}_n.\, u \rangle\!\rangle$ has been introduced, we write $s\langle\!\langle \bar{x}_n.\, u' \rangle\!\rangle_\eta$ to denote the same context with a different subterm u' at that position. The η subscript is a reminder that u' is not necessarily an orange subterm of $s\langle\!\langle \bar{x}_n.\, u' \rangle\!\rangle_\eta$ due to potential applications of η-reduction. For example, if $s\langle\!\langle x.\, \mathsf{g}\,x\,x \rangle\!\rangle = (\lambda x.\, \mathsf{g}\,x\,x)$, then $s\langle\!\langle x.\, \mathsf{f}\,x \rangle\!\rangle_\eta = (\lambda x.\, \mathsf{f}\,x) = \mathsf{f}$.

Demodulation, which destructively rewrites using an equality $t \approx t'$, is available at green positions. A variant rewrites inside λ-expressions:

$$\frac{t \approx t' \quad C\langle s\langle\!\langle \bar{x}.\, t\sigma \rangle\!\rangle\rangle}{t \approx t' \quad C\langle s\langle\!\langle \bar{x}.\, t'\sigma \rangle\!\rangle_\eta\rangle \quad s\langle\!\langle \bar{x}.\, t\sigma \rangle\!\rangle \approx s\langle\!\langle \bar{x}.\, t'\sigma \rangle\!\rangle_\eta}\ \lambda\textsc{DemodExt}$$

where $s\langle\!\langle \bar{x}.\, t\sigma \rangle\!\rangle{\downarrow_{\beta\eta}}$ is a λ-expression or an applied variable. The term $t\sigma$ may refer to the bound variables $\bar{x}$. Side condition: The second premise is larger than ($\succ$) the second and third conclusion. This ensures that this premise is redundant with respect to these conclusions and may be removed. The double bar indicates that the conclusions collectively make the premises redundant and can replace them. The third conclusion, which is entailed by the extensionality axiom and $t \approx t'$, can be omitted if the corresponding extensionality axiom instance, for $\lambda x.\, s\langle\!\langle \bar{x}.\, t\sigma \rangle\!\rangle$ and $\lambda x.\, s\langle\!\langle \bar{x}.\, t'\sigma \rangle\!\rangle_\eta$, is smaller than the second premise. An instance of the rule, where $\mathsf{g}\,z$ is rewritten to $\mathsf{f}\,z\,z$ under a λ binder, follows:

$$\frac{\mathsf{g}\,x \approx \mathsf{f}\,x\,x \quad \mathsf{k}\,(\lambda z.\, \mathsf{h}\,(\mathsf{g}\,z)) \approx \mathsf{c}}{\mathsf{g}\,x \approx \mathsf{f}\,x\,x \quad \mathsf{k}\,(\lambda z.\, \mathsf{h}\,(\mathsf{f}\,z\,z)) \approx \mathsf{c} \quad (\lambda z.\, \mathsf{h}\,(\mathsf{g}\,z)) \approx (\lambda z.\, \mathsf{h}\,(\mathsf{f}\,z\,z))}\ \lambda\textsc{DemodExt}$$

Lemma 29. λDemodExt *is sound and preserves refutational completeness of our calculus.*

Proof. Soundness of the first conclusion is obvious. Soundness of the second and third conclusion follows from congruence using the two premises.

Preservation of completeness is justified by redundancy. In particular, we must justify the deletion of the second premise by showing that it is redundant with respect to the conclusions. By definition, it is redundant if for every ground instance $C\langle s\langle\!\langle \bar{x}.\, t\sigma \rangle\!\rangle\rangle\theta \in \mathcal{G}_\Sigma(C\langle s\langle\!\langle \bar{x}.\, t\sigma \rangle\!\rangle\rangle)$, its floor encoding $\lfloor C\langle s\langle\!\langle \bar{x}.\, t\sigma \rangle\!\rangle\rangle\theta \rfloor$ is entailed by $\lfloor \mathcal{G}_\Sigma(N) \rfloor$, where N are the conclusions of λDemodExt.

The first conclusion cannot help us to prove redundancy because $s\langle\!\langle \bar{x}.\, t\sigma \rangle\!\rangle\theta{\downarrow_{\beta\eta}}$ might be a λ-expression and then $\lfloor s\langle\!\langle \bar{x}.\, t\sigma \rangle\!\rangle\theta \rfloor$ is a constant that is completely unrelated to $\lfloor t\sigma\theta \rfloor$ in the floor logic. Instead, we use the θ-instances of the second and the third conclusion. By Lemma 8, $\lfloor C\langle s\langle\!\langle \bar{x}.\, t'\sigma \rangle\!\rangle_\eta\rangle\theta \rfloor$ has subterm $\lfloor s\langle\!\langle \bar{x}.\, t'\sigma \rangle\!\rangle_\eta\theta \rfloor$.

If this subterm is replaced by $\lfloor s\langle\!\langle \bar{x}.\, t\sigma \rangle\!\rangle \theta \rfloor$, we obtain $\lfloor C\langle s\langle\!\langle \bar{x}.\, t\sigma \rangle\!\rangle\rangle \theta \rfloor$. Thus, the floor encoding of the θ-instances of the second and the third conclusion entail the floor encoding of the θ-instance of the second premise by congruence. Due to the side condition that the second premise is larger than the second and third conclusion, by stability under substitutions, the θ-instances of the second and the third conclusion must be smaller than the θ-instance of the second premise. This shows that the second premise is redundant and can be removed. □

The next simplification rule can be used to prune arguments to variables that can be expressed as functions of the remaining arguments. For example, the clause $C[y\ \mathsf{a}\ \mathsf{b}\ (\mathsf{f}\ \mathsf{b}\ \mathsf{a}),\ y\ \mathsf{b}\ \mathsf{d}\ (\mathsf{f}\ \mathsf{d}\ \mathsf{b})]$, in which y occurs twice, can be simplified to $C[y'\ \mathsf{a}\ \mathsf{b},\ y'\ \mathsf{b}\ \mathsf{d}]$. The rule can also be used to remove the repeated arguments in $y\ \mathsf{b}\ \mathsf{b} \not\approx y\ \mathsf{a}\ \mathsf{a}$, the static argument a in $y\ \mathsf{a}\ \mathsf{c} \not\approx y\ \mathsf{a}\ \mathsf{b}$, and all four arguments in $y\ \mathsf{a}\ \mathsf{b} \not\approx z\ \mathsf{b}\ \mathsf{d}$. It is stated as

$$\frac{C}{C\{y \mapsto (\lambda \bar{x}_j.\, y'\ \bar{x}_{j-1})\}}\ \textsc{PruneArg}$$

where y' is a fresh variable, the minimum number k of arguments passed to any occurrence of y in the clause $C{\downarrow_{\beta\eta}}$ is at least j, and there exists a term t containing no variables bound in the clause such that $s_j = t\,\bar{s}_{j-1}\,s_{j+1} \ldots s_k$ for all terms of the form $y\ \bar{s}_k$ occurring in the clause. For example, clauses with a static argument correspond to the case $t := (\lambda \bar{x}_{j-1}\ x_{j+1} \ldots x_k.\ u)$, where u is the static argument (containing no variables bound in t) and j is its index in y's argument list. The repeated argument case corresponds to $t := (\lambda \bar{x}_{j-1}\, x_{j+1} \ldots x_k.\ x_i)$ where i is the index of the repeated argument's mate.

Lemma 30. PruneArg *is sound and preserves refutational completeness of our calculus.*

Proof. The rule is sound because it simply applies a substitution to C. It preserves completeness of our calculus because the premise is redundant with respect to the conclusion. This is because the sets of ground instances of premise and conclusion are the same. Let

$$\rho = \{y' \mapsto (\lambda \bar{x}_{j-1}\ x_{j+1} \ldots x_k.\ y\ \bar{x}_{j-1}\ (t\ \bar{x}_{j-1}\ x_{j+1} \ldots x_k)\ x_{j+1} \ldots x_k)\}$$

We will show that $C\{y \mapsto (\lambda \bar{x}_j.\, y'\ \bar{x}_{j-1})\}\rho = C$, which proves that the sets of ground instances of premise and conclusion are the same. Consider an occurrence of y in C. By the side conditions of PruneArg, any occurrence will have the form $y\ \bar{s}_k$ (with possibly more irrelevant arguments after that), where $s_j = t\ \bar{s}_{j-1}\ s_{j+1} \ldots s_k$. Hence,

$$\begin{aligned}(y\ \bar{s}_k)\{y \mapsto (\lambda \bar{x}_j.\, y'\ \bar{x}_{j-1})\}\rho &= (y'\ \bar{s}_{j-1}\ s_{j+1} \ldots s_k)\rho \\ &= y\ \bar{s}_{j-1}\ (t\ \bar{s}_{j-1}\ s_{j+1} \ldots s_k)\ s_{j+1} \ldots s_k \\ &= y\ \bar{s}_k\end{aligned}$$

Therefore, $C\{y \mapsto (\lambda \bar{x}_j.\, y'\ \bar{x}_{j-1})\}\rho = C$, which is what we needed to show. □

We created an algorithm that efficiently computes the correct value for term t in the definition of PruneArg. The algorithm will not find all possible values for the term t, but our informal tests show that it manages to discover most cases of prunable arguments that occur in practice.

The algorithm works by maintaining the mapping from functional variables and the indices of their expected arguments to the set of terms describing the ways i-th argument can be expressed using the remaining ones. For each occurence of a functional variable, set of terms expressing an argument using only the remaining ones is computed. Those sets are then intersected to determine whether there exists a t from the definition of PruneArg. Variables bound outside the scope of the occurence of the free variable are replaced by skolems that correspond to the de Bruijn index of the bound variable, for uniform handling of constants and loosely bound variables. For example, the occurence $\lambda.x\ 0\ \mathsf{a}\ (\lambda.\mathsf{f}\ 0)$ is skolemized to $\lambda.x\ \mathsf{sk}_0\ \mathsf{a}\ (\lambda.\mathsf{f}\ 0)$.

The following example clarifies the way in which the algorithm works. Suppose that $x\ \mathsf{a}\ (\mathsf{fa})\ \mathsf{b}$ and $\lambda.x\ 0\ (\mathsf{f0})\ \mathsf{b}$ are the only occurrences of the variable x. If we enumerate the arguments of x with 2,1 and 0 (following de Bruijn notation) and take ω to be the set of all terms that can be created for given signature, initial mapping is:

	2	1	0
x	ω	ω	ω

After computing the ways each argument can be computed using the remaining ones for the first occurence and intersecting the sets we get:

	2	1	0
x	$\{\mathsf{a}\}$	$\{\mathsf{f\ a}, \mathsf{f}\ 2\}$	$\{\mathsf{b}\}$

Lastly, after computing the corresponding sets for the second occurence we get

	2	1	0
x	$\emptyset$	$\{\mathsf{f}\ 2\}$	$\{\mathsf{b}\}$

Final value of the mapping shows us we can remove argument 1, since it functionally depends on argument 2. Similarly, we can remove argument 0, since its value is always b. Note that once the argument is removed, every occurence of it in the values of the mapping should be removed as well.

Following the literature [35, 62], we provide a rule for negative extensionality:

$$\frac{C \vee s \not\approx s'}{C \vee s\,(\mathsf{sk}\langle\bar{\alpha}_m\rangle\,\bar{y}_n) \not\approx s'\,(\mathsf{sk}\langle\bar{\alpha}_m\rangle\,\bar{y}_n)}\,\textsc{NegExt}$$

where sk is a fresh n-ary Skolem symbol, $\bar{\alpha}_m, \bar{y}_n$ are the type and term variables occurring free in the the literal $s \not\approx s'$, and $s \not\approx s'$ is eligible in the premise. Negative extensionality can also be applied as a simplification rule to all literals in the initial problem.

The next rule aims at creating opportunities for applying NegExt:

$$\frac{C \vee s[\bar{u}_n] \not\approx s[\bar{u}'_n]}{C \vee \bigvee_j u_j \not\approx u'_j}\,\textsc{NegCongFun}$$

Side conditions: The shared context $s[\,]$ is maximal, the type of at least one u_j is a function or a type variable, and $s[\bar{u}_n] \not\approx s[\bar{u}'_n]$ is eligible in the premise.

One reason why the extensionality axiom is so prolific is that both sides of its maximal literal, $y\,(\mathsf{diff}\langle\alpha, \beta\rangle\,y\,z) \not\approx z\,(\mathsf{diff}\langle\alpha, \beta\rangle\,y\,z)$, are fluid terms. An incomplete alternative is to omit the axiom and instead generate less explosive instances dynamically, based on the unapplied functions that occur at green positions in the problem. The rule is stated as

$$\frac{D\langle t\rangle \quad C\langle s\rangle}{(t\,(\mathsf{diff}\langle\alpha, \beta\rangle\,t\,s) \not\approx s\,(\mathsf{diff}\langle\alpha, \beta\rangle\,t\,s) \vee t \approx s)\sigma}\,\textsc{ExtInst}$$

where s, t are neither fluid nor variables, and σ is the most general unifier of s's and t's types and $\alpha \to \beta$, for fresh α, β. Moreover, the following restrictions apply, mimicking the unification performed by the calculus's core inference rules: (1) t occurs in a literal of the form $t \approx t'$ and Sup's conditions 5 to 9 are met; or (2) $C\langle s\rangle = D\langle t\rangle$, the terms s and t occur as subterms at the same green position p on opposite sides of the same literal, and EqRes's order restriction is satisfied for that literal; or (3) $C\langle s\rangle = D\langle t\rangle$, the terms s and t occur in different literals of the forms $s \approx v'$ and $t \approx v$, and EqFact's order and eligibility restrictions are satisfied for $u' := s$ and $u := t$. Intuitively, ExtInst generates extensionality instances that might help repair unification mismatches in the core rules. This approach is complete for first-order problems, since they do not need extensionality, and might be complete for other interesting fragments (e.g., second-order logic or ground higher-order logic).

Superposition can be generalized to orange subterms as follows:

$$\frac{D' \vee t \approx t' \quad C' \vee [\neg]\,s\langle\!\langle\bar{x}.\,u\rangle\!\rangle \approx s'}{(D' \vee C' \vee [\neg]\,s\langle\!\langle\bar{x}.\,t'\rangle\!\rangle_\eta \approx s')\sigma\rho}\,\lambda\textsc{Sup}$$

Sup's side conditions apply. We also require that $\bar{x}\sigma = \bar{x}$ and that the variables $\bar{x}$ do not occur in $y\sigma$ for all variables y in u. Moreover, let $P_y = \{y\}$ for all type and term variables $y \notin \bar{x}$. For each i, let P_{x_i} be recursively defined as the union

of all P_y such that y occurs free in the λ-expression that binds x_i in $s\langle\!\langle \bar{x}.\, u \rangle\!\rangle\sigma$ or that occurs free in the corresponding subterm of $s\langle\!\langle \bar{x}.\, t' \rangle\!\rangle_\eta\sigma$. The substitution ρ is defined as $\{x_i \mapsto \mathsf{sk}_i\langle\bar{\alpha}_i\rangle\, \bar{y}_i \text{ for each } i\}$, where $\bar{y}_i$ are the term variables in P_{x_i} and $\bar{\alpha}_i$ are the type variables in P_{x_i} and the type variables occurring in the type of the λ-expression binding x_i. This substitution introduces Skolem terms to represent bound variables that would otherwise escape their binders. The rule can be justified in terms of paramodulation and extensionality, with the Skolem terms standing for diff terms. An instance of the rule follows:

$$\frac{n \approx \mathsf{zero} \vee \mathsf{div}\ n\ n \approx \mathsf{one} \quad \mathsf{prod}\ K\ (\lambda k.\ \mathsf{div}\ (\mathsf{succ}\ k)\ (\mathsf{succ}\ k)) \not\approx \mathsf{one}}{\mathsf{succ}\ \mathsf{sk} \approx \mathsf{zero} \vee \mathsf{prod}\ K\ (\lambda k.\ \mathsf{one}) \not\approx \mathsf{one}}\ \lambda\textsc{Sup}$$

Intuitively, the term $\mathsf{prod}\ K\ (\lambda k.\ u)$ is intended to denote the product $\prod_{k\in K} u$, where k ranges over a finite set K of natural numbers.

Lemma 31. λSup *is satisfiability-preserving.*

Proof. Let N be a clause set and E be the conclusion of a λSup inference from N. It suffices to show that if N is satisfiable, then $N \cup \{E\}$ is satisfiable, because the converse is trivial. Let $\mathcal{I}_1$ be a model of N. We need to construct a model of $N \cup \{E\}$. By the definition of models, the extensionality axiom holds in $\mathcal{I}_1$:

$$\mathcal{I}_1 \models y\ (\mathsf{diff}\langle\alpha,\beta\rangle\ y\ z) \not\approx z\ (\mathsf{diff}\langle\alpha,\beta\rangle\ y\ z) \vee y \approx z$$

In the following, we will omit the type arguments of diff, since they can be derived from the type of the arguments.

For each i, let v_i be the λ-expression binding x_i in the term $s\langle\!\langle \bar{x}.\, u \rangle\!\rangle\sigma$ in the λSup rule. Let v_i' the corresponding term, in which the relevant occurrence of $u\sigma$ is replaced by $t'\sigma$. We define a substitution π recursively as

$$x_i\pi = \mathsf{diff}\ (v_i\pi)\ (v_i'\pi) \text{ for all } i$$

We extend the model $\mathcal{I}_1$ further to a model $\mathcal{I}_2$, interpreting the symbol sk_i such that $\mathcal{I}_2 \models \mathsf{sk}_i\langle\bar{\alpha}_i\rangle\, \bar{y}_i \approx \mathsf{diff}\ (v_i\pi)\ (v_i'\pi)$ for each i. The arguments of sk_i have been defined to coincide with the free type and term variables in $\mathsf{diff}\ (v_i\pi)\ (v_i'\pi)$, allowing us to extend $\mathcal{I}_1$ to $\mathcal{I}_2$ in this way.

By assumption, the premises of our λSup inference are true in $\mathcal{I}_1$ and hence in $\mathcal{I}_2$. We want to show that the conclusion $(D' \vee C' \vee [\neg]\, s\langle\!\langle \bar{x}.\, t' \rangle\!\rangle_\eta \approx s')\sigma\rho$ is also true in $\mathcal{I}_2$. Let ξ be a valuation. If $\mathcal{I}_2, \xi \models (D' \vee C')\sigma\rho$, we are done. So we assume that $D'\sigma\rho$ and $C'\sigma\rho$ are false in $\mathcal{I}_2$ under ξ. In the following, we omit '$\mathcal{I}_2, \xi \models$', but all equations ($\approx$) are meant to be true in $\mathcal{I}_2$ under ξ.

Assuming $D'\sigma\rho$ and $C'\sigma\rho$ are false, the premises imply that $t\sigma\rho \approx t'\sigma\rho$ and $[\neg]\, s\langle\!\langle \bar{x}.\, u \rangle\!\rangle\sigma\rho \approx s'\sigma\rho$. Due to the way we extended $\mathcal{I}_1$ to $\mathcal{I}_2$, we have $\mathcal{I}_2 \models w\pi \approx w\rho$ for any term w. Hence, we have $t\sigma\pi \approx t'\sigma\pi$. By congruence, this and $t\sigma = u\sigma$ imply that $v_k\pi\ (\mathsf{diff}\ (v_k\pi)\ (v_k'\pi)) \approx v_k'\pi\ (\mathsf{diff}\ (v_k\pi)\ (v_k'\pi))$, where k is the length of the tuple $\bar{x}$, i.e., v_k is the innermost λ-expression surrounding u. The extensionality axiom then implies $v_k\pi \approx v_k'\pi$.

It follows directly from the definition of π that for all i

$$v_i\pi\ (\mathsf{diff}\ (v_i\pi)\ (v'_i\pi)) = s_i\langle\!\langle v_{i+1}\pi\rangle\!\rangle$$
$$v'_i\pi\ (\mathsf{diff}\ (v_i\pi)\ (v'_i\pi)) = s_i\langle\!\langle v'_{i+1}\pi\rangle\!\rangle$$

for some context $s_i\langle\!\langle\ \rangle\!\rangle$. The subterms $v_{i+1}\pi$ of $s_i\langle\!\langle v_{i+1}\pi\rangle\!\rangle$ and $v'_{i+1}\pi$ of $s_i\langle\!\langle v'_{i+1}\pi\rangle\!\rangle$ may be below applied variables but not below λ-expressions. Since substitutions avoid capture, in v_i and v'_i, π only substitutes x_j with $j < i$, but in v_{i+1} and v'_{i+1}, it substitutes all x_j with $j \leq i$. By an induction using these equations, congruence and the extensionality axiom, we can derive from $v_k\pi \approx v'_k\pi$ that $v_1\pi \approx v'_1\pi$. Since $\mathcal{I}_2 \models w\pi \approx w\rho$ for any term w, we have $v_1\rho \approx v'_1\rho$. By congruence, it follows that $s\langle\!\langle\bar{x}.\ u\rangle\!\rangle\sigma\rho \approx s\langle\!\langle\bar{x}.\ t'\rangle\!\rangle_\eta\sigma\rho$. With $[\neg]\, s\langle\!\langle\bar{x}.\ u\rangle\!\rangle\sigma\rho \approx s'\sigma\rho$, it follows that $([\neg]\, s\langle\!\langle\bar{x}.\ t'\rangle\!\rangle_\eta \approx s')\sigma\rho$.

Hence, the conclusion of the λSUP inference is true in $\mathcal{I}_2$. □

Finally, *duplicating superposition* is a lightweight alternative to FLUIDSUP:

$$\frac{D' \vee t \approx t' \quad C' \vee [\neg]\, s\langle y\ \bar{u}_n\rangle \approx s'}{(D' \vee C' \vee [\neg]\, s\langle z\ \bar{u}_n\ t'\rangle \approx s')\rho\sigma}\ \textsc{DupSup}$$

where $n \geq 1$, $\rho = \{y \mapsto \lambda\bar{x}_n.\ z\ \bar{x}_n\ (w\ \bar{x}_n)\}$, and $\sigma \in \mathrm{CSU}(t, w\ \bar{u}_n\rho)$ for fresh variables w, z. The order and eligibility restrictions are as for SUP. The rule can be understood as the composition of an inference that applies ρ and of a paramodulation inference into the subterm $w\ \bar{u}_n\rho$ of $s\langle z\ \bar{u}_n\rho\ (w\ \bar{u}_n\rho)\rangle$. DUPSUP is general enough to replace FLUIDSUP in Examples 4 and 5 but not in Example 6. We conjecture that DUPSUP, in conjunction with an extended SUP rule that considers the green subterms of $\bar{u}_n$, constitutes a complete alternative to FLUIDSUP for fluid subterms of the form $y\ \bar{u}_n$ if the types of t, all u_j, and $y\ \bar{u}_n$ are not functions or type variables.

6 Implementation

Zipperposition [27,28] is an open source superposition prover written in OCaml.[2] Originally designed for polymorphic first-order logic (TF1 [20]), it was later extended with an incomplete higher-order mode based on pattern unification [51]. Bentkamp et al. [11] extended it further with a complete λ-free higher-order mode. As a prototype, we have now implemented a Boolean-free higher-order mode based on our calculus.

We use a metaorder induced by a λ-free KBO [9]. We use weight 1 for lam and db_i with the precedence $\mathsf{lam} \succ \cdots \succ \mathsf{db}_k \succ \cdots \succ \mathsf{db}_1 \succ \mathsf{db}_0$. We currently use $\succeq$ as the nonstrict term order but could improve precision by employing a more precise computable approximation of $\succsim$.

Except for FLUIDSUP, the core calculus rules already existed in Zipperposition in a similar form. To retrieve candidate right premises for FLUIDSUP, we created

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

an index of all fluid green subterms in the active clause set. Among the proposed higher-order optional rules, we implemented NegExt, λSup, a mildly incomplete variant of λDemodExt without the third conclusion, and a variant of the Prune-Arg rule that removes most functional dependencies that occur in practice.

For unification, we started with Jensen and Pietrzykowski's procedure [39]. The procedure is not ideal because it computes a nonminimal set of unifiers; for example, given the flex–flex constraint $y\,\mathsf{a} \stackrel{?}{=} z\,\mathsf{b}$, it generates not only the most general unifier $\{y \mapsto (\lambda w.\ y'\,w\,\mathsf{b}),\ z \mapsto y'\,\mathsf{a}\}$ but also infinitely many superfluous unifiers. It is not clear whether Snyder and Gallier's procedure [61] would behave better. To support polymorphism, we extended Jensen and Pietrzykowski's projection rule to check type unifiability instead of equality and their iteration rule to consider the possibility that a type variable is instantiated with a function type. On the other hand, polymorphism allows us to avoid the enumeration of types in the iteration rule.

To interleave the unification with other computation, our unification procedure returns a possibly infinite stream of subsingletons (sets of cardinality 0 or 1) computed on demand. It can even cope with nonterminating unification problems that do not yield any unifiers, by representing them as an infinite stream of empty sets. We use this procedure for inference rules, keeping simpler pattern-style unification for simplification rules. The inference rules turn the possibly infinite streams of unifiers into possibly infinite streams of clauses—the conclusions of inferences. To consume these streams fairly while giving flexibility to heuristics, we designed a priority queue that associates a weight with each stream. This queue is used in the main given clause loop to store new streams resulting from inferences and to extract clauses, which are then moved to the passive clause set.

The implemented heuristic extracts as many clauses from the stream queue as there are streams in the queue. In the first order case all streams have exactly one element and will hence be worked off immediately, yielding a graceful strategy. In addition, to ensure fairness of both heuristics, every sixth iteration of the main loop, a clause is extracted from all streams in the queue (The number 6 was arbitrarily chosen). During the other five iterations, the heuristic proceeds in the following way: determine the stream with the lowest weight and extract a clause from it; increase the stream weight by a penalty and put it back in the queue; repeat until you have enough clauses or no stream remains in the queue. Empty streams are discarded.

The penalty used to increase the stream weight when a clause is extracted is 1 for most streams, but higher for streams resulting from FluidSup inferences to tame the explosion of this rule. Moreover, clauses such as the extensionality clause can also carry a penalty, which decreases the chances of being selected as the given clause. This clause penalty is passed on to streams and clauses when performing inferences.

Based on informal experiments, we developed or tuned a few general heuristics of Zipperposition. Definition unfolding, in conjunction with β-reduction, transforms many higher-order TPTP problems into first-order problems. We also modified KBO's weight generation scheme to take symbol frequencies into

account and modified other heuristics to prioritize clauses containing symbols present in the conjecture.

7 Evaluation

We evaluated our prototype implementation of the calculus in Zipperposition with other higher-order provers and with Zipperposition's modes for less expressive logics. All of the experiments presented in this section were performed on StarExec nodes equipped with Intel Xeon E5-2609 0 CPUs clocked at 2.40 GHz. Provers were invoked with a CPU time limit of 300 s. The raw data are available online.[3]

We used both standard TPTP benchmarks [63] and Sledgehammer-generated benchmarks. From the TPTP, we selected all 709 TFF (monomorphic and polymorphic first-order) problems without arithmetic and all 597 TH0 (monomorphic higher-order) problems without first-class Booleans and arithmetic. We partitioned the TH0 problems into those containing no λs (TH0λf, 545 problems) and those containing λs (TH0λ, 52 problems). The Sledgehammer benchmarks, corresponding to Isabelle's Judgment Day suite [23], were regenerated to target Boolean-free higher-order logic. They comprise 1253 problems, 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 is further divided into two subgroups based on the processing of λ-expressions: SH256-λ and SH16-λ preserve λ-expressions, whereas SH256-ll and SH16-ll encode them as λ-lifted supercombinators [50] to make the problems accessible to λ-free higher-order provers.

We chose Leo-III 1.3 and Satallax 3.3 as representatives of the state of the art. These are cooperative higher-order provers that can be set up to regularly invoke first-order provers as terminal proof procedures. Leo-III can be used on its own or as a metaprover (Leo-III-meta) with CVC4, E, and iProver as backends. Satallax can be used on its own or as a metaprover (Satallax-meta) with E. We also included Ehoh [66], the λ-free higher-order mode of E 2.3. For Zipperposition, we included its first-order and λ-free modes (FOZip and λfreeZip) as well as a mode that performs an applicative encoding [66, Section 2] before invoking the first-order mode (@+FOZip). We experimented with three variants of our calculus implementation. λZip-full is designed to be refutationally complete. λZip-pragmatic disables FluidSup and the extensionality axiom, and uses a lightweight higher-order unification algorithm instead of Jensen and Pietrzykowski's procedure. Finally, λZip-competitive is a variant of λZip-pragmatic that is further tuned for small problems requiring a substantial amount of higher-order reasoning.

A summary of our experiments is presented in Figure 1. To enhance readability, we highlight in bold the winning system for each column *excluding the metaprovers*. We observe that Leo-III-meta emerges as winner on all benchmark sets, but λZip-pragmatic and λZip-competitive compare very well with Leo-III and Satallax. In contrast, λZip-full cannot seem to keep its FluidSup rule and extensionality under control. More research into heuristics design appears necessary.

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

	TFF	TH0λf	TH0λ	SH256-ll	SH16-ll	SH256-λ	SH16-λ
Leo-III	85	387	**42**	234	323	228	338
Satallax	–	400	**42**	495	371	516	384
Ehoh	–	396	–	**671**	397	–	–
FOZip	**238**	–	–	–	–	–	–
@+FOZip	194	398	–	495	389	–	–
λfreeZip	233	401	–	603	**401**	–	–
λZip-full	178	388	27	394	351	385	348
λZip-pragmatic	227	416	27	560	386	**567**	**387**
λZip-competitive	216	**418**	40	413	351	399	357
Leo-III-meta	252	438	44	706	412	688	416
Satallax-meta	–	427	42	491	372	513	385

Figure 1: Number of proved problems

It is disappointing that on Sledgehammer problems (SH256 and SH16), we obtain better performance by using λfreeZip with λ-lifting than using λZip with native λs. On TH0λf problems, the situation is reversed. This seems to suggest that λ reasoning is rarely needed for Sledgehammer problems. Clearly, this is another area where research into heuristics design could be beneficial.

8 Discussion and Related Work

Bentkamp et al. [11] introduced four calculi for λ-free higher-order logic organized along two axes: *intensional* versus *extensional*, and *nonpurifying* versus *purifying*. The purifying calculi flatten the clauses containing applied variables, thereby eliminating the need for superposition into variables. As we extended their work to support λs, we found the purification approach problematic and quickly gave it up because it needs x to be smaller than $x\,t$, which is impossible to achieve with a term order on $\beta\eta$-equivalence classes. We also gave up our attempt at supporting intensional higher-order logic. Extensionality is the norm for higher-order unification [31] and is employed in the TPTP THF format [64] and in proof assistants such as HOL4, HOL Light, Isabelle/HOL, Lean, Nuprl, and PVS. Bentkamp et al. viewed their approach as "a stepping stone towards full higher-order logic." It already included a notion analogous to green subterms and an ArgCong rule, which help cope with the complications occasioned by β-reduction.

Our superposition calculus joins the family of proof systems for higher-order logic. Closely related are Andrews's higher-order resolution [1], Huet's constrained resolution [37], Jensen and Pietrzykowski's ω-resolution [39], Snyder's higher-order E-resolution [60], Benzmüller and Kohlhase's extensional higher-order resolution [14], and Benzmüller's higher-order unordered paramodulation and RUE resolution [13]. A noteworthy variant is Steen and Benzmüller's higher-order ordered paramodulation [62], whose order restrictions undermine refutational completeness but yield good empirical results. Other approaches are based on

analytic tableaux [7, 44, 45, 56], connections [2], sequents [48], and satisfiability modulo theories [8]. Andrews [3] and Benzmüller and Miller [15] provide excellent surveys of higher-order automation.

The main advantage of our calculus is that it gracefully generalizes the highly successful first-order superposition rules without sacrificing refutational completeness. It also includes a powerful simplification rule, PruneArg, that could be useful in other provers. Among the drawbacks of our approach are the need to solve flex–flex pairs eagerly and the explosion caused by the extensionality axiom. We believe that this is a reasonable trade-off, especially for large problems with a substantial first-order component, such as those originating from proof assistants.

Our prototype λZipperposition joins the league of higher-order automatic theorem provers. We briefly list some of its rivals. TPS [4] is based on the connection method and expansion proofs. LEO [14] and Leo-II [17] implement variants of RUE resolution. Leo-III [62] is based on higher-order paramodulation. Satallax [24] implements a higher-order tableau calculus guided by a SAT solver. Leo-II, Leo-III, and recent versions of Satallax integrate first-order provers as terminal procedures. AgsyHOL [48] is based on a focused sequent calculus guided by narrowing. Finally, there is ongoing work by the developers of CVC4, veriT, and Vampire to extend their provers to higher-order logic [8, 18].

Half a century ago, Robinson [57] proposed to reduce higher-order logic to first-order logic via a translation. Tools such as Sledgehammer [55], Miz$\mathbb{AR}$ [65], HOLyHammer [43], and CoqHammer [29] have since popularized this approach in proof assistants. Such translations must eliminate the λ-expressions, typically using SKBCI combinators or λ-lifting [50], and encode typing information [19]. Most translations are implemented outside provers, but hybrid approaches are also possible [30]. For example, the Vampire developers are experimenting with a combinator-based representation of higher-order unifiers [18], which would allow them to reuse most first-order data structures and algorithms unchanged.

9 Conclusion

We presented a superposition calculus for a Boolean-free fragment of extensional polymorphic higher-order logic. With the notable exception of a functional extensionality axiom, it gracefully generalizes standard superposition. Our prototype prover Zipperposition shows promising results on TPTP and Isabelle benchmarks. In future work, we plan to pursue five main avenues of investigation.

We first plan to *extend the calculus to support Booleans and Hilbert choice.* Booleans are notoriously explosive. We want to experiment with both axiomatizations and native support in the calculus. Native support would likely take the form of a primitive substitution rule that enumerates predicate instantiations [2], delayed clausification rules [33], and rules for reasoning about Hilbert choice.

We want to investigate techniques to *curb the explosion caused by functional extensionality.* The extensionality axiom reintroduces the search space explosion that the calculus's order restrictions aim at avoiding. Maybe we can replace it by more restricted inference rules without compromising refutational completeness.

We will also look into approaches to *curb the explosion caused by higher-order unification.* Our calculus suffers because it needs to solve flex–flex pairs. Existing procedures [39, 61] enumerate redundant unifiers. This can probably be avoided to some extent. It could also be interesting to investigate unification algorithms that would delay imitation/projection choices via special schematic variables, inspired by Libal's concise representation of regular unifiers [47].

We clearly need to *fine-tune and develop heuristics.* We expect heuristics to be a fruitful area for future research in higher-order reasoning. Proof assistants are an inexhaustible source of easy-looking benchmarks that are beyond the power of today's provers. Whereas "hard higher-order" may remain forever out of reach, there is a substantial "easy higher-order" fragment that awaits automation.

Finally, we plan to *implement the calculus in a state-of-the-art prover.* A suitable basis for an optimized implementation of our calculus would be Ehoh, the λ-free higher-order version of the E prover developed by Vukmirović et al. [66].

Acknowledgment. Simon Cruanes patiently explained Zipperposition's internals and allowed us to continue the development of his prover. Christoph Benzmüller and Alexander Steen shared insights and examples with us, guiding us through the literature and clarifying how the Leos work. Maria Paola Bonacina and Nicolas Peltier gave us some ideas on how to treat the extensionality axiom as a theory axiom, ideas we have yet to explore. Mathias Fleury helped us set up regression tests for Zipperposition. Ahmed Bhayat, Tomer Libal, and Enrico Tassi shared their insights on higher-order unification. Andrei Popescu and Dmitriy Traytel explained the terminology surrounding the λ-calculus. Haniel Barbosa, Daniel El Ouraoui, Pascal Fontaine, and Hans-Jörg Schurr were involved in many stimulating discussions. Christoph Weidenbach made this collaboration possible. Ahmed Bhayat, Mark Summerfield, and the anonymous reviewers suggested several textual improvements. We thank them all.

Bentkamp, Blanchette, and Vukmirović'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). Bentkamp and Blanchette also benefited from the Netherlands Organization for Scientific Research (NWO) Incidental Financial Support scheme. Blanchette has received funding from the NWO under the Vidi program (project No. 016.Vidi.189.037, Lean Forward).

References

[1] Andrews, P.B.: Resolution in type theory. J. Symb. Log. 36(3), 414–432 (1971)
[2] Andrews, P.B.: On connections and higher-order logic. J. Autom. Reason. 5(3), 257–291 (1989)
[3] 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)
[4] Andrews, P.B., Bishop, M., Issar, S., Nesmith, D., Pfenning, F., Xi, H.: TPS: A theorem-proving system for classical type theory. J. Autom. Reason. 16(3), 321–353 (1996)
[5] Bachmair, L., Ganzinger, H.: Rewrite-based equational theorem proving with selection and simplification. J. Log. Comput. 4(3), 217–247 (1994)

[6] 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)
[7] Backes, J., Brown, C.E.: Analytic tableaux for higher-order logic with choice. J. Autom. Reason. 47(4), 451–479 (2011)
[8] Barbosa, H., Reynolds, A., Fontaine, P., Ouraoui, D.E., Tinelli, C.: Higher-order SMT solving (work in progress). In: Dimitrova, R., D'Silva, V. (eds.) SMT 2018 (2018)
[9] 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)
[10] 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`
[11] 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)
[12] Bentkamp, A., Blanchette, J.C., Cruanes, S., Waldmann, U.: Superposition for lambda-free higher-order logic (technical report). Technical report (2018), `http://matryoshka.gforge.inria.fr/pubs/lfhosup_report.pdf`
[13] Benzmüller, C.: Extensional higher-order paramodulation and RUE-resolution. In: Ganzinger, H. (ed.) CADE-16. LNCS, vol. 1632, pp. 399–413. Springer (1999)
[14] Benzmüller, C., Kohlhase, M.: Extensional higher-order resolution. In: Kirchner, C., Kirchner, H. (eds.) CADE-15. LNCS, vol. 1421, pp. 56–71. Springer (1998)
[15] 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)
[16] Benzmüller, C., Paulson, L.C.: Multimodal and intuitionistic logics in simple type theory. Log. J. IGPL 18(6), 881–892 (2010)
[17] Benzmüller, C., Sultana, N., Paulson, L.C., Theiss, F.: The higher-order prover Leo-II. J. Autom. Reason. 55(4), 389–404 (2015)
[18] Bhayat, A., Reger, G.: Set of support for higher-order reasoning. In: Konev, B., Urban, J., Rümmer, P. (eds.) PAAR-2018. CEUR Workshop Proceedings, vol. 2162, pp. 2–16. CEUR-WS.org (2018)
[19] Blanchette, J.C., Böhme, S., Popescu, A., Smallbone, N.: Encoding monomorphic and polymorphic types. Log. Meth. Comput. Sci. 12(4) (2016)
[20] 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)
[21] 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)
[22] Blanqui, F., Jouannaud, J.P., Rubio, A.: The computability path ordering. Log. Meth. Comput. Sci. 11(4) (2015)
[23] Böhme, S., Nipkow, T.: Sledgehammer: Judgement Day. In: Giesl, J., Hähnle, R. (eds.) IJCAR 2010. LNCS, vol. 6173, pp. 107–121. Springer (2010)
[24] 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)
[25] de Bruijn, N.G.: Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church–Rosser theorem. Indag. Math 75(5), 381–392 (1972)

[26] Cervesato, I., Pfenning, F.: A linear spine calculus. J. Log. Comput. 13(5), 639–688 (2003)
[27] Cruanes, S.: Extending Superposition with Integer Arithmetic, Structural Induction, and Beyond. Ph.D. thesis, École polytechnique (2015)
[28] Cruanes, S.: Superposition with structural induction. In: Dixon, C., Finger, M. (eds.) FroCoS 2017. LNCS, vol. 10483, pp. 172–188. Springer (2017)
[29] Czajka, Ł., Kaliszyk, C.: Hammer for Coq: Automation for dependent type theory (2018)
[30] Dougherty, D.J.: Higher-order unification via combinators. Theor. Comput. Sci. 114(2), 273–298 (1993)
[31] Dowek, G.: Higher-order unification and matching. In: Robinson, J.A., Voronkov, A. (eds.) Handbook of Automated Reasoning, vol. II, pp. 1009–1062. Elsevier and MIT Press (2001)
[32] Fitting, M.: Types, Tableaus, and Gödel's God. Kluwer (2002)
[33] Ganzinger, H., Stuber, J.: Superposition with equivalence reasoning and delayed clause normal form transformation. Information and Computation 199(1–2), 3–23 (2005)
[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] Huet, G.P.: A mechanization of type theory. In: Nilsson, N.J. (ed.) IJCAI-73. pp. 139–146. William Kaufmann (1973)
[38] Huet, G.P.: A unification algorithm for typed lambda-calculus. Theor. Comput. Sci. 1(1), 27–57 (1975)
[39] Jensen, D.C., Pietrzykowski, T.: Mechanizing ω-order type theory through unification. Theor. Comput. Sci. 3(2), 123–171 (1976)
[40] Jouannaud, J.P., Rubio, A.: Rewrite orderings for higher-order terms in eta-long beta-normal form and recursive path ordering. Theor. Comput. Sci. 208(1–2), 33–58 (1998)
[41] Jouannaud, J.P., Rubio, A.: Polymorphic higher-order recursive path orderings. J. ACM 54(1), 2:1–2:48 (2007)
[42] Kaliszyk, C., Sutcliffe, G., Rabe, F.: TH1: The TPTP typed higher-order form with rank-1 polymorphism. In: Fontaine, P., Schulz, S., Urban, J. (eds.) PAAR-2016. CEUR Workshop Proceedings, vol. 1635, pp. 41–55. CEUR-WS.org (2016)
[43] Kaliszyk, C., Urban, J.: HOL(y)Hammer: Online ATP service for HOL Light. Math. Comput. Sci. 9(1), 5–22 (2015)
[44] Kohlhase, M.: Higher-order tableaux. In: Baumgartner, P., Hähnle, R., Posegga, J. (eds.) TABLEAUX '95. LNCS, vol. 918, pp. 294–309. Springer (1995)
[45] Konrad, K.: HOT: A concurrent automated theorem prover based on higher-order tableaux. In: Grundy, J., Newey, M.C. (eds.) TPHOLs '98. LNCS, vol. 1479, pp. 245–261. Springer (1998)
[46] 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)
[47] Libal, T.: Regular patterns in second-order unification. In: Felty, A.P., Middeldorp, A. (eds.) CADE-25. LNCS, vol. 9195, pp. 557–571. Springer (2015)
[48] 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)

[49] Mayr, R., Nipkow, T.: Higher-order rewrite systems and their confluence. Theor. Comput. Sci. 192(1), 3–29 (1998)
[50] Meng, J., Paulson, L.C.: Translating higher-order clauses to first-order clauses. J. Autom. Reason. 40(1), 35–60 (2008)
[51] Miller, D.: A logic programming language with lambda-abstraction, function variables, and simple unification. J. Log. Comput. 1(4), 497–536 (1991)
[52] Miller, D.A.: A compact representation of proofs. Studia Logica 46(4), 347–370 (1987)
[53] Nieuwenhuis, R., Rubio, A.: Basic superposition is complete. In: Krieg-Brückner, B. (ed.) ESOP '92. LNCS, vol. 582, pp. 371–389. Springer (1992)
[54] 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)
[55] Paulson, L.C., Blanchette, J.C.: Three years of experience with Sledgehammer, a practical link between automatic and interactive theorem provers. In: Sutcliffe, G., Schulz, S., Ternovska, E. (eds.) IWIL-2010. EPiC, vol. 2, pp. 1–11. EasyChair (2012)
[56] Robinson, J.: Mechanizing higher order logic. In: Meltzer, B., Michie, D. (eds.) Machine Intelligence, vol. 4, pp. 151–170. Edinburgh University Press (1969)
[57] 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)
[58] Schlichtkrull, A., Blanchette, J.C., Traytel, D., Waldmann, U.: Formalizing Bachmair and Ganzinger's ordered resolution prover. In: Galmiche, D., Schulz, S., Sebastiani, R. (eds.) IJCAR 2018. LNCS, vol. 10900, pp. 89–107. Springer (2018)
[59] 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)
[60] Snyder, W.: Higher order E-unification. In: Stickel, M.E. (ed.) CADE-10. LNCS, vol. 449, pp. 573–587. Springer (1990)
[61] Snyder, W., Gallier, J.H.: Higher-order unification revisited: Complete sets of transformations. J. Symb. Comput. 8(1/2), 101–140 (1989)
[62] 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)
[63] Sutcliffe, G.: The TPTP problem library and associated infrastructure—from CNF to TH0, TPTP v6.4.0. J. Autom. Reason. 59(4), 483–502 (2017)
[64] 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)
[65] Urban, J., Rudnicki, P., Sutcliffe, G.: ATP and presentation service for Mizar formalizations. J. Autom. Reason. 50(2), 229–241 (2013)
[66] 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. LNCS, Springer (2019)
[67] 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`
[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)