A Combinator-Based Superposition Calculus for Higher-Order Logic (Technical Report)

Ahmed Bhayat[0000−0002−1343−5084] and Giles Reger[0000−0001−6353−952X]

University of Manchester, Manchester, UK

Abstract. We present a refutationally complete superposition calculus for a version of higher-order logic based on the combinatory calculus. We also introduce a novel method of dealing with extensionality. The calculus was implemented in the Vampire theorem prover and we test its performance against other leading higher-order provers. The results suggest that the method is competitive.

1 Introduction

First-order superposition provers are often used to reason about problems in extensional higher-order logic (HOL) [18,26]. Commonly, this is achieved by translating the higher-order problem to first-order logic (FOL) using combinators. Such a strategy is sub-optimal as translations generally sacrifice completeness and at times even soundness. In this paper, we provide a modification of first-order superposition that is sound and complete for a combinatory version of HOL. Moreover, it is *graceful* in the sense of that it coincides with standard superposition on purely first-order problems.

The work is complementary to the lambda superposition calculus of Bentkamp et al. [4]. Our approach appears to offer two clear differences. Firstly, as our calculus is based on the combinatory logic and first-order unification, it is far closer to standard first-order superposition. Therefore, it should be easier to implement in state-of-the-art first-order provers. Secondly, the $>_{\mathsf{ski}}$ ordering that we propose to parameterise our calculus with can compare more terms than can be compared by the ordering presented in [4]. On the other hand, we suspect that for problems requiring complex unifiers, our approach will not be competitive with lambda superposition.

Developing a complete and efficient superposition calculus for a combinatory version of HOL poses some difficulties. When working with a monomorphic logic it is impossible to select a finite set of typed combinator axioms that can guarantee completeness for a particular problem [10]. Secondly, using existing orderings, combinator axioms can superpose among themselves, leading to a huge number of consequences of the axioms. If the problem is first-order, these consequences can never interact with non-combinator clauses and are therefore useless.

We deal with both issues in the current work. To circumvent the first issue, we base our calculus on a polymorphic rather than monomorphic first-order logic. The second issue can be dealt with by an ordering that orients combinator axioms left-to-right. Consider the S-combinator axiom $\mathsf{S}\,x\,y\,z \approx x\,z\,(y\,z)$. Assume that there exists a simplification ordering $\succ$ such that $\mathsf{S}\,x\,y\,z \succ x\,z\,(y\,z)$. Then, since superposition is only carried out on the larger side of literals and not at variables, there can be no

inferences between the **S**-axiom and any other combinator axiom. Indeed, in this case the axioms can be removed from the clause set altogether and replaced by an inference rule (Section 8).

No ground-total simplification ordering is known that is capable of orienting all axioms for a complete set of combinators.[1] The authors suspect that no such simplification ordering exists. Consider a KBO-like ordering. Since, the variable x appears twice on the right-hand side of the **S**-axiom and only once on the left-hand side, the ordering would not be able to orient it. The same is the case for any other combinator which duplicates its arguments.

In other, as yet unpublished, work [9], we have developed an ordering that enjoys most of the properties of a simplification ordering, but lacks full compatibility with contexts. In particular, the ordering is not compatible with what we call *unstable* contexts. We propose using such an ordering to parameterise the superposition calculus. In the standard proof of the completeness of superposition, compatibility with contexts is used to rule out the need for superposition at or beneath variables. As the ordering doesn't enjoy full compatibility with contexts, limited superposition at and below variables needs to be carried out. This is dealt with by the addition of an extra inference rule to the standard rules of superposition, which we call SUBVARSUP (Section 3).

By turning combinator axioms into rewrite rules, the calculus represents a folding of higher-order unification into the superposition calculus itself. Whilst not as goal-directed as a dedicated higher-order unification algorithm, it is still far more goal-directed than using **SK**-style combinators in superposition provers along with standard orderings. Consider the conjecture $\exists z.\, \forall xy.\, z\, x\, y \approx \mathsf{f}\, y\, x$. Bentkamp et al. ran an experiment and found that the E prover [24] running on this conjecture supplemented with the **S**- and **K**-combinator axioms had to perform 3756 inferences in order to find a refutation [4]. Our calculus reduces this number to 427 inferences. With the addition of rewrite rules for **C**-, **B**- and **I**-combinators, the required inferences reduces to 18.

We consider likely that for problems requiring 'simple' unifiers, folding unification into superposition will be competitive with higher-order unification whilst providing the advantages that data structures and algorithms developed for first-order superposition can be re-used unchanged. The results of the empirical evaluation of our method can be found in Section 9.

2 The Logic

The logic we use is polymorphic applicative first-order logic otherwise known as λ-free (clausal) higher-order logic.

Syntax Let $\mathcal{V}_{\mathsf{ty}}$ be a set of type variables and Σ_{ty} be a set of type constructors with fixed arities. It is assumed that a binary type constructor $\to$ is present in Σ_{ty} which is written infix. The set of types is defined:

Polymorphic Types $\quad \tau ::= \kappa(\overline{\tau_n}) \mid \alpha \mid \tau \to \tau \qquad$ where $\alpha \in \mathcal{V}_{\mathsf{ty}}$ and $\kappa \in \Sigma_{\mathsf{ty}}$

[1] A complete set of combinators is a set of combinators whose members can be composed to form a term extensionally equivalent to any given λ-term.

The notation $\overline{t_n}$ is used to denote a tuple or list of types or terms depending on the context. A type declaration is of the form $\Pi\,\overline{\alpha_n}\,.\,\sigma$ where σ is a type and all type variables in σ appear in $\overline{\alpha}$. Let Σ be a set of typed function symbols and $\mathcal{V}$ a set of variables with associated types. It is assumed that Σ contains the following function symbols, known as *basic combinators*:

$$\mathsf{S} : \Pi\alpha\tau\gamma.\,(\alpha \to \tau \to \gamma) \to (\alpha \to \tau) \to \alpha \to \gamma$$
$$\mathsf{C} : \Pi\alpha\tau\gamma.\,(\alpha \to \tau \to \gamma) \to \tau \to \alpha \to \gamma$$
$$\mathsf{B} : \Pi\alpha\tau\gamma.\,(\alpha \to \gamma) \to (\tau \to \alpha) \to \tau \to \gamma$$
$$\mathsf{K} : \Pi\alpha\gamma.\,\alpha \to \gamma \to \alpha$$
$$\mathsf{I} : \Pi\alpha.\,\alpha \to \alpha$$

The set of terms over Σ and $\mathcal{V}$ is defined below. In what follows, type subscripts are generally omitted.

Terms $\mathcal{T} ::= x \mid \mathsf{f}\langle\,\overline{\tau_n}\,\rangle \mid t_{\tau'\to\tau}t'_{\tau'}$ where $\mathsf{f} : \Pi\,\overline{\alpha_n}\,.\,\sigma \in \Sigma, x \in \mathcal{V}$ and $t, t' \in \mathcal{T}$

The type of the term $\mathsf{f}\langle\,\overline{\tau_n}\,\rangle$ is $\sigma\{\,\overline{\alpha_n} \;\to\; \overline{\tau_n}\,\}$. Terms of the form $t_1\,t_2$ are called applications. Non-application terms are called heads. A term can uniquely be decomposed into a head and n arguments. Let $t = \zeta\;\overline{t'_n}$. Then $head(t) = \zeta$ where ζ could be a variable or constant applied to possibly zero type arguments. The symbol $\boldsymbol{\mathcal{C}_{\mathsf{any}}}$ denotes an arbitrary combinator, whilst $\boldsymbol{\mathcal{C}_3}$ denotes a member of $\{\mathsf{S}, \mathsf{C}, \mathsf{B}\}$. The S-, C- or B-combinators are *fully applied* if they have 3 or more arguments. The K-combinator is fully applied if it has 2 or more arguments and the I is fully applied if it has any arguments. Thes symbols $\boldsymbol{\mathcal{C}_{\mathsf{any}}}$ and $\boldsymbol{\mathcal{C}_3}$ are only used if the symbols they represent are fully applied. Thus, in $\boldsymbol{\mathcal{C}_3}\;\overline{t_n}$, $n \geq 3$ is assumed. The symbols $x, y, z \ldots$ are reserved for variables, $\mathsf{c}, \mathsf{d}, \mathsf{f} \ldots$ for non-combinator constants and ζ, ξ range over arbitrary function symbols and variables and, by an abuse of notation, at times even terms. A head symbol that is not a combinator applied to type arguments or a variable is called *first-order*.

Positions over terms: for a term t, if $t \in \mathcal{V}$ or $t = \mathsf{f}\langle\,\overline{\tau}\,\rangle$, then $pos(t) = \{\epsilon\}$ (type arguments have no position). If $t = t_1\,t_2$ then $pos(t) = \{\epsilon\} \cup \{i.p \mid 1 \leq i \leq 2,\ p \in pos(t_i)\}$. Subterms at positions of the form $p.1$ are called *prefix* subterms. We define *first-order* subterms inductively as follows. For any term t, t is a first-order subterm of itself. If $t = \zeta\,\overline{t_n}$, where ζ is not a fully applied combinator, then the first-order subterms of each t_i are also first-order subterms of t. The notation $s\langle u\rangle$ is to be read as u is a first-order subterm of s. Note that this defintion is subtly different to that in [9] since subterms underneath a fully applied combinator are not considered to be first-order.

Stable subterms: let $\mathsf{FNP}(t, p)$ be a partial function that takes a term t, a position p and returns the longest proper prefix p' of p such that $head(t|_{p'})$ is not a partially applied combinator if such a position exists. For a position $p \in pos(t)$, p is a stable position in t if p is not a prefix position and either $\mathsf{FNP}(t, p)$ is not defined or $head(t|_{\mathsf{FNP}(t,p)})$ is not a variable or combinator. A *stable subterm* is a subterm occurring at a stable position. For example, the subterm a is not stable in $\mathsf{f}\;(\mathsf{S}\;\mathsf{a}\;\mathsf{b}\;\mathsf{c})$, $\mathsf{S}\;(\mathsf{S}\;\mathsf{a})\;\mathsf{b}\;\mathsf{c}$ (in both cases, $head(t|_{\mathsf{FNP}(t,p)}) = \mathsf{S}$) and $\mathsf{a}\;\mathsf{c}$ (a is in a prefix position), but is in $\mathsf{g}\;\mathsf{a}\;\mathsf{b}$ and $\mathsf{f}\;(\mathsf{S}\;\mathsf{a})\;\mathsf{b}$. A subterm that is not stable is known as an *unstable* subterm.

The notation $t[u]$ denotes an arbitrary subterm u of t that may be unstable. The notation $t[u_1, \ldots, u_n]_n$, at times given as $t[\,\overline{u}\,]_n$ denotes that the the term t contains n *non-overlapping* subterms u_1 to u_n. By $u[]_n$, we refer to a context with n non-overlapping holes.

Weak reduction: each combinator is defined by its characteristic equation; $\mathsf{S}\,x\,y\,z = x\,z\,(y\,z)$, $\mathsf{C}\,x\,y\,z = x\,z\,y$, $\mathsf{B}\,x\,y\,z = x\,(y\,z)$, $\mathsf{K}\,x\,y = x$ and $\mathsf{I}\,x = x$. A term t *weak-reduces* to a term t' in one step (denoted $t \longrightarrow_w t'$) if $t = u[s]_p$ and there exists a combinator axiom $l = r$ and substitution σ such that $l\sigma = s$ and $t' = u[r\sigma]_p$. The term $l\sigma$ in t is called a *weak redex* or just redex. By $\longrightarrow_w^*$, the reflexive transitive closure of $\longrightarrow_w$ is denoted. Weak-reduction is terminating and confluent as proved in [13]. By $(t)\!\downarrow^w$, we denote the term formed from t by contracting its leftmost redex.

Literals and clauses: an equation $s \approx t$ is an unordered pair of terms and a literal is an equation or the negation of an equation represented $s \not\approx t$. A clause is a multiset of literals represented $L_1 \vee \cdots \vee L_n$ where each L_i is a literal.

Semantics We follow Bentkamp et al. [6] closely in specifying the semantics. An interpretation is a triple $(\mathcal{U}, \mathcal{E}, \mathcal{J})$ where $\mathcal{U}$ is a ground-type indexed family of non-empty sets called *universes* and $\mathcal{E}$ is a family of functions $\mathcal{E}_{\tau,v} : \mathcal{U}_{\tau\to v} \to (\mathcal{U}_\tau \to \mathcal{U}_v)$. A *type valuation* ξ is a substitution that maps type variables to ground types and whose domain is the set of all type variables. A type valuation ξ is extended to a *valuation* by setting $\xi(x_\tau)$ to be a member of $\mathcal{U}_{(\tau\xi)}$. An interpretation function $\mathcal{J}$ maps a function symbol $\mathsf{f} : \Pi\,\overline{\alpha_n}\,.\,\sigma$ and a tuple of ground types $\overline{\tau_n}$ to a member of $\mathcal{U}_{(\sigma\{\alpha_i\to\tau_i\})}$. An interpretation is *extensional* if $\mathcal{E}_{\tau,v}$ is injective for all τ, v and is *standard* if $\mathcal{E}_{\tau,v}$ is bijective for all τ, v.

For an interpretation $\mathcal{I} = (\mathcal{U}, \mathcal{E}, \mathcal{J})$ and a valuation ξ, a term is denoted as follows: $[\![x]\!]^\xi_\mathcal{I} = \xi(x)$, $[\![\mathsf{f}\langle\,\overline{\tau}\,\rangle]\!]^\xi_\mathcal{I} = \mathcal{J}(\mathsf{f}, [\![\,\overline{\tau}\,]\!]^\xi)$ and $[\![st]\!]^\xi_\mathcal{I} = \mathcal{E}([\![s]\!]^\xi_\mathcal{I})([\![t]\!]^\xi_\mathcal{I})$. An equation $s \approx t$ is true in an interpretation $\mathcal{I}$ with valuation function ξ if $[\![s]\!]^\xi_\mathcal{I}$ and $[\![t]\!]^\xi_\mathcal{I}$ are the same object and is false otherwise. A disequation $s \not\approx t$ is true if $s \approx t$ is false. A clause is true if one of its literals is true and a clause set is true if every clause in the set is true. An interpretation $\mathcal{I}$ models a clause set N, written $\mathcal{I} \models N$, if N is true in $\mathcal{I}$ for all valuation functions ξ.

As Bentkamp et al. point out in [4] there is a subtlety relating to higher-order models and choice. If, as is the case here, attention is not restricted to models that satisfy the axiom of choice, naive skolemisation is unsound. One solution would be to implement skolemisation with mandatory arguments as explained in [20]. However, the introduction of mandatory arguments considerably complicates both the calculus and the implementation. Therefore, we resort to the same 'trick' as Bentkamp et al., namely, claiming completeness for our calculus with respect to models as described above. This holds since we assume problems to be clausified. Soundness is claimed for the implementation with respect to models that satisfy the axiom of choice and completeness can be claimed if the axiom of choice is added to the clause set.

3 The Calculus

The calculus is closely modeled after Bentkamp et al.'s intensional non-purifying calculus [6]. The extensionality axiom can be added if extensionality is required. The main difference between our calculus and that of [6] is that superposition inferences are not allowed beneath fully applied combinators and an extra inference rule is added to deal with superposition beneath variables.

Term Ordering We also demand that the calculus is parameterised by a partial ordering $\succ$ that is well-founded, total on ground terms, stable under substitutions and has the subterm property and which orients all instances of combinator axioms left-to-right. It is an open problem whether a simplification ordering enjoying this last property exists, but it appears unlikely. However, for completeness, compatibility with stable contexts suffices. The $>_{\mathsf{ski}}$ ordering introduced in [9] orients all instances of combinator axioms left-to-right and is compatible with stable contexts. It is *not* compatible with arbitrary contexts. For terms t_1 and t_2 such that $t_1 >_{\mathsf{ski}} t_2$, it is not necessarily the case that $t_1\, u >_{\mathsf{ski}} t_2\, u$ or that $\mathsf{S}\, t_1\, a\, b >_{\mathsf{ski}} \mathsf{S}\, t_2\, a\, b$. We show that by not superposing underneath fully applied combinators and carrying out some restricted superposition beneath variables, this lack of compatibility with arbitrary contexts can be circumvented and does not lead to a loss of completeness. In a number of places in the completeness proof, we assume the following conditions on the ordering (satisfied by the $>_{\mathsf{ski}}$ ordering). It may be possible to relax the conditions at the expense of an increased number of inferences.

P1 For terms t, t' such that $t \longrightarrow_w t'$, then $t \succ t'$
P2 For terms t, t' such that $t \succ t'$ and $head(t')$ is first-order, $u[t] \succ u[t']$

The ordering $\succ$ is extended to literals and clauses using the multiset extension as explained in [21].

Inference Rules The calculus is further parameterised by a selection function that maps a clause to a subset of its negative literals. Due to the requirements of the completeness proof, if a term $t = x\ \overline{s_{n>0}}$ is the maximal term in a clause C, then a literal containing x as a first-order subterm may not be selected. A literal L is called *eligible* if it is selected or there are no selected literals in the clause and it is maximal. In the latter case, it is *strictly eligible* if it is strictly maximal. A variable x has a *bad* occurrence in a clause C if it occurs in C at an unstable position. Occurrences of x in C at stable positions are *good*. A clause C is an *extended combinator axiom* if it is either a combinator axiom or the result of an ARGCONG inference on a combinator axiom.

Conventions: Often a clause is written with a single distinguished literal such as $C' \vee t \approx t'$. In this case:

1. The distinguished literal is always eligible.
2. The name of the clause is assumed to be the name of the remainder without the dash.
3. If the clause is involved in an inference, the distinguished literal is the literal that takes part.

Positive and negative superposition:

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

with the following side conditions:

1. The variable condition (below) holds
2. C is not an extended combinator axiom;
3. $\sigma = mgu(t, u)$;
4. $t\sigma \not\preceq t'\sigma$;
5. $s\langle u\rangle\sigma \not\preceq s'\sigma$;
6. $C\sigma \not\preceq D\sigma$ or D is an extended combinator axiom;
7. $(t \approx t')\sigma$ is strictly eligible in $D\sigma$;
8. $([\neg]\, s\langle u\rangle \approx s')\sigma$ is eligible in $C\sigma$, and strictly eligible if it is positive.

Definition 1. *Let $l = \mathcal{C}_{\mathsf{any}}\, \overline{x_n}$ and $l \approx r$ be an extended combinator axiom. A term $v\ \overline{u_m}$ is* compatible *with $l \approx r$ if $\mathcal{C}_{\mathsf{any}} = \mathsf{I}$ and $m = n$ or if $\mathcal{C}_{\mathsf{any}} = \mathsf{K}$ and $m \geq n - 1$ or if $\mathcal{C}_{\mathsf{any}} \in \{\mathsf{B}, \mathsf{C}, \mathsf{S}\}$ and $m \geq n - 2$.*

Variable condition: $u \notin \mathcal{V}$. If $u = x\ \overline{s_n}$ and D is an extended combinator axiom, then D and u must be compatible.

Because the term ordering $\succ$ is not compatible with unstable contexts, there are instances when superposition beneath variables must be carried out. The SUBVARSUP rule deals with this.

$$\frac{D' \vee t \approx t' \qquad C' \vee [\neg] s\langle y\, \overline{u_n}\,\rangle \approx s'}{(C' \vee D' \vee [\neg] s\langle z t'\, \overline{u_n}\,\rangle \approx s')\sigma}\ \textsc{SubVarSup}$$

with the following side conditions in addition to conditions 3 – 8 of SUP:

1. y has another occurrence bad in C;
2. z is a fresh variable;
3. $\sigma = \{y \to z\, t\}$;
4. t' has a variable or combinator head;
5. $n \leq 1$;
6. D is not an extended combinator axiom.

The EQRES and EQFACT inferences:

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

For both inferences $\sigma = mgu(u, u')$. For EQRES, $(u \not\approx u')\sigma$ is eligible in the premise. For EQFACT, $u'\sigma \not\preceq v'\sigma$, $u\sigma \not\preceq v\sigma$, and $(u \approx v)\sigma$ is eligible in the premise.

In essence, the ARGCONG inference allows superposition to take place at prefix positions by 'growing' equalities to the necessary size.

$$\frac{C' \vee s \approx s'}{C'\sigma \vee (s\sigma)x \approx (s'\sigma)x}\ \textsc{ArgCong}$$
$$C'\sigma \vee (s\sigma)\, \overline{x_2} \approx (s'\sigma)\, \overline{x_2}$$
$$C'\sigma \vee (s\sigma)\, \overline{x_3} \approx (s'\sigma)\, \overline{x_3}$$
$$\vdots$$

The literal $s\sigma \approx s'\sigma$ must be eligible in $C\sigma$. Let s and s' be of type $\alpha_1 \to \cdots \to \alpha_m \to \beta$. If β is not a type variable, then σ is the identity substitution and the inference has m conclusions. Otherwise, if β is a type variable, the inference has an infinite number of conclusions. In conclusions where $n > m$, σ is the substitution that maps β to type $\tau_1 \to \cdots \to \tau_{n-m} \to \beta'$ where β' and each τ_i are fresh type variables. In each conclusion, the x_is are variables fresh for C.

3.1 Extensionality

The calculus can be either intensional or extensional. For the calculus to be extensional, we provide two possibilities. The first is to add a polymorphic extensionality axiom. Let diff be a polymorphic symbol of type $\Pi\tau_1, \tau_2.\, (\tau_1 \to \tau_2) \to (\tau_1 \to \tau_2) \to \tau_1$. The the extensionality axiom can be given as:

$$x\,(\mathsf{diff}\langle\tau_1, \tau_2\rangle\; x\; y) \;\not\approx\; y\,(\mathsf{diff}\langle\tau_1, \tau_2\rangle\; x\; y) \;\vee\; x \approx y$$

However, adding the extensionality axiom to a clause set can be explosive and is not graceful. By any common ordering, the negative literal will be the larger literal and therefore the literal involved in inferences. As it is not of functional type it can unify with terms of atomic type including first-order terms.

In order to circumvent this issue, we developed another method of dealing with extensionality. Unification is replaced by unification with abstraction. During the unification procedure, no attempt is made to unify pairs consisting of terms of functional or variable type. Instead, if the remaining unification pairs can be solved successfully, such pairs are added to the resulting clause as negative constraint literals. This process works in conjunction with the negative extensionality rule presented below.

$$\frac{C' \vee s \not\approx s'}{(C' \vee s\,(\mathsf{sk}\langle\,\overline{\alpha}\,\rangle\;\overline{x}\,) \not\approx s'\,(\mathsf{sk}\langle\,\overline{\alpha}\,\rangle\;\overline{x}\,))\sigma}\;\textsc{NegExt}$$

where $s \not\approx s'$ is eligible in the premise, $\overline{\alpha}$ and $\overline{x}$ are the free type and term variable of the literal $s \not\approx s'$ and σ is the most general type unifier that ensures the well-typedness of the conclusion.

We motivate this second approach to extensionality with an example. Consider the clause set:

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

equality resolution with abstraction on the second clause produces the clause $\mathsf{g} \not\approx \mathsf{f}$. A NEGEXT inference on this clause results in $\mathsf{g}\,\mathsf{sk} \not\approx \mathsf{f}\,\mathsf{sk}$ which can superpose with $\mathsf{g}\,x \approx \mathsf{f}\,x$ to produce $\bot$.

The unification with abstraction procedure used here is very similar to that introduced in [23]. Pseudocode for the algorithm can be found in Algorithm 1. Though the authors strongly suspect this method of dealing with extensionality to be complete, we have been unable to prove it. The inference rules other than ARGCONG and SUBVARSUP must be modified to utilise unification with abstraction rather than standard unification. We show the updated superposition rule. The remaining rules can be modified along similar lines.

Algorithm 1 Unification algorithm with constraints

function $\mathsf{mgu}_{\mathsf{Abs}}(l, r)$
  let $\mathcal{P}$ be a set of unification pairs; $\mathcal{P} := \{\langle l, r\rangle\}$, $\mathcal{D}$ be a set of disequalities; $\mathcal{D} := \emptyset$
  let θ be a substitution; $\theta := \{\}$
  loop
    if $\mathcal{P}$ is empty **then return** (θ, D), where D is the disjunction of literals in $\mathcal{D}$
    Select a pair $\langle s, t\rangle$ in $\mathcal{P}$ and remove it from $\mathcal{P}$
    if s coincides with t **then** do nothing
    else if s is a variable and s does not occur in t **then** $\theta := \theta \circ \{s \mapsto t\}$; $\mathcal{P} := \mathcal{P}\{s \mapsto t\}$
    else if s is a variable and s occurs in t **then** fail
    else if t is a variable **then** $\mathcal{P} := \mathcal{P} \cup \{\langle t, s\rangle\}$
    else if s and t have functional or variable type **then** $\mathcal{D} := \mathcal{D} \cup \{s \not\approx t\}$
    else if s and t have different head symbols **then** fail
    else if $s = \mathsf{f}\, s_1 \ldots s_n$ and $t = \mathsf{f}\, t_1 \ldots t_n$ for some f **then**
      $\mathcal{P} := \mathcal{P} \cup \{\langle s_1, t_1\rangle, \ldots, \langle s_n, t_n\rangle\}$

$$\frac{C_1 \vee t \approx t' \qquad C_2 \vee [\neg] s\langle u\rangle \approx s'}{(C_1 \vee C_2 \vee D \vee [\neg] s\langle t'\rangle \approx s')\sigma}\ \textsc{Sup-wA}$$

where D is the possibly empty set of negative literals returned by unification. SUP-WA shares all the side conditions of SUP given above.

4 Examples

We provide some examples of how the calculus works. Some of the examples utilised come from Bentkamp et al.'s paper [4] in order to allow a comparison of the two methods. In all examples, it is assumed that the clause set has been enriched with the combinator axioms.

Example 1. Consider the unsatisfiable clause:

$$x\,\mathsf{a}\,\mathsf{b} \not\approx x\,\mathsf{b}\,\mathsf{a}$$

Superposing onto the left-hand side with the extended K axiom $\mathsf{K}\,x_1\,x_2\,x_3 \approx x_1\,x_3$ results in the clause $x_1\,\mathsf{b} \not\approx x_1\,\mathsf{a}$. Superposing onto the left-hand side of this clause, this time with the standard K axiom adds the clause $x \not\approx x$ from which $\bot$ is derived by an EQRES inference.

Example 2. Consider the unsatisfiable clause set where $\mathsf{f}\,\mathsf{a} \succ \mathsf{c}$:

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

A SUP inference between the B axiom and the subterm $y\,\mathsf{b}$ of the second clause adds the clause $\mathsf{h}(x_1(x_2\,\mathsf{b}))(x_1(x_2\,\mathsf{a})) \not\approx \mathsf{h}(\mathsf{g}(\mathsf{f}\,\mathsf{b}))(\mathsf{g}\,\mathsf{c})$ to the set. By superposing onto the subterm $x_2\,\mathsf{a}$ of this clause with the equation $\mathsf{f}\,\mathsf{a} \approx \mathsf{c}$, we derive the clause $\mathsf{h}(x_1(\mathsf{f}\,\mathsf{b}))(x_1\,\mathsf{c}) \not\approx \mathsf{h}(\mathsf{g}(\mathsf{f}\,\mathsf{b}))(\mathsf{g}\,\mathsf{c})$ from which $\bot$ can be derived by an EQRES inference.

Example 3. Consider the unsatisfiable clause set where $\mathsf{f}\,\mathsf{a} \succ \mathsf{c}$. This example is the combinatory equivalent of Bentkamp et al.'s Example 6.

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

A SUP inference between the extended $\mathbf{I}$ axiom $\mathbf{I}\,x_1\,x_2 \approx x_1\,x_2$ and the subterm $y\,(\mathbf{B}\,\mathsf{g}\,\mathsf{f})\,\mathsf{a}$ of the second clause adds the clause $\mathsf{h}\,(\mathbf{B}\,\mathsf{g}\,\mathsf{f}\,\mathsf{a})\,\mathbf{I} \not\approx \mathsf{h}\,(\mathsf{g}\,\mathsf{c})\,\mathbf{I}$ to the set. Superposing onto the subterm $\mathbf{B}\,\mathsf{g}\,\mathsf{f}\,\mathsf{a}$ of this clause with the $\mathbf{B}$ axiom results in the clause $\mathsf{h}\,(\mathsf{g}\,(\mathsf{f}\,\mathsf{a}))\,\mathbf{I} \not\approx \mathsf{h}\,(\mathsf{g}\,\mathsf{c})\,\mathbf{I}$. Superposition onto the subterm $\mathsf{f}\,\mathsf{a}$ with the first clause of the original set gives $\mathsf{h}\,(\mathsf{g}\,\mathsf{c}))\,\mathbf{I} \not\approx \mathsf{h}\,(\mathsf{g}\,\mathsf{c})\,\mathbf{I}$ from which $\bot$ can be derived via EQRES.

Note that in Examples 2 and 3, no use is made of SUBVARSUP even though the analogous FLUIDSUP rule in required in Bentkamp et al.'s calculus. We have been unable to develop an example that requires the SUBVARSUP rule even though it is required for the completeness result in Section 6.

5 Redundancy Criterion

In Section 6, we prove that the calculus is refutationally complete. The proof follows that of Bachmair and Ganzinger's original proof of the completeness of superposition [2], but is presented in the style of Bentkamp et al. [6] and Waldmann [31]. As is normal with such proofs, it utilises the concept of *redundancy* to reduce the number of clauses that must be considered in the induction step during the model construction process (view Section 6, Lemma 12, part (ii)).

We define a weaker logic by an encoding $\lfloor\,\rfloor$ of ground terms into non-applicative first-order terms with $\lceil\,\rceil$ as its inverse. The encoding works by indexing each symbol with its type arguments and argument number. For example, $\lfloor\mathsf{f}\rfloor = \mathsf{f}_0$, $\lfloor\mathsf{f}\langle\overline{\tau}\rangle\mathsf{a}\rfloor = \mathsf{f}_1^{\overline{\tau}}\,(\mathsf{a}_0)$. Terms with fully applied combinators as their head symbols are translated to constants such that syntactically identical terms are translated to the same constant. For example, $\lfloor\mathbf{S}\,t_1\,t_2\,t_3\rfloor = \mathsf{s}_0$. The weaker logic is known as the *floor logic* whilst the original logic is called the *ceiling logic*. The encoding can be extended to literals and clauses in the obvious manner as detailed in [5]. The function $\lceil\,\rceil$ is used to compare floor terms. More precisely, for floor logic terms t and t', $t \succ t'$ if $\lceil t\rceil \succ \lceil t'\rceil$. It is straightforward to show that the order $\succ$ on floor terms is compatible with all contexts, well-founded, total on ground terms and has the subterm property.

The encoding serves a dual purpose. Firstly, as redundancy is defined with respect to the floor logic, it prevents the conclusion of all ARGCONG from being redundant. Secondly, subterms in the floor logic correspond to first-order subterms in the ceiling logic. This is of critical importance in the completeness proof.

An inference is the ground instance of an inference I if it is equivalent to I after the application of some grounding substitution θ to the premise(s) and conclusion of I and the result is still an inference.

A ground ceiling clause C is redundant with respect to a set of ground ceiling clauses N if $\lfloor C\rfloor$ is entailed by clauses in $\lfloor N\rfloor$ smaller than itself and the floor of ground instances of extended combinator axioms in $\lfloor N\rfloor$. An arbitrary ceiling clause C is redundant to a set of ceiling clauses N if all its ground instances are redundant with

respect to $\mathcal{G}_\Sigma(N)$, the set of all ground instances of clauses in N. $Red(N)$ is the set of all clauses redundant with respect to N.

For ground inferences other than ARGCONG, an inference with right premise C and conclusion E is redundant with respect to a set of clauses N if $\lfloor E \rfloor$ is entailed by clauses in $\lfloor \mathcal{G}_\Sigma(N) \rfloor$ smaller than $\lfloor C \rfloor$. A non-ground inference is redundant if all its ground instances are redundant.

An ARGCONG inference from a combinator axiom is redundant with respect to a set of clauses N if its conclusion is in N. For any other ARGCONG inference, it is redundant with respect N if its premise is redundant with respect to N, or its conclusion is in N or redundant with respect to N. A set N is saturated up to redundancy if every inference with premises in N is redundant with respect to N.

6 Refutational Completeness

The completeness proof is presented in the style of Bentkamp et al. Let N be a set of clauses containing the five combinator axioms and saturated up to redundancy by the above set of inferences. Let $\mathcal{G}_\Sigma(N)$ be the set of all ground instances of N. By the fact that N is saturated up to redundancy and by the definition of inference redundancy for ARGCONG inferences on combinator axioms, it follows that the following extended combinator axioms are in N for all $n \in \mathbb{N}$.

$$
\begin{aligned}
\mathsf{I}\, x\, \overline{x'_n} &= x\, \overline{x'_n} \\
\mathsf{K}\, x\, y\, \overline{x'_n} &= x\, \overline{x'_n} \\
\mathsf{B}\, x\, y\, z\, \overline{x'_n} &= x\, (y\, z)\, \overline{x'_n} \\
\mathsf{C}\, x\, y\, z\, \overline{x'_n} &= x\, z\, y\, \overline{x'_n} \\
\mathsf{S}\, x\, y\, z\, \overline{x'_n} &= x\, z\, (y\, z)\, \overline{x'_n}
\end{aligned}
$$

Thus, there exists an infinite set $ECA \subseteq N$ consisting of all extended combinator axioms. The set of all ground instances of clauses in ECA is denoted by $\mathcal{G}_\Sigma(ECA)$ and $\mathcal{G}_\Sigma(ECA) \subseteq \mathcal{G}_\Sigma(N)$. As per [6] we build a model for $\lfloor \mathcal{G}_\Sigma(N) \rfloor$ which can then be lifted to a model of $\mathcal{G}_\Sigma(N)$ and N. We define a term-rewriting system R_∞ and use it to construct an interpretation of $\lfloor \mathcal{G}_\Sigma(N) \rfloor$. We then use induction to prove that this interpretation is a model of $\lfloor \mathcal{G}_\Sigma(N) \rfloor$.

Candidate Interpretation We define the set of rewrite rules R_{ECA} as $\{l \to r \mid l \approx r \in \lfloor \mathcal{G}_\Sigma(ECA) \rfloor \text{ and } l \succ r\}$. By the condition that the term order orient all instances of combinator axioms left-to-right, we have that R_{ECA} is the set of all combinator axioms turned into left-to-right rewrite rules.

Lemma 1. *Let $R_{ECA} = R'_{ECA} \cup \{l \to r\}$. Then, l is not reducible by any rule in R'_{ECA}.*

Proof. Let $l = \lfloor \mathcal{C}_{\mathsf{any}}\ \overline{t_n} \rfloor$ and $l' = \lfloor \mathcal{C}_{\mathsf{any}}\ \overline{t'_n} \rfloor$ be the left hand side of two difference members of $\lfloor \mathcal{G}_\Sigma(ECA) \rfloor$. By the definition of the floor translation $\lfloor\rfloor$ we have that $\lfloor \mathcal{C}_{\mathsf{any}}\ \overline{t_n} \rfloor$ and $\lfloor \mathcal{C}_{\mathsf{any}}\ \overline{t'_n} \rfloor$ are both constants and $\lfloor \mathcal{C}_{\mathsf{any}}\ \overline{t_n} \rfloor \neq \lfloor \mathcal{C}_{\mathsf{any}}\ \overline{t'_n} \rfloor$ proving the lemma.

For every clause $C \in \lfloor \mathcal{G}_\Sigma(N) \rfloor$, induction on $\succ$ is used to define sets of rewrite rules E_C and R_C. Assume that E_D has been defined for all clauses $D \in \lfloor \mathcal{G}_\Sigma(N) \rfloor$ such that $D \prec C$. Then R_C is defined as $R_{ECA} \cup (\bigcup_{D \prec C} E_D)$. The set E_C contains the rewrite rule $s \to t$ if the following conditions are met. Otherwise $E_C = \emptyset$.

(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 $R_C \cup \{s \to t\}$
(f) s is not reducible by any rule in R_C

In this case C is called *productive*. R_∞ is defined as $R_{ECA} \cup (\bigcup_{C \in \lfloor \mathcal{G}_\Sigma(N) \rfloor} E_C)$. Note that due to the definition of R_C and condition (d) an extended combinator axiom is never productive. We define a first-order interpretation $\mathcal{T}_\Sigma(\emptyset)/R$ from a rewrite system R as follows. For an equation $t \approx t'$, $\mathcal{T}_\Sigma(\emptyset)/R \models t \approx t'$ if and only if $t \leftrightarrow_R^* t'$. By an abuse of notation, we use R to refer to both the rewrite system and the interpretation that it induces.

Lemma 2. *The rewrite systems R_C and R_∞ are confluent and terminating.*

Proof. Condition (c) ensures that for every rule $s \to t$ in R_C or R_∞ we have $s \succ t$. By the well-foundedness of $\succ$ we have that R_C and R_∞ must be terminating.

By Lemma 1 there are no critical pairs between rules in R_{ECA}. Using this fact and condition (f), we have that there are no critical pairs between rules in R_C and R_∞. Absence of critical pairs implies local confluence. Local confluence and termination implies confluence.

Lemma 3. *If a clause D is true in R_D then it is true in R_C for all $C \succ D$ and in R_∞.*

Proof. As per Waldmann's proof [31].

Lemma 4. *If a clause $D = D' \vee s \approx t$ is productive then D' is false and $s \approx t$ is true in R_C and R_∞ for all $C \succ D$.*

Proof. As per Waldmann's proof [31].

Lemma 5. *Every member of $\lfloor \mathcal{G}_\Sigma(ECA) \rfloor$ is true in R_C for all $C \in \lfloor \mathcal{G}_\Sigma(N) \rfloor$ and in R_∞.*

Proof. By construction of R_{ECA}, all members of $\lfloor \mathcal{G}_\Sigma(ECA) \rfloor$ are true in R_{ECA}. Thus, by definition of R_C all members of $\lfloor \mathcal{G}_\Sigma(ECA) \rfloor$ are true in R_C.

Lemma 6 (Lifting non-SUP inferences). *Let $C\theta \in \mathcal{G}_\Sigma(N)$, such that the selected literals in $C\theta$ match those in C. Then, every* EQRES, EQFACT *and ground instance of an* ARGCONG *inference from $C\theta$ is the ground instance of a corresponding inference from C.*

Proof. The proof is identical to that in [6] ignoring purification.

Definition 2. *A ground* SUP *inference between clauses $D\theta = D'\theta \vee t\theta \approx t'\theta$ and $C\theta$ is* liftable *in three cases:*

1. *The superposed subterm in $C\theta$ is not at or below a variable in C and the variable conditions holds between C and D*
2. *The superposed subterm in $C\theta$ is at a variable y in C which has another occurrence bad in C, $head(t'\theta)$ is a combinator, and D is not a combinator axiom.*
3. *The superposed subterm in $C\theta$ is below a variable y in C which has another occurrence bad in C, $head(y\theta)$ and $head(t'\theta)$ are combinators and D is not a combinator axiom.*

Next, we link liftable SUP inference to non-ground SUP or SUBVARSUP inferences. One of the tricky aspects of the proof is to show that if the inference occurs beneath a variable, the z variable of the SUBVARSUP inference can be used to create a context that weak reduces to an arbitrary conext surrounding a term t. To this end, the function $(\!|\ |\!)$ from λ-terms to combinatory terms is defined:

$$\begin{array}{lcl}
(\!|\lambda x.x|\!) & = & \mathsf{I} \\
(\!|\lambda x.t|\!) & = & \mathsf{K}(\!|t|\!), \quad x \text{ doesn't occur in } t \\
(\!|\lambda x.tt'|\!) & = & \mathsf{B}(\!|t|\!)(\!|\lambda x.t'|\!), \quad x \text{ only occurs free in } t' \\
(\!|\lambda x.tt'|\!) & = & \mathsf{C}(\!|\lambda x.t|\!)(\!|t'|\!), \quad x \text{ only occurs free in } t \\
(\!|\lambda x.tt'|\!) & = & \mathsf{S}(\!|\lambda x.t|\!)(\!|\lambda x.t'|\!), \quad x \text{ occurs free in } t \text{ and } t'
\end{array}$$

The function enjoys the easy to prove property that for λ-free terms t_1 and t_2, $(\!|\lambda x.t_1|\!)\, t_2 \longrightarrow_w t_1\{x \to t_2\}$.

Lemma 7 (Lifting SUP inferences). *Let $D\theta$ and $C\theta$ be members of $\mathcal{G}_\Sigma(N)$ where θ is a substitution and the selected literals of $D, C \in N$ correspond to those of $D\theta$ and $C\theta$. Then for every liftable* SUP *inference between $C\theta$ and $D\theta$, there exists a ground substitution θ' such that $C\theta = C\theta'$, $D\theta = D\theta'$ and the conclusion of the inference between $C\theta$ and $D\theta$ is combinatory congruent to the conclusion of the θ' instance of a* SUP *or* SUBVARSUP *inference between C and D.*

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

$$\frac{D'\theta \vee t\theta \approx t'\theta \qquad C'\theta \vee [\neg]s\theta\langle t\theta\rangle|_p \approx s'\theta}{C'\theta \vee D'\theta \vee [\neg]s\theta\langle t'\theta\rangle|_p \approx s'\theta}\ \text{SUP}$$

where $t\theta \approx t'\theta$ is strictly eligible, $[\neg]s\theta \approx s'\theta$ is strictly eligible if positive and eligible if negative, $C\theta \not\preceq D\theta$, $t\theta \not\preceq t'\theta$ and $s\theta \not\preceq s'\theta$. The proof is broken into two cases.

Case 1: $t\theta$ is not at or beneath a variable in C. In this case, p must be a position of s. Let $u = s|_p$. We have that θ is a unifier of u and t and therefore, there must exist an idempotent mgu σ of t and u. The inference conditions can be lifted. $C\theta \not\preceq D\theta$ implies $C \not\preceq D$, $t\theta \not\preceq t'\theta$ implies $t \not\preceq t'$ and $s\theta \not\preceq s'\theta$ implies $s \not\preceq s'$. Moreover $t\theta \approx t'\theta$ being strictly eligible in $D\theta$ implies that $t \approx t'$ is strictly eligible in D and $[\neg]s\theta \approx s'\theta$ being (strictly) eligible in $C\theta$ implies that $[\neg]s \approx s'$ is (strictly) eligible in C. By liftability, we have that the variable condition holds between D and C. Therefore there is the following SUP inference between D and C:

$$\frac{D' \vee t \approx t' \qquad C' \vee [\neg]s\theta\langle t\rangle|_p \approx s'}{(C' \vee D' \vee [\neg]s\langle t'\rangle|_p \approx s')\sigma}\ \textsc{Sup}$$

We have $(C' \vee D' \vee [\neg]s\langle t'\rangle|_p \approx s')\sigma\theta = (C' \vee D' \vee [\neg]s\langle t'\rangle|_p \approx s')\theta = C'\theta \vee D'\theta \vee [\neg]s\theta\langle t'\theta\rangle|_p \approx s'\theta$ by the idempotency of σ proving that the conclusion of the ground inference is the θ-ground instance of the conclusion of the non-ground inference and we can take $\theta' = \theta$.

Case 2: $t\theta$ is at or beneath a variable in C which has another instance bad in C. In this case, there must exist positions p' and p'' such that $p = p'.p''$ and $s|_{p'} = y\,\overline{u_n}$. Let $u = s|_{p'}$. As per case 1, (strict) eligibility of the ground literals implies (strict) eligibility of the non-ground literals. Further $t'\theta$ having a combinator head implies that t' has a variable or combinator head.

If the inference is beneath a variable in C, then by liftability, $y\theta = \boldsymbol{\mathcal{C}}_{\mathbf{any}}\ \overline{v_{m>0}}$. If $u = y\,\overline{u_{n>1}}$, then $u\theta = \boldsymbol{\mathcal{C}}_{\mathbf{any}}\ \overline{v_m}\,(\,\overline{u_n}\,)\theta$. Thus, $t\theta$ which is a proper subterm of $u\theta$ would be beneath a fully applied combinator which is impossible. Therefore, we have that $n \leq 1$. If the inference is at a variable in C, then we must have $n = 0$ otherwise $y\theta = t\theta$ would not be a first-order subterm.

Thus in both cases $n \leq 1$ and there is a SUBVARSUP inference between C and D:

$$\frac{D' \vee t \approx t' \qquad C' \vee [\neg]s\theta\langle y\,\overline{u_n}\,\rangle|_{p'} \approx s'}{(C' \vee D' \vee [\neg]s\langle zt'\,\overline{u_n}\,\rangle|_{p'} \approx s')\{y \to zt\}}\ \textsc{SubVarSup}$$

where z is a fresh variable. Define the substitution θ' that maps z to $(\!|\lambda x.(y\theta)\langle x\rangle|_{p''}|\!)$ and all other variables w to $w\theta$. Since z is fresh, $C\theta = C\theta'$ and $D\theta = D\theta'$. Further, we have that $(zt')\theta' = (\!|\lambda x.(y\theta)\langle x\rangle|_{p''}|\!)t'\theta \to_w^* y\theta\langle t'\theta\rangle|_{p''}$ and $(zt)\theta' \to_w^* y\theta\langle t\theta\rangle|_{p''} = y\theta$. Therefore:

$$\begin{aligned} & (C' \vee D' \vee [\neg]s\langle zt'\,\overline{u_n}\,\rangle|_{p'} \approx s')\{y \to zt\}\theta' \\ = \quad & (C' \vee D' \vee [\neg]s\langle (\!|\lambda x.(y\theta)\langle x\rangle|_{p''}|\!)t'\,\overline{u_n}\,\rangle \approx s')\theta[y \to z\theta' t\theta] \\ \to_w^* \quad & C'\theta \vee D'\theta \vee [\neg]s\theta\langle t'\theta\rangle|_p \approx s'\theta \end{aligned}$$

proving that the conclusion of the ground inference is combinatory congruent to the θ'-instance of the conclusion of the non-ground inference.

Model Construction

In this section the candidate interpretation R_∞ developed in the previous section is shown to be a model of $\lfloor \mathcal{G}_\Sigma(N) \rfloor$. As per the standard proof, this is done by induction on the clause order $\succ$. For some fixed clause $\lfloor C\theta \rfloor \in \lfloor \mathcal{G}_\Sigma(N) \rfloor$, by Lemma 5, we have that all members of $\lfloor \mathcal{G}_\Sigma(ECA) \rfloor$ are true in $R_{\lfloor C\theta \rfloor}$. By the induction hypothesis, we have that for all clauses $\lfloor D\theta \rfloor \in \lfloor \mathcal{G}_\Sigma(N) \rfloor$ smaller than $C\theta$ are true in $R_{\lfloor C\theta \rfloor}$. In the induction step, it is shown that this implies that $\lfloor C\theta \rfloor$ is true in $R_{\lfloor C\theta \rfloor}$.

Lemma 8 proves that the conclusion of liftable inferences are entailed by clauses smaller than the larger premise along with clauses in $\lfloor \mathcal{G}_\Sigma(ECA) \rfloor$. Its proof is similar to that of its first-order counterpart with the added complication of having to deal with the fact that the conclusion of the ground inference is not necessarily an instance of the conclusion of the non-ground inference. Lemma 11 is about non-liftable SUP inferences. The purpose of the lemma is to show that if there is a non-liftable SUP inference onto a ground clause C, then C must be true in any interpretation that shares the properties of $R_{\lfloor C \rfloor}$.

The lemma identifies the various ways in which a ground inference may not be liftable and then shows that for each, C is entailed by clauses smaller than itself in $\lfloor \mathcal{G}_\Sigma(N) \rfloor$, clauses in $\lfloor \mathcal{G}_\Sigma(ECA) \rfloor$ and $\neg \lfloor D'\theta \rfloor$ where $D\theta$ is the minor premise. The general proof strategy is to show that there exists another clause C' in $\mathcal{G}_\Sigma(N)$ such that $C \succ C'$. By assumption C' is true. To show that this implies $C\theta$ is true, it needs to be proven that $\lfloor C \rfloor$ can be rewritten to $\lfloor C' \rfloor$ by equations assumed to be true in $R_{\lfloor C \rfloor}$. This is difficult, because in the floor logic, rewriting cannot take place beneath a combinator. This is where the assumption that all equalities in $\lfloor \mathcal{G}_\Sigma(ECA) \rfloor$ are true comes into use. Lemmas 9 and 10 are crucial in this stage of the proof. The conclusion of Lemma 9 is stronger than that of Lemma 10, but it has stronger assumptions as well.

Lemma 8. *Let $C\theta, D\theta \in \mathcal{G}_\Sigma(N)$ where the selected literals of $C, D \in N$ correspond with those of $C\theta$ and $D\theta$. Let E be the conclusion of a liftable* SUP *inference from $C\theta$ and $D\theta$ or an* EQRES *or* EQFACT *inference from $C\theta$. Assume that $C\theta$ and $D\theta$ are not redundant with respect to $\mathcal{G}_\Sigma(N)$. Then, $\lfloor E \rfloor$ is entailed by clauses in $\lfloor \mathcal{G}_\Sigma(N) \rfloor$ smaller than $\lfloor C\theta \rfloor$ along with clauses in $\lfloor \mathcal{G}_\Sigma(ECA) \rfloor$.*

Proof. By Lemmas 6 and 7, we have that the conclusion of a SUP inference form $C\theta$ and $D\theta$ or an EQRES or EQFACT inference from $C\theta$ is combinatory congruent to the conclusion of the θ'-ground instance of an inference from C and D (or just C as the case may be) where θ' is a ground substitution such that $D\theta' = D\theta$ and $C\theta' = C\theta$. Let E' be the conclusion of the inference between C and D, we have that $E'\theta' \rightarrow_w^* E$. Because N is saturated up to redundancy, the inference is redundant with respect to N. Thus, the θ' ground instance of the inference is redundant with respect to $\mathcal{G}_\Sigma(N)$. By the definition of inference redundancy, this means that $\lfloor E'\theta' \rfloor$ is entailed by clauses in $\lfloor \mathcal{G}_\Sigma(N) \rfloor$ smaller than $\lfloor C\theta' \rfloor = \lfloor C\theta \rfloor$. Using this and the fact that $E'\theta' \rightarrow_w^* E$, we have that $\lfloor E \rfloor$ is entailed by clauses in $\lfloor \mathcal{G}_\Sigma(N) \rfloor$ smaller than $\lfloor C\theta \rfloor$ along with clauses in $\lfloor \mathcal{G}_\Sigma(ECA) \rfloor$.

Next, terminology is defined that is used in the following couple of Lemmas and the rest of the paper. Let R be an interpretation and t and t' be ground ceiling terms. Then,

$t \sim_R t'$ stands for $R \models \lfloor t \rfloor \approx \lfloor t' \rfloor$. Where the interpretation is obvious, the subscript is omitted.

Lemma 9. *Let R be an interpretation such that every member of $\lfloor \mathcal{G}_\Sigma(ECA) \rfloor$ holds in R. Let u and u' be ground terms such that $u\,\overline{s} \sim_R u'\,\overline{s}$ for every type correct tuple of ground terms $\overline{s}$. Then for ground terms s and s' such that s and s' only differ at positions where s contains u and s' contains u', we have $s \sim_R s'$.*

Proof. The proof proceeds by showing how equations that are true in R can be used to rewrite s into s'. Since R is a model of $\lfloor \mathcal{G}_\Sigma(ECA) \rfloor$ all ground instances of combinator and extended combinator axioms are true in R.

Let $s_0 = \lfloor s \rfloor$ and $\tilde{s}_0 = \lfloor s' \rfloor$. Terms $s_1, s_2, s_3, \ldots$ are defined inductively as follows: s_{i+1} is formed from s_i by rewriting all subterms of the form $\lfloor u\ \overline{v}\ \rfloor$ in s_i to $\lfloor u'\ \overline{v}\ \rfloor$ and then reducing the outermost leftmost weak-redex in the resulting term. Terms $\tilde{s}_1, \tilde{s}_2, \tilde{s}_3, \ldots$ are also defined inductively: $\tilde{s}_{i+1}$ is formed from $\tilde{s}_i$ by reducing the left-most outermost weak-redex in $\tilde{s}_i$.

The algorithm maintains the invariant that for all i, $\lceil s_i \rceil$ and $\lceil \tilde{s}_i \rceil$ are identical other than at positions where $\lceil s_i \rceil$ contains a u and $\lceil \tilde{s}_i \rceil$ contains a u'. Let $\tilde{s}_*$ be the final term in the $\tilde{s}_n$ series. Such a term must exist as weak reduction is terminating. Let s_* be the equivalent term in s_n series. We show that $s_* = \tilde{s}_*$.

Assume that $s_* \neq \tilde{s}_*$. Consider the outermost position at which $\lceil s_* \rceil$ and $\lceil \tilde{s_*} \rceil$ differ. If this position is not beneath a fully applied combinator in $\lceil s_* \rceil$, then s_* and $\tilde{s}_*$ must contain $\lfloor u\,\overline{v}\,\rfloor$ and $\lfloor u'\,\overline{v'}\,\rfloor$ at the corresponding positions. But this is impossible since $\lfloor u\,\overline{v}\,\rfloor$ would have been rewritten to $\lfloor u'\,\overline{v'}\,\rfloor$ at the previous step of the algorithm. Therefore, assume that the outermost position at which $\lceil s_* \rceil$ and $\lceil \tilde{s_*} \rceil$ differ is beneath a fully applied combinator in $\lceil s_* \rceil$. Since we are considering the outermost position, this combinator cannot be a part of u. Thus, we must have that the same combinator occurs fully applied in $\lceil \tilde{s}_* \rceil$. This contradicts the fact that $\tilde{s}_*$ is in weak normal form. Therefore the original assumption is false and $s_* = \tilde{s}_*$.

Lemma 10. *Let R be an interpretation such that every member of $\lfloor \mathcal{G}_\Sigma(ECA) \rfloor$ holds in R. Let u and u' be ground terms such that $u \sim_R u'$. Let $s[]_n$ be a ground context with n holes at stable positions. Then $s[\,\overline{u}\,]_n \sim_R s[\,\overline{u'}\,]_n$.*

Proof. The proof proceeds by induction on $\|s[]_n\|$. In the base case $\|s[]_n\| = 0$ and no instance of u occurs beneath a fully applied combinator. By the definition of stable positions, no instance of u occurs with arguments. Thus, $\lfloor s[\,\overline{u}\,]_n \rfloor$ can be rewritten to $\lfloor s[\,\overline{u'}\,]_n \rfloor$ directly using the equation $\lfloor u \approx u' \rfloor$.

For the inductive case, we have that $\|s[]_n\| > 0$. This splits into two cases depending on whether any of the holes are underneath the leftmost fully applied combinator or not.

Case 1: There are no holes underneath the leftmost fully applied combinator. Consider the context $s'[]_n$ formed by reducing the leftmost fully applied combinator in s. Because $\lfloor \mathcal{G}_\Sigma(ECA) \rfloor$ holds in R, we have $s[\,\overline{u}\,]_n \sim_R s'[\,\overline{u}\,]_n$. By the induction hypothesis we have $s'[\,\overline{u}\,]_n \sim_R s'[\,\overline{u'}\,]_n$ and then using the relevant member of $\lfloor \mathcal{G}_\Sigma(ECA) \rfloor$, we have $s'[\,\overline{u'}\,]_n \sim_R s[\,\overline{u'}\,]_n$. By the transitivity of $\sim_R$, we have $s[\,\overline{u}\,]_n \sim_R s[\,\overline{u'}\,]_n$.

Case 2: The leftmost fully applied combinator is of the form $\boldsymbol{\mathcal{C}}_{\mathsf{any}}\ \overline{t_k}$ where at least one t_i has a hole as a proper subterm. Note that by the definition of stable subterms, none of the t_is can be a hole. Consider the context $s'[]_m$ formed by replacing $\boldsymbol{\mathcal{C}}_{\mathsf{any}}\ \overline{t_k}$ with $(\boldsymbol{\mathcal{C}}_{\mathsf{any}}\ \overline{t_k})\downarrow^w$ in s. Because any holes occurring as subterms of $\boldsymbol{\mathcal{C}}_{\mathsf{any}}\ \overline{t_k}$ occur at stable positions, any holes occurring in $(\boldsymbol{\mathcal{C}}_{\mathsf{any}}\ \overline{t_k})\downarrow^w$ also occur at stable positions. Thus all holes in $s'[]_m$ occur at stable positions. Because $\lfloor \mathcal{G}_\Sigma(ECA) \rfloor$ holds in R, we have $s[\overline{u}]_n \sim_R s'[\overline{u}]_m$. By the induction hypothesis we have $s'[\overline{u}]_m \sim_R s'[\overline{u'}]_m$ and then using the relevant member of $\lfloor \mathcal{G}_\Sigma(ECA) \rfloor$, we have $s'[\overline{u'}]_m \sim_R s[\overline{u'}]_n$. By the transitivity of $\sim_R$, we have $s[\overline{u}]_n \sim_R s[\overline{u'}]_n$.

Lemma 11. *Let $C\theta, D\theta \in \mathcal{G}_\Sigma(N)$, where the selected literals of $C\theta = C'\theta \vee [\neg]s\theta \approx s'\theta$ and $D\theta = D'\theta \vee t\theta \approx t'\theta$ match those of $C, D \in N$. Consider a non-liftable* SUP *inference between $C\theta$ and $D\theta$. Assume that $C\theta$ and $D\theta$ are not redundant with respect to $\mathcal{G}_\Sigma(N)$. Then $\lfloor C\theta \rfloor$ is entailed by clauses in $\lfloor \mathcal{G}_\Sigma(N) \rfloor$ smaller than it, clauses in $\lfloor \mathcal{G}_\Sigma(ECA) \rfloor$ and $\neg\lfloor D'\theta \rfloor$.*

Proof. Assume that the $s\theta \succ s'\theta$ and $t\theta \succ t'\theta$. Let R be an interpretation such that clauses in $\lfloor \mathcal{G}_\Sigma(N) \rfloor$ smaller than $\lfloor C\theta \rfloor$ and all members of $\lfloor \mathcal{G}_\Sigma(ECA) \rfloor$ are true in R whilst $\lfloor D'\theta \rfloor$ is false. By the SUP conditions, we have that either $C\theta \succ D\theta$ or $D\theta \in \mathcal{G}_\Sigma(ECA)$ and in both cases $R \models \lfloor t\theta \approx t'\theta \rfloor$. We need to show that this implies $R \models \lfloor C\theta \rfloor$.

The inference can be non-liftable for one of the following reasons:

1. The variable condition not holding between C and D;
2. It is at or below a variable in C all of whose other occurrences are good in C;
3. It is below a variable x in C and $x\theta$ has a first-order head;
4. It is at or below a variable in C and $t'\theta$ has a first-order head;
5. It is at or below a variable in C and D is an extended combinator axiom.

We fix some terminology common to cases 2–5. Let x be the variable at or beneath which the inference takes place. Then $t\theta$ is a first-order subterm of $x\theta$. Let v be the result of replacing $t\theta$ by $t'\theta$ in $x\theta$ at the relevant position. Let $C' = C\theta[x \to v]$.

Case 1: The variable condition fails to hold because D is an extended combinator axiom not compatible with u By the definition of compatibility, $u = x\,\overline{s_m}$ and $t = \boldsymbol{\mathcal{C}}_{\mathsf{any}}\ \overline{x_n} \approx t'$ and $\boldsymbol{\mathcal{C}}_{\mathsf{any}}\ \overline{x_{n-m}}$ is a fully applied combinator. Thus, $x\theta = (\boldsymbol{\mathcal{C}}_{\mathsf{any}}\ \overline{x_{n-m}})\theta$ is also a fully applied combinator. Let $C'' = C\{x \mapsto ((\boldsymbol{\mathcal{C}}_{\mathsf{any}}\ \overline{x_{n-m}})\theta)\downarrow^w\}$. Because the maximal weak reduction from the largest term in $C\theta$ is greater than the maximal weak reduction from the largest term in $C''\theta$, we have $C\theta \succ C''\theta$ and thus $R \models \lfloor C''\theta \rfloor$. But then, as $C\theta \longrightarrow_w C''\theta$ and all members of $\lfloor \mathcal{G}_\Sigma(ECA) \rfloor$ are true in R, we have $R \models \lfloor C\theta \rfloor$ by congruence.

Case 2: The superposed subterm $t\theta$ is at or beneath a variable x in C all of whose other occurences are good in C. From the existence of the inference between $C\theta$ and $D\theta$, it follows that $t\theta$ must occur at a first-order and thus stable position in $x\theta$. Likewise, the existence of the inference implies that the $x\theta$ involved in the inference occurs at a stable position within $s\theta$. For all other instances of $x\theta$, the fact that x is good in C implies that $x\theta$ is stable in $C\theta$. Thus, by compatibility with stable contexts and the fact that $t\theta \succ t'\theta$, we have that $C\theta \succ C'$ and thus by assumption on R, $R \models \lfloor C' \rfloor$. Since $C\theta$ and C' only differ at stable positions where one contains $t\theta$ and the other $t'\theta$, Lemma

10 can be used to show that every literal of $C\theta$ is equivalent in R to the corresponding literal of C'. This implies $R \models \lfloor C\theta \rfloor$ by congruence.

Case 3: The superposed subterm $t\theta$ is beneath a variable x in C and $x\theta$ has a first order head. We have that $x\theta = (\mathsf{f}\ \overline{s_n})\langle t\theta \rangle$ and $v = (\mathsf{f}\ \overline{s_n})\langle t'\theta \rangle$. Thus, $x\theta \succ v$ follows from the ordering's compatibility with stable context. As v has a first order head and C' is formed from $C\theta$ by replacing $x\theta$ with v, $C\theta \succ C'$ follows from property P2 of the ordering. Hence, $R \models \lfloor C' \rfloor$ and $R \models \lfloor C\theta \rfloor$ via Lemma 9 and congruence.

Case 4: The superposed subterm $t\theta$ is at or beneath a variable x in C and $t'\theta$ has a first order head. We have $t\theta \succ t'\theta$ and C' can be formed from $C\theta$ by replacing all occurrences of t with t'. Thus, $C\theta \succ C'$ follows from property P2 of the ordering. Hence, $R \models \lfloor C' \rfloor$.

We perform a case analysis depending on whether $[\neg]s\theta \approx s'\theta$ is a maximal or selected literal in $C\theta$. First though, note that for all type correct tuple of ground terms $\overline{u}$, if $t\theta\,\overline{u}$ and $t'\theta\,\overline{u}$ are smaller than the maximal term of $C\theta$ then

$$R \models \lfloor t\theta\,\overline{u} \approx t'\theta\,\overline{u} \rfloor \tag{1}$$

This can be shown exactly as Bentkamp et al. do [5]. If $[\neg]s\theta \approx s'\theta$ is the maximal literal in $C\theta$, consider the clause C'' formed by rewriting $\lfloor t\theta \rfloor$ to $\lfloor t'\theta \rfloor$ wherever possible in $\lfloor C\theta \rfloor$. As $t\theta$ is a first-order subterm of the maximal term of $C\theta$, $s\theta$, we have that every term of C'' is smaller than the maximal term of $\lfloor C\theta \rfloor$. Further, we have that $\lceil C'' \rceil$ and C' differ only at positions where $\lceil C'' \rceil$ contains a $t\theta$ and C' contains a $t'\theta$. We prove that C'' can be rewritten to $\lfloor C' \rfloor$ using equalities true in R. Hence $R \models \lfloor C\theta \rfloor$ via congruence.

Let $l \approx r$ be an arbitrary literal of C'' and $l' \approx r'$ be the corresponding literal in $\lfloor C' \rfloor$. Then, since $\lceil C'' \rceil$ and C' only differ where $\lceil C'' \rceil$ contains a $t\theta$ and C' contains a $t'\theta$, we have that $l = \lfloor k[\overline{t\theta_n}] \rfloor$ and $l' = \lfloor k[\overline{t'\theta_n}] \rfloor$ for some context k. We provide an algorithm for rewriting l into l' using equalities true in R. The same can be done for r and r' proving that the literal $l \approx r$ can be rewritten to the literal $l' \approx r'$. Since the literal was chosen arbitrarily, this proves that every literal of C'' can be rewritten to the corresponding literal of $\lfloor C' \rfloor$ in R.

The algorithm is the same as that in Lemma 9. It is repeated here for clarity. Let $l_0 = \lfloor l \rfloor$ and $\tilde{l}_0 = \lfloor l' \rfloor$. Terms $l_1, l_2, l_3, \ldots$ are defined inductively as follows: l_{i+1} is formed from l_i by rewriting all subterms of the form $\lfloor t\theta\ \overline{u} \rfloor$ in l_i to $\lfloor t'\theta\ \overline{u} \rfloor$ and then reducing the outermost leftmost weak-redex in the resulting term. Terms $\tilde{l}_1, \tilde{l}_2, \tilde{l}_3, \ldots$ are also defined inductively: $\tilde{l}_{i+1}$ is formed from $\tilde{l}_i$ by reducing the left-most outermost weak-redex in $\tilde{l}_i$.

The argument that the algorithm terminates and that l_* and $\tilde{l}_*$ are syntactically identical is the same as for Lemma 9. The reason Lemma 9 cannot be invoked here is that using Lemma 9 would require that $R \models \lfloor t\theta\ \overline{w} \approx t'\theta\ \overline{w} \rfloor$ for every type correct tuple of terms $\overline{w}$. In the current context, this does not hold. It therefore remains to justify the rewrites from each l_i to l_{i+1} in the above algorithm. Both l_0 and $\tilde{l}_0$ are smaller than $\lfloor s\theta \rfloor$. Reducing a leftmost redex of a term results in a smaller term. Likewise, by property P2 of the ordering, rewriting a term of the form $\lfloor t\theta\ \overline{u} \rfloor$ to one of the form $\lfloor t'\theta\ \overline{u} \rfloor$ results in a smaller term. Therefore, for all $i > 0$, $l_i \prec l_{i-1} \prec \ldots \prec l_0 \prec \lfloor s\theta \rfloor$ justifying the use of Equation 1.

If $[\neg]s\theta \approx s'\theta$ is a selected literal, then by the selection criteria x cannot be the head of the maximal term of C. Therefore, the algorithm provided above can be used to rewrite C'' into $\lfloor C' \rfloor$ proving that $R \models C\theta$ by congruence.

Case 5: The superposed subterm $t\theta$ is at or beneath a variable x in C and $t\theta \approx t'\theta$ is a member of $\mathcal{G}_\Sigma(ECA)$. In this case $C\theta \longrightarrow_w C'$ and thus $C\theta \succ C'$ by property P1 of the ordering. Hence, $R \models \lfloor C' \rfloor$. Since all members of $\lfloor \mathcal{G}_\Sigma(ECA) \rfloor$ are true in R, every side of a literal in $C\theta$ is equal in R to the equivalent term in C' and $R \models C\theta$ follows by congruence.

Lemma 12 (R_∞ is a model). *Let $\lfloor C\theta \rfloor \in \lfloor \mathcal{G}_\Sigma(N) \rfloor$, then*

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

Proof. The proof proceeds by induction on ground clauses of the floor logic. We assume that (i) to (v) are satisfied by all clauses $D \in \lfloor \mathcal{G}_\Sigma(N) \rfloor$ such that $\lfloor C\theta \rfloor \succ D$. We prove that (i) to (v) hold for $C\theta$. The $\Rightarrow$ direction of (i) follows from the construction. Part (iii) follows from (i) by Lemmas 3 and 4. Part (v) is a straightforward extension of Lemma 5. Therefore, it remains to prove the $\Leftarrow$ direction of (i) and (ii) and (iv) for the case where $C\theta \notin \mathcal{G}_\Sigma(ECA)$. We assume that the selected literals of $C\theta$ match those of $C \in N$.

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

Case 2: $C\theta = C'\theta \vee s\theta \not\approx s'\theta$ and $C\theta$ is not redundant with respect to $\mathcal{G}_\Sigma(N)$.

Case 2.1: $s\theta = s'\theta$. In this case, there is an EQRES inference from $C\theta$:

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

by Lemma 8, we have that $\lfloor C'\theta \rfloor$ is entailed by clauses in $\lfloor \mathcal{G}_\Sigma(N) \rfloor$ smaller than $\lfloor C\theta \rfloor$. By part (iii) of the induction hypothesis, these clauses are true in $R_{\lfloor C\theta \rfloor}$. Therefore, $\lfloor C'\theta \rfloor$ and thus $\lfloor C\theta \rfloor$ are true in $R_{\lfloor C\theta \rfloor}$.

Case 2.2: $s\theta \succ s'\theta$. If $R_{\lfloor C\theta \rfloor} \models \lfloor s\theta \not\approx s'\theta \rfloor$ then the lemma follows. Therefore, assume that it doesn't hold and $\lfloor s\theta \rfloor \downarrow_{R_{\lfloor C\theta \rfloor}} \lfloor s'\theta \rfloor$. There must exist some rule in $R_{\lfloor C\theta \rfloor}$ which reduces $s\theta$. Such a rule must have been produced by some clause $\lfloor D\theta \rfloor = \lfloor D'\theta \vee t\theta \approx t'\theta \rfloor$. Without loss of generality, it is assumed that the C and D are variable disjoint, so that the same substitution θ can be used. It is also assumed that the selected literals of $D\theta$ match those of D. Then there exists the following SUP inference:

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

by Lemma 11, if the inference is non-liftable then $\lfloor C\theta \rfloor$ is entailed by clauses in $\lfloor \mathcal{G}_\Sigma(N) \rfloor$ that are smaller than it, by $\neg\lfloor D'\theta \rfloor$ and by clauses in $\lfloor \mathcal{G}_\Sigma(ECA) \rfloor$. By part (iii) of the induction hypothesis, clauses in $\lfloor \mathcal{G}_\Sigma(N) \rfloor$ smaller than $C\theta$ are true in $R_{\lfloor C\theta \rfloor}$. By Lemma 5 all members of $\lfloor \mathcal{G}_\Sigma(ECA) \rfloor$ are true in $R_{\lfloor C\theta \rfloor}$. By Lemma 4 $\neg\lfloor D'\theta \rfloor$ is true in $R_{\lfloor C\theta \rfloor}$. Therefore $C\theta$ is true in $R_{\lfloor C\theta \rfloor}$.

If the inference is liftable then by Lemma 8, we have that $\lfloor C'\theta \vee D'\theta \vee s\theta\langle t'\theta \rangle \not\approx s'\theta \rfloor$ is entailed by clauses in $\lfloor \mathcal{G}_\Sigma(N) \rfloor$ smaller than $\lfloor C\theta \rfloor$ along with clauses in $\lfloor \mathcal{G}_\Sigma(ECA) \rfloor$. By part (iii) of the induction hypothesis and Lemma 5, all these clauses are true in $R_{\lfloor C\theta \rfloor}$. Therefore, $\lfloor C'\theta \vee D'\theta \vee s\theta\langle t'\theta \rangle \not\approx s'\theta \rfloor$ holds in $R_{\lfloor C\theta \rfloor}$. Since $\lfloor D'\theta \rfloor$ is false in $R_{\lfloor C\theta \rfloor}$, either $R_{\lfloor C\theta \rfloor} \models \lfloor C'\theta \rfloor$ or $R_{\lfloor C\theta \rfloor} \models \lfloor s\theta\langle t'\theta \rangle \not\approx s'\theta \rfloor$. In the latter case, $R_{\lfloor C\theta \rfloor} \models \lfloor s\theta\langle t\theta \rangle \not\approx s'\theta \rfloor$ because $\lfloor t\theta \rfloor \to \lfloor t'\theta \rfloor \in R_{\lfloor C\theta \rfloor}$. Thus, in both cases $R_{\lfloor C\theta \rfloor} \models \lfloor C\theta \rfloor$.

Case 3: $C\theta$ is not redundant and contains no negative literals. In this case $C\theta = C'\theta \vee s\theta \approx s'\theta$. If $E_{\lfloor C\theta \rfloor} = \{\lfloor s\theta \rfloor \to \lfloor s'\theta \rfloor\}$ or $R_{\lfloor C\theta \rfloor} \models C'\theta$ or $s\theta = s'\theta$ then $R_{\lfloor C\theta \rfloor} \models C\theta$. Therefore, assume $E_{\lfloor C\theta \rfloor} = \emptyset$, $s\theta \neq s'\theta$ and $C'\theta$ is false in $R_{\lfloor C\theta \rfloor}$.

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

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

by Lemma 8 the conclusion of the inference is entailed by clauses in $\lfloor \mathcal{G}_\Sigma(N) \rfloor$ smaller than $\lfloor C\theta \rfloor$ and therefore by part (iii) of the induction hypothesis true in $R_{\lfloor C\theta \rfloor}$. Since $t'\theta = s'\theta$, $\lfloor t'\theta \not\approx s'\theta \rfloor$ is false in $R_{\lfloor C\theta \rfloor}$. Therefore $\lfloor t\theta \approx t'\theta \rfloor$ is true in $R_{\lfloor C\theta \rfloor}$ and hence $\lfloor C\theta \rfloor$ is true in $R_{\lfloor C\theta \rfloor}$.

Case 3.2: The literal $s\theta \approx s'\theta$ is strictly maximal and $\lfloor s\theta \rfloor$ is reducible by some rule in $R_{\lfloor C\theta \rfloor}$. This rule must be produced by a clause $\lfloor D\theta \rfloor = \lfloor D'\theta \vee t\theta \approx t'\theta \rfloor$. Without loss of generality, we assume that D and C are variable disjoint so that the same substitution θ can be used. We also assume that the selected literals of D match those of $D\theta$. then there is an SUP inference:

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

by Lemma 11, if the inference is non-liftable then $\lfloor C\theta \rfloor$ is entailed by clauses in $\lfloor \mathcal{G}_\Sigma(N) \rfloor$ that are smaller than it, by $\neg\lfloor D'\theta \rfloor$ and by clauses in $\lfloor \mathcal{G}_\Sigma(ECA) \rfloor$. By part (iii) of the induction hypothesis, clauses in $\lfloor \mathcal{G}_\Sigma(N) \rfloor$ smaller than $C\theta$ are true in $R_{\lfloor C\theta \rfloor}$. By Lemma 5 all members of $\lfloor \mathcal{G}_\Sigma(ECA) \rfloor$ are true in $R_{\lfloor C\theta \rfloor}$. By Lemma 4 $\neg\lfloor D'\theta \rfloor$ is true in $R_{\lfloor C\theta \rfloor}$. Therefore $C\theta$ is true in $R_{\lfloor C\theta \rfloor}$.

If the inference is liftable then by Lemma 8, we have that $\lfloor C'\theta \vee D'\theta \vee s\theta\langle t'\theta \rangle \approx s'\theta \rfloor$ is entailed by clauses in $\lfloor \mathcal{G}_\Sigma(N) \rfloor$ smaller than $\lfloor C\theta \rfloor$ along with clauses in $\lfloor \mathcal{G}_\Sigma(ECA) \rfloor$. By part (iii) of the induction hypothesis and Lemma 5, all these clauses are true in $R_{\lfloor C\theta \rfloor}$. Therefore, $\lfloor C'\theta \vee D'\theta \vee s\theta\langle t'\theta \rangle \approx s'\theta \rfloor$ holds in $R_{\lfloor C\theta \rfloor}$. Since $\lfloor D'\theta \rfloor$ is false in $R_{\lfloor C\theta \rfloor}$, either $R_{\lfloor C\theta \rfloor} \models \lfloor C'\theta \rfloor$ or $R_{\lfloor C\theta \rfloor} \models \lfloor s\theta\langle t'\theta \rangle \approx s'\theta \rfloor$. In the latter case, $R_{\lfloor C\theta \rfloor} \models \lfloor s\theta\langle t\theta \rangle \approx s'\theta \rfloor$ because $\lfloor t\theta \rfloor \to \lfloor t'\theta \rfloor \in R_{\lfloor C\theta \rfloor}$. Thus, in both cases $R_{\lfloor C\theta \rfloor} \models \lfloor C\theta \rfloor$.

Case 3.3: $s\theta \approx s'\theta$ is strictly maximal and $\lfloor s\theta \rfloor$ is not reducible by any rule in $R_{\lfloor C\theta \rfloor}$. By assumption we have that $E_{\lfloor C\theta \rfloor} = \emptyset$. Assume that $\lfloor C\theta \rfloor$ is false in $R_{\lfloor C\theta \rfloor}$. By the construction of $E_{\lfloor C\theta \rfloor}$, it must be the case that $C'\theta$ is true in $R_{\lfloor C\theta \rfloor} \cup \{\lfloor s\theta \rfloor \to \lfloor s'\theta \rfloor\}$. Thus $C'\theta$ must have the form $C''\theta \vee t\theta \approx t'\theta$ where the literal $\lfloor t\theta \approx t'\theta \rfloor$ is true in $R_{\lfloor C\theta \rfloor} \cup \{\lfloor s\theta \rfloor \to \lfloor s'\theta \rfloor\}$, but not in $R_{\lfloor C\theta \rfloor}$. By the confluence of $R_{\lfloor C\theta \rfloor}$, this is equivalent to saying $\lfloor t\theta \rfloor \downarrow_{R_{\lfloor C\theta \rfloor} \cup \{\lfloor s\theta \rfloor \to \lfloor s'\theta \rfloor\}} \lfloor t'\theta \rfloor$, but not $\lfloor t\theta \rfloor \downarrow_{R_{\lfloor C\theta \rfloor}} \lfloor t'\theta \rfloor$. Therefore, the rule $\lfloor s\theta \rfloor \to \lfloor s'\theta \rfloor$ must be used at least once in rewriting either $t\theta$ or $t'\theta$ to a normal form. As $s\theta \succ s'\theta$, $t\theta \succ t'\theta$ and $s\theta \approx s'\theta \succ t\theta \approx t'\theta$, we have that $s\theta \succ t'\theta$ and $s\theta \succeq t\theta$. But the fact that $\lfloor s\theta \rfloor \to \lfloor s'\theta \rfloor$ is used in the rewrite proof implies that $s\theta = t\theta$ and that the rewrite proof looks like this: $\lfloor t\theta \rfloor \to \lfloor s'\theta \rfloor \to^* u \ {}^*\!\leftarrow \lfloor t'\theta \rfloor$. Hence, we have that $R_{\lfloor C\theta \rfloor} \models \lfloor s'\theta \approx t'\theta \rfloor$. Now consider the following EQFACT inference from $C\theta$:

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

by Lemma 8 the conclusion of the inference is entailed by clauses in $\lfloor \mathcal{G}_\Sigma(N) \rfloor$ smaller than $\lfloor C\theta \rfloor$ and therefore by part (iii) of the induction hypothesis true in $R_{\lfloor C\theta \rfloor}$. Since $\lfloor t'\theta \not\approx s'\theta \rfloor$ is false in $R_{\lfloor C\theta \rfloor}$, $\lfloor t\theta \approx t'\theta \rfloor$ is true in $R_{\lfloor C\theta \rfloor}$ and hence $\lfloor C\theta \rfloor$ is true in $R_{\lfloor C\theta \rfloor}$ contradicting our assumption.

7 Construction of Higher-order Model

In this section we lift the model R_∞ of $\lfloor \mathcal{G}_\Sigma(N) \rfloor$ to an interpretation $R_\infty^\uparrow$. We then show that $R_\infty^\uparrow$ is a model of $\mathcal{G}_\Sigma(N)$. We use the notation $t \sim t'$ as a shorthand for $t \sim_{R_\infty} t'$ which, as explained, is equivalent to $R_\infty \models \lfloor t \approx t' \rfloor$.

Lemma 13. *Let t and t' be ground ceiling terms such that $\lfloor t \rfloor \to \lfloor t' \rfloor$ is a rule in R_∞. Then, for all type correct tuple of terms $\overline{u}$, $t\ \overline{u} \sim t'\ \overline{u}$.*

Proof. The rule $\lfloor t \rfloor \to \lfloor t' \rfloor$ must stem from a productive clause of the form $\lfloor C\theta \rfloor \approx \lfloor C'\theta \vee t_1\theta \approx t_2\theta \rfloor$ where $t_1\theta \approx t$ and $t_2\theta \approx t'$. By the definition of a productive clause and part (iv) of Lemma 12, $t \approx t'$ is strictly eligible in $C\theta$ and therefore $t_1 \approx t_2$ is strictly eligible in C. Further, t and t' are of functional type, so t_1 and t_2 must be of functional or polymorphic type. Thus, there is an ARGCONG inference from C with conclusions $(C' \vee t_1\ \overline{x_n} \approx t_2\ \overline{x_n})\sigma$ for all possible n. Let these conclusions be called $E_1 \dots E_n$.

By part (ii) of Lemma 12, $C\theta$ is not redundant and therefore C is not redundant. Thus E_i is either in N or $Red(N)$ for $i \leq i \leq n$. The ground instance of $\lfloor E_i \rfloor$ for some i, $\lfloor C'\theta \vee t\ \overline{u_i} \approx t'\ \overline{u_i} \rfloor$ is thus either in $\lfloor \mathcal{G}_\Sigma(N) \rfloor$ or entailed by clauses in $\lfloor \mathcal{G}_\Sigma(N) \rfloor$. Therefore it is true in R_∞. By Lemma 4, we have that $\lfloor C'\theta \rfloor$ is false in R_∞ which implies that $\lfloor t\ \overline{u_i} \approx t'\ \overline{u_i} \rfloor$ must be true.

Lemma 14. *Let t and t' be ground ceiling terms such that $t \sim t'$ in a single step, but neither $\lfloor t \rfloor \to \lfloor t' \rfloor$ nor $\lfloor t' \rfloor \to \lfloor t \rfloor$ is a rule of R_∞. Let u be a ceiling ground term of the relevant type. Then, $t\,u \sim t'\,u$.*

Proof. It must be the case that there is a single position at which $\lfloor t \rfloor$ and $\lfloor t' \rfloor$ differ. Let v be the subterm in t at this position and and v' be the subterm in t' at the same position. It must also be the case that $\lfloor v \rfloor \to \lfloor v' \rfloor$ or $\lfloor v' \rfloor \to \lfloor v \rfloor$ is a rule in R_∞. Without loss of generality, assume that it is the first. By Lemma 13, we have that $v\ \overline{u} \sim v'\ \overline{u}$ for every type-correct tuple of terms u. Now, Lemma 9 can be invoked to show that $t\,u \sim t'\,u$ since $t\,u$ and $t'\,u$ only differ at a position where one contains v and the other v'.

Lemma 15. *For ground ceiling terms, t, t' and u, if $t \sim t'$ then $t\,u \sim t'\,u$.*

Proof. The proof proceeds by induction on the number of rewrite steps between $\lfloor t \rfloor$ and $\lfloor t' \rfloor$. If $t = t'$ then the Lemma follows trivially. Let the number of rewrite steps between $\lfloor t \rfloor$ and $\lfloor t' \rfloor$ be n. Let t'' be the term such that $\lfloor t' \rfloor$ rewrites to $\lfloor t'' \rfloor$ in $n-1$ steps and $\lfloor t'' \rfloor$ rewrites to $\lfloor t \rfloor$ in a single step. By the induction hypothesis we have that $t'\,u \sim t''\,u$. Thus, if it can be shown that $t''\,u \sim t\,u$ the Lemma follows by the transitivity of $\sim$. $t''\,u \sim t\,u$ follows from either Lemma 13 or 14 depending on whether the rewrite between t'' and t takes place at the top level or not completing the proof.

Lemma 16. *For ground ceiling terms t, t', u, u', if $t \sim t'$ and $u \sim u'$, then $t\,u \sim t'\,u'$ if neither t nor t' is of the form $\boldsymbol{\mathcal{C}_3}\,\overline{s_{n>1}}$, $\mathbf{K}\ \overline{s_{n>0}}$ or $\mathbf{I}\ \overline{s_n}$.*

Proof. By Lemma 15, we have that $t'\,u' \sim t\,u'$, so if it can be shown that $t\,u' \sim t\,u$ the Lemma follows immediately by the transitivity of $\sim$. We have that $\lfloor t \rfloor = \zeta_n(\overline{s_n})$. By the condition on the form of t and t', $t\,u'$ cannot have a fully applied combinator as its head symbol. Therefore, $\lfloor t\,u' \rfloor = \zeta_{n+1}(\overline{s_n}, u')$. It is obvious that any rewrite steps from $\lfloor u' \rfloor$ can be carried out from $\lfloor t\,u' \rfloor$ and therefore $t\,u' \sim t\,u$.

Lemma 17. *For ground ceiling terms u and u' such that $u \sim u'$, $\boldsymbol{\mathcal{C}_3}\,t_1\,t_2\,u \sim \boldsymbol{\mathcal{C}_3}\,t_1\,t_2\,u'$ and $\mathbf{K}\,t\,u \sim \mathbf{K}\,t\,u'$ and $\mathbf{I}\,u \sim \mathbf{I}\,u'$.*

Proof. By multiple applications of Lemma 15, we have that $\lfloor u\,\overline{t}\,\rfloor \approx \lfloor u'\,\overline{t}\,\rfloor$ holds in R_∞ for all type correct tuple of terms $\overline{t}$. We also have that every member of $\lfloor \mathcal{G}_\Sigma(ECA) \rfloor$ holds in R_∞. Thus, the lemma follows by an appeal to Lemma 9 with $R = R_\infty$.

Lemma 18. *For all ground ceiling terms t, t', u and u', if $t \sim t'$ and $u \sim u'$, then $t\,u \sim t'\,u'$.*

Proof. Proof is by induction on $\|t\| + \|u\| + \|t'\| + \|u'\|$. the base case splits into two cases.

Case 1: Neither t nor t' is of the form $\boldsymbol{\mathcal{C}_3}\,t_1\,t_2$, $\mathbf{K}\,t$ or $\mathbf{I}$. Then the proof follows by an application of Lemma 16.
Case 2: One or both of t and t' are of the form $\boldsymbol{\mathcal{C}_3}\,t_1\,t_2$, $\mathbf{K}\,t$ or $\mathbf{I}$. Without loss of generality, assume that t is of the form $\boldsymbol{\mathcal{C}_3}\,t_1\,t_2$. By Lemma 16, $t'\,u' \sim t\,u'$. Thus, if it can be proven that $t\,u' = \boldsymbol{\mathcal{C}_3}\,t_1\,t_2\,u' \sim \boldsymbol{\mathcal{C}_3}\,t_1\,t_2\,u = t\,u$, the theorem follows by the transitivity of $\sim$. By Lemma 17, $\boldsymbol{\mathcal{C}_3}\,t_1\,t_2\,u' \sim \boldsymbol{\mathcal{C}_3}\,t_1\,t_2\,u$ completing the base case.

For the inductive case, one or more of $\|t\|, \|u\|, \|t'\|$ or $\|u'\|$ is greater than 0. We show that the theorem holds for the first two cases. The latter two can be proved in a like manner.

Case 1: $\|t\| > 0$. Let $t'' = (t)\downarrow^w$. Since R_∞ is a model of $\lfloor \mathcal{G}_\Sigma(N) \rfloor$, the floors of the ground instances of all combinator and extended combinator axioms must be true in R_∞. Thus, $t \sim t''$ and by Lemma 15, $t\,u \sim t''\,u$. Since $\|t''\| < \|t\|$, the induction hypothesis can be used to conclude that $t''\,u \sim t'\,u'$. By the transitivity of $\sim$, $t\,u \sim t'\,u'$ follows.
Case 2: $\|u\| > 0$ Since R_∞ models all combinator and extended combinator axioms, we have $u \sim u_2$ where $u_2 = (u)\downarrow^w$. By the induction hypothesis, we have $t\,u_2 \sim t'\,u'$. Either Lemma 16 or Lemma 17 is applicable to prove $t\,u \sim t\,u_2$. The theorem follows by the transitivity of $\sim$.

Definition 3. *Define an interpretation $R_\infty^\uparrow = (\mathcal{U}^\uparrow, \mathcal{E}^\uparrow, \mathcal{J}^\uparrow)$ in the ceiling logic as follows. Let $(\mathcal{U}, \mathcal{E}, \mathcal{J}) = R_\infty$. Let $\mathcal{U}_\tau^\uparrow = \mathcal{U}_{\lfloor\tau\rfloor}$ and $\mathcal{J}^\uparrow(\mathsf{f}, \overline{\tau}) = \mathcal{J}(\mathsf{f}_0^{\overline{\tau}}, \lfloor \overline{\tau} \rfloor)$. Since R_∞ is term-generated, for every $a \in \mathcal{U}_{\lfloor\tau\to\upsilon\rfloor}$ and $b \in \mathcal{U}_{\lfloor\tau\rfloor}$, there exists ground ceiling terms $s : \tau \to \upsilon$ and $t : \tau$ such that $[\![\lfloor s \rfloor]\!]_{R_\infty}^\xi = a$ and $[\![\lfloor t \rfloor]\!]_{R_\infty}^\xi = b$. We define $\mathcal{E}^\uparrow$ as*

$$\mathcal{E}^\uparrow_{\tau,\upsilon}(a)(b) = [\![\lfloor s\,t \rfloor]\!]_{R_\infty}^\xi$$

This interpretation is well defined if the definition of $\mathcal{E}^\uparrow$ does not depend on the choice of the ground terms s and t. To show this, we assume that there exists other ground terms s' and t' such that $[\![\lfloor s' \rfloor]\!]_{R_\infty}^\xi = a$ and $[\![\lfloor t' \rfloor]\!]_{R_\infty}^\xi = b$. By Lemma 18, it follows from $[\![\lfloor s \rfloor]\!]_{R_\infty}^\xi = [\![\lfloor s' \rfloor]\!]_{R_\infty}^\xi$ and $[\![\lfloor t \rfloor]\!]_{R_\infty}^\xi = [\![\lfloor t' \rfloor]\!]_{R_\infty}^\xi$ that

$$[\![\lfloor s\,t \rfloor]\!]_{R_\infty}^\xi = [\![\lfloor s'\,t' \rfloor]\!]_{R_\infty}^\xi$$

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

Since R_∞ is a term-generated model of $\lfloor \mathcal{G}_\Sigma(N) \rfloor$, we can show that $R_\infty^\uparrow$ is also term-generated.

Lemma 19 (Substitution lemma). *For all ceiling logic terms t and grounding substitutions ρ, $[\![t\rho]\!]_{R_\infty^\uparrow} = [\![t]\!]_{R_\infty^\uparrow}^\xi$ if $\alpha\xi = \alpha\rho$ for all α and $\xi(x) = [\![x\rho]\!]_{R_\infty^\uparrow}$ for all x.*

Proof. By induction on the structure of t. If t is a variable x, then $[\![x]\!]_{R_\infty^\uparrow}^\xi = \xi(x) = [\![x\rho]\!]_{R_\infty^\uparrow}$.

If t is of the form $\mathsf{f}\langle\,\overline{\tau}\,\rangle$, then $[\![t]\!]_{R_\infty^\uparrow}^\xi = \mathcal{J}^\uparrow(\mathsf{f}, [\![\,\overline{\tau}\,]\!]^\xi) = \mathcal{J}^\uparrow(\mathsf{f}, \overline{\tau}\,\rho) = [\![\mathsf{f}\langle\,\overline{\tau}\,\rangle\rho]\!]_{R_\infty^\uparrow}$.

Finally, if t is an application of the form $t_1\,t_2$, then

$$\begin{aligned}
[\![t_1\,t_2]\!]_{R_\infty^\uparrow}^\xi &= \mathcal{E}^\uparrow([\![t_1]\!]_{R_\infty^\uparrow}^\xi)([\![t_2]\!]_{R_\infty^\uparrow}^\xi) \\
&\overset{\text{IH}}{=} \mathcal{E}^\uparrow([\![t_1\rho]\!]_{R_\infty^\uparrow})([\![t_2\rho]\!]_{R_\infty^\uparrow}) \\
&= [\![t_1\rho\,t_2\rho]\!]_{R_\infty^\uparrow} \\
&= [\![(t_1\,t_2)\rho]\!]_{R_\infty^\uparrow}
\end{aligned}$$

Lemma 20 (Model transfer to ceiling logic). *$R_\infty^\uparrow$ is a term-generated model of $\mathcal{G}_\Sigma(N)$.*

Proof. By induction on ground terms t of the ceiling logic it is shown that $[\![t]\!]_{R_\infty^\uparrow}^\xi = [\![\lfloor t \rfloor]\!]_{R_\infty}^\xi$. Let t be a ground ceiling term. If t is of the form $\mathsf{f}\langle\,\overline{\tau}\,\rangle$, then $[\![t]\!]_{R_\infty^\uparrow}^\xi = \mathcal{J}^\uparrow(\mathsf{f}, [\![\,\overline{\tau}\,]\!]^\xi) = \mathcal{J}^\uparrow(\mathsf{f}, \overline{\tau}\,) = \mathcal{J}(\mathsf{f}_0, \lfloor\,\overline{\tau}\,\rfloor) = \mathcal{J}(\mathsf{f}_0, [\![\lfloor\,\overline{\tau}\,\rfloor]\!]^\xi) = [\![\lfloor t \rfloor]\!]_{R_\infty}^\xi$

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

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

We have shown that $[\![t]\!]^{\xi}_{R^{\uparrow}_{\infty}} = [\![\lfloor t \rfloor]\!]^{\xi}_{R_{\infty}}$ for all ground ceiling logic terms t. It follows that a ground equation $s \approx t$ or inequality $s \not\approx t$ is true in $R^{\uparrow}_{\infty}$ if and only if $\lfloor s \approx t \rfloor$ or $\lfloor s \not\approx t \rfloor$ is true in R_{∞}. Hence, a ground clause C is true in $R^{\uparrow}_{\infty}$ if and only if $\lfloor C \rfloor$ is true in R_{∞}.

By Lemma 12, 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 $R^{\uparrow}_{\infty}$ and therefore $R^{\uparrow}_{\infty}$ is a model of $\mathcal{G}_{\Sigma}(N)$.

To show that $R^{\uparrow}_{\infty}$ is term-generated, let a be an arbitrary member of $\mathcal{U}^{\uparrow}_{\tau}$. Since $\mathcal{U}^{\uparrow}_{\tau} = \mathcal{U}_{\lfloor \tau \rfloor}$, we have that $a \in \mathcal{U}_{\lfloor \tau \rfloor}$. Since R_{∞} is term-generated, there must exist a floor ground term t such that $[\![t]\!]^{\xi}_{R_{\infty}} = a$. We have $[\![\lceil t \rceil]\!]^{\xi}_{R^{\uparrow}_{\infty}} = [\![t]\!]^{\xi}_{R_{\infty}} = a$, since we have just shown $[\![u]\!]^{\xi}_{R^{\uparrow}_{\infty}} = [\![\lfloor u \rfloor]\!]^{\xi}_{R_{\infty}}$ for all ground ceiling terms u. Hence, $R^{\uparrow}_{\infty}$ is term-generated.

Lemma 21 (Model transfer). *$R^{\uparrow}_{\infty}$ is a model of N.*

Proof. We need to show that for all clauses $C \in N$, C holds in $R^{\uparrow}_{\infty}$ for all ξ. Since $R^{\uparrow}_{\infty}$ is term-generated, we have that for all variables x in C, there exists a ground term s_x such that $\xi(x) = [\![s_x]\!]_{R^{\uparrow}_{\infty}}$. Let ρ be a substitution that maps each term variable x in C to s_x and each type variable α to $\alpha\xi$. Then for any term variable x in C, $[\![x\rho]\!]_{R^{\uparrow}_{\infty}} = \xi(x)$ and for any type variable α in C, $\alpha\rho = \alpha\xi$. Then by Lemma 19 $[\![C]\!]^{\xi}_{R^{\uparrow}_{\infty}} = [\![C\rho]\!]_{R^{\uparrow}_{\infty}}$. As $C\rho$ is ground, by Lemma 20 it is true in $R^{\uparrow}_{\infty}$. Thus so is C.

Lemma 22 (Extensional model). *If the extensionality axiom is present in N, then $R^{\uparrow}_{\infty}$ is an extensional model.*

Proof. The proof is the same as that of Bentkamp et al. [6]

8 Removing Combinator Axioms

Next, we show that it is possible to replace the combinator axioms with a dedicated inference rule. We name the inference NARROW. Unlike the other infernce rules, it works at prefix positions. We define *nearly first-order* positions inductively. For any term t, either $t = \zeta\, \overline{t_n}$ where ζ is not a fully applied combinator or $t = \boldsymbol{\mathcal{C}}_{\mathsf{any}}\, \overline{t_n}$. In the first case, the nearly first-order subterms of t are $\zeta\, \overline{t_i}$ for $0 \leq i \leq n$ and all the nearly first-order subterms of the t_i. In the second case, the nearly first-order subterms are $\boldsymbol{\mathcal{C}}_{\mathsf{any}}\, \overline{t_i}$ for $0 \leq i \leq n$. The notation $s\langle\!\langle u \rangle\!\rangle$ is to be read as u is a nearly first-order subterm of s. The NARROW inference:

$$\frac{C' \vee [\neg] s\langle\!\langle u \rangle\!\rangle \approx s'}{(C' \vee [\neg] s\langle\!\langle r \rangle\!\rangle \approx s')\sigma}\ \textsc{Narrow}$$

with the following side conditions:

1. $u \notin \mathcal{V}$
2. Let $l \approx r$ be a combinator axiom. $\sigma = mgu(l, u)$;
3. $s\langle\!\langle u \rangle\!\rangle\sigma \not\preceq s'\sigma$;
4. $([\neg]\, s\langle\!\langle u \rangle\!\rangle \approx s')\sigma$ is eligible in $C\sigma$, and strictly eligible if it is positive.

We show that any inference that can be carried out using an extended combinator axiom can be simulated with NARROW proving completeness. It is obvious that an EQRES or EQFACT inference cannot have an extended combinator axiom as its premise. By the SUBVARSUP side conditions, an extended combinator axiom cannot be either of its premises. Thus we only need to show that SUP inferences with extended combinator axioms can be simulated. Note that an extended axiom can only be the left premise of a SUP inference. Consider the following inference:

$$\frac{l \approx r \qquad C' \vee [\neg]s\langle u \rangle|_p \approx s'}{(C' \vee [\neg]s\langle r \rangle \approx s')\sigma}\ \textsc{Sup}$$

Let $l = \mathsf{S}\ \overline{x_{n>3}}$. By the variable condition, we have that $u = \zeta\ \overline{t_m}$ where $n \geq m \geq n-2$. If $u = y\ \overline{t_{n-2}}$, then $\sigma = \{y \to \mathsf{S}\, x_1\, x_2, x_3 \to t_1, \ldots, x_n \to t_{n-2}\}$. In this case $r\sigma = (x_1\, x_3\, (x_2\, x_3)\, x_4 \ldots x_n)\sigma = x_1\, t_1\, (x_2\, t_1)\, t_2 \ldots t_{n-2}$ and the conclusion of the inference is $(C' \vee [\neg]s\langle x_1\, t_1\, (x_2\, t_1)\, t_2 \ldots t_{n-2} \rangle \approx s')\{y \to \mathsf{S}\, x_1\, x_2\}$. Now consider the following NARROW inference from C at the nearly first-order subterm $y\, t_1$:

$$\frac{C' \vee [\neg]s\langle \langle\!\langle y\, t_1 \rangle\!\rangle t_2 \ldots t_n \rangle|_p \approx s'}{(C' \vee [\neg]s\langle x_1\, t_1\, (x_2\, t_1)\, t_2 \ldots t_{n-2} \rangle \approx s')\{y \to \mathsf{S}\, x_1\, x_2\}}\ \textsc{Narrow}$$

As can be seen, the conclusion of the SUP inference is equivalent to that of the NARROW inference up to variable naming. The same can be shown to be the case where $u = y\ \overline{t_{n-1}}$ or $u = y\ \overline{t_n}$ or $u = \mathsf{S}\ \overline{t_n}$. Likewise, the same can be shown to hold when the $l \approx r$ is an extended B, C, K or I axiom.

9 Implementation and Evaluation

The calculus described above has been implemented in the Vampire theorem prover [16]. The prover was first extended to support polymorphism. This turned out to be simpler than expected with types being turned into terms and type equality checking changing to a unifiability (or matching) check. Applicative terms are supported by the use of a polymorphic function app of type $\Pi\alpha, \beta.\, (\alpha \to \beta) \to \alpha \to \beta$.

As the SUP, EQRES and EQFACT inferences are identical to their first-order counterparts, these required no updating. The NARROW, SUBVARSUP and ARGCONG inferences had to be added to the implementation. Further, though the NEGEXT inference is not required for completeness, empirical results suggest that it is so useful, that it is permanently on in the implementation.

The ARGCONG inference implemented in Vampire does not match the rule given in the calculus. The rule provided can have an infinite number of conclusions. In Vampire, we have implemented a version of ARGCONG that appends a single fresh variable to each side of the selected literal rather than a tuple and therefore only has a single

conclusion. This version matches what was originally in the calculus. Shortly before the submission of this paper, it was discovered that this lead to a subtle issue in the completeness proof and the inference was changed to its current version. We expect to be able to revert to the previous version and fix the proof. As matters stand, Vampire contains a potential source of incompleteness.

A greater challenge was posed by the implementation of the $>_{\mathsf{ski}}$ ordering in the prover. The ordering is based on the length of the longest weak-reduction from a term. In order to increase the efficiency of calculating this quantity, we implemented caching and lazy evaluation. For example, when inserting a term of the form $\mathsf{f}\, t_1\, t_2$ into the term-sharing data structure, a check is made to see if the maximum reduction lengths of t_1 and t_2 have already been calculated. If they have, then the maximum reduction length of the term being inserted is set to the sum of the maximum reduction lengths of t_1 and t_2. If not, it is left unassigned and only calculated at the time it is required.

During the experimentation phase, it was realised that many redundant clauses were being produced due to narrowing. For example, consider the single literal clause $x\,\mathsf{a}\,\mathsf{b} \approx \mathsf{d} \vee \mathsf{f}\, x \approx \mathsf{a}$. Narrowing the first literal with $\mathbf{C}$-axiom results in $x'\,\mathsf{b}\,\mathsf{a} \approx \mathsf{d} \vee \mathsf{f}\,(\mathbf{C}\, x') \approx \mathsf{a}$. A second narrow with the same axiom results in $x''\,\mathsf{a}\,\mathsf{b} \approx \mathsf{d} \vee \mathsf{f}\,(\mathbf{C}\,(\mathbf{C}\, x'')) \approx \mathsf{a}$ which is extensionally equivalent to first clause and therefore redundant. However, it will not be removed by subsumption since it is only equivalent extensionally. To deal with this problem, we implemented some rewrite rules that replace combinator terms with smaller extensionally equivalent terms. [2] For example, any term of the form $\mathbf{C}\,(\mathbf{C}\, t)$ is rewritten to t. There is no guarantee that these rewrite remove all such redundant clauses, but in practice, they appear to help.

To implement unification with abstraction, we reused the method introduced in our previous work relating to the use of substitution trees as filters [8]. In our current context, this involves replacing all subterms of functional or variable sort with special symbols that unify with any term prior to inserting a term into the substitution tree index.

To evaluate our implementation, we ran a number of versions of our prover across two problem sets and compared their performance against that of some of the leading higher-order provers. The first problem set we tested on was the set of all 592 monomorphic, higher-order problems from the TPTP problem library [29] that do not contain first-class boolean subterms. We restricted our attention to monomorphic problems since some of the provers we used in our evaluation do not support polymorphism. The second benchmark set was produced by the Isabelle theorem prover's Sledgehammer system. It contains 1253 benchmarks kindly made available to us by the Matryoshka team and is called SH-λ following their naming convention. All tests were run with a CPU time limit of 300. Experiments were performed on StarExec [28] nodes equipped with four 2.40 GHz Intel Xeon CPUs. Our experimental results are publicly available[3].

To compare out current implementation against, we chose the Leo-III, 1.4, Satallax 3.4, Zipperposition 1.5 and Vampire-THF 4.4 provers. These provers achieved the top four spots in the 2019 CASC system competition. Vampire THF 4.4 was developed by the authors, but uses different principles being based on combinatory unifica-

[2] Thanks to Petar Vukmirović for suggesting and discussing this idea.

[3] `https://github.com/vprover/vampire_publications/tree/master/experimental_data/IJCAR-2020-COMB-SUP`

Table 1: Problems proved theorem or unsat

	TPTP TH0 Problems		Sh-λ Problems	
	Solved	Uniques	Solved	Uniques
Satallax 3.4	473	0	628	5
Leo-III 1.4	**482**	**6**	661	13
Vampire-THF 4.4	472	1	**717**	**14**
Vampire-csup-ninj	470	0 (1)	687	1 (2)
Vampire-csup-ax	469	0 (0)	680	0 (3)
Vampire-csup-abs	472	0 (0)	685	0 (0)
Vampire-csup-prag	475	1 (3)	628	0 (1)
Zipperposition 1.5	476	0	609	6

tion. We compare the performance of these provers against four variants of our current implementation. First, Vampire-csup-ax which implements the calculus as described above and uses the extensionality axiom abstraction. Second, Vampire-csup-abs which deals with extensionality via unification with. Third, Vampire-csup-ninj which incorporates an inference to synthesise left-inverses for injective functions in a manner similar to Leo-III [26, Section 4.2.5] and finally Vampire-csup-prag which introduces various heuristics to try and control the search space, though at the expense of completeness. For example, it implements a heuristic that restricts the number of narrow steps. It also switches off the SUBVARSUP rule which is never used in a proof produced by the other variants of Vampire-csup. All four versions are run on top of a first-order portfolio of strategies. These strategies control options such as the saturation algorithm used, which simplification inferences are switched on and so forth. The results of the experiments can be found summarised in Table 1. In brackets, the number of uniques between Vampire-csup versions is provided. The closeness of the results on the TPTP benchmarks is striking. Out of the 592 benchmarks, 95 are known not to be theorems, leaving 497 problems that could possibly be proved. All the provers are remarkably close to this number and each other. Leo-III which is slightly ahead of the other provers, only manages this through function synthesis which is not implemented in any of the other provers.

It is disappointing that Vampire-csup performs worse than its predecessor Vampire-THF 4.4 on Sledgehammer problems. We hypothesise that this is related to the explosion in clauses created as a result of narrowing. Vampire-csup-prag is supposed to control such an explosion, but actually performs worst of all. This is likely due to the fact that it runs a number of lengthy strategies aimed particularly at solving higher-order problems requiring complex unifiers. Interestingly, the pragmatic version solved a difficulty rating 1.00 TPTP problem, namely, NUM829^5.p.

10 Conclusion and Related Work

The combinatory superposition calculus presented here is amongst a small group of complete proof calculi for higher-order logic. This group includes the RUE resolution

calculus of Benzmüller which has been implemented in the LEO-II theorem prover [7]. The Satallax theorem prover implements a complete higher-order tableaux calculus [11]. More recently, Bentkamp et al. have developed a complete superposition calculus for clausal HOL [4]. As superposition is one of the most successful calculi in first-order theorem proving [21], their work answered a significant open question, namely, whether superposition could be extended to higher-order logic.

Our work is closely related to theirs, and in some senses, the SUBVARSUP rule of our calculus mirrors the FLUIDSUP rule of their calculus. However, there are some crucial differences. Arguably, the side conditions on SUBVARSUP are tighter than those on FLUIDSUP and some problems such as the one in Example 3 can be solved by our calculus without the use of SUBVARSUP whilst requiring the use of FLUIDSUP in Bentkamp et al.'s calculus. Lambda superposition is based on higher-order unification and λ-terms. Our calculus is based on (applicative) first-order terms and first-order unification and implementations can therefore reuse the well-studied data structures and algorithms of first-order theorem proving. On the downside, narrowing terms with combinator axioms is still explosive and results in redundant clauses. It is also never likely to be competitive with higher-order unification in finding complex unifiers. This is particularly the case with recent improvements in higher-order unification being reported [30].

Many other calculi for higher-order theorem proving have been developed, most of them incomplete. Amongst the early calculi to be devised are Andrew's mating calculus [1] and Miller's expansion tree method [19] both linked to tableaux proving. More recent additions include an ordered (incomplete) paramodulation calculus as implemented in the Leo-III prover [27] and a higher-order sequent calculus implemented in the AgsyHOL prover [17]. In previous work, the current authors have extended first-order superposition to use a combinatory unification algorithm [8]. Finally there is ongoing work to extend SMT solving to higher-order logic [3].

There have also been many attempts to prove theorems in HOL by translating to FOL. One of the pioneers in suggesting this approach was Kerber [15]. Since his early work, it has become commonplace to combine a dedicated higher-order theorem prover with a first-order prover used to discharge first-order proof obligations. This is the approach taken by many interactive provers and their associated hammers such as Sledgehammer [22] and CoqHammer [12]. It is also the approach adopted by leading automated higher-order provers Leo-III and Satallax.

Our work is also relevant to the much discussed philosophical question regarding the status of higher-order logic as an independent logic separate from first-order logic [25] [14]. We make no attempt to provide a definitive answer to this question, but note that our work would suggest that, at least when discussing higher-order logic with Henkin semantics, the gap between the two is not as large as it may appear.

In this paper we have presented a complete calculus for a polymorphic, boolean-free, intensional, combinatory formulation of higher-order logic. For the calculus to be extensional, an extensionality axiom can be added maintaining completeness, but losing gracefulness. Alternatively, unification can be turned into unification with abstraction maintaining gracefulness, but losing a completeness guarantee. Experimental results show the calculus to be competitive with the leading higher-order provers.

It remains to tune the implementation and calculus. We plan to further investigate the use of heuristics in taming the explosion of clauses that result from narrowing. the heuristics may lead to incompleteness. It would also be of interest to investigate the use of heuristics or even machine learning to guide the prover in selecting specific combinator axioms to narrow a particular clause with. One of the advantages of our calculus is that it does not consider terms modulo β- or weak-reduction. Therefore, theoretically, a larger class of terms should be comparable by the non-ground order than is possible with a calculus that deals with β- or weak-equivalence classes. It remains to implement a stricter version of the $>_{\mathsf{ski}}$ ordering and evaluate its usefulness.

As a next step, we plan to add support for booleans and choice to the calculus. An appealing option for booleans is to extend the unification with abstraction approach currently used for functional extensionality. No attempt would be made to solve unification pairs consisting of boolean terms. Rather, these would be added as negated bi-implications to the result which would then be re-clausified.

Finally, we feel that our calculus calculus complements existing higher-order calculi and presents a particularly attractive option for extending existing first-order superposition provers to dealing with HOL.

Acknowledgements Thanks to Jasmin Blanchette, Alexander Bentkamp and Petar Vukmirović for many discussions on aspects of this research. We would also like to thank Andrei Voronkov, Martin Riener and Michael Rawson. Thanks is also due to the maintainers of StarExec and the TPTP problem library both of which were invaluable to this research. The first author thanks the family of James Elson for funding his research.

References

1. Andrews, P.B.: On connections and higher-order logic. Journal of Automated Reasoning **5**(3), 257–291 (1989)
2. Bachmair, L., Ganzinger, H.: Rewrite-based equational theorem proving with selection and simplification. J. Log. Comput. **4**(3), 217–247 (1994)
3. Barbosa, H., Reynolds, A., El Ouraoui, D., Tinelli, C., Barrett, C.: Extending smt solvers to higher-order logic. In: Fontaine, P. (ed.) CADE-27. LNCS, vol. 11716, pp. 35–54. Springer (2019)
4. Bentkamp, A., Blanchette, J., Tourret, S., Vukmirović, P., Waldmann, U.: Superposition with lambdas. In: Fontaine, P. (ed.) CADE-27. LNCS, vol. 11716, pp. 55–73. Springer (2019)
5. Bentkamp, A., Blanchette, J., Tourret, S., Vukmirović, P., Waldmann, U.: Superposition with lambdas (technical report). Technical report (2019), `http://matryoshka.gforge.inria.fr/pubs/lamsup_report.pdf`
6. Bentkamp, A., Blanchette, J.C., Cruanes, S., Waldmann, U.: Superposition for lambda-free higher-order logic. In: Galmiche, D., Schulz, S., Sebastiani, R. (eds.) International Joint Conference on Automated Reasoning (IJCAR 2018). LNCS, vol. 10900, pp. 28–46. Springer (2018)
7. Benzmüller, C., Sultana, N., Paulson, L.C., Theib, F.: The higher-order prover Leo-II. Journal of Automated Reasoning **55**(4), 389–404 (2015). https://doi.org/10.1007/s10817-015-9348-y
8. Bhayat, A., Reger, G.: Restricted combinatory unification. In: Fontaine, P. (ed.) CADE-27. LNCS, vol. 11716, pp. 74–93. Springer (2019)

9. Bhayat, A., Reger, G.: A knuth-bendix-like ordering for orienting combinator equations (technical report). Technical report, University of Mancester (2020), https://github.com/vprover/vampire_publications/blob/master/paper_drafts/comb_compat_ordering_report.pdf
10. Bobot, F., Paskevich, A.: Expressing polymorphic types in a many-sorted language. In: Tinelli, C., Sofronie-Stokkermans, V. (eds.) FroCoS. LNCS, vol. 6989, pp. 87–102. Springer (2011)
11. Brown, C.E.: Satallax: An automatic higher-order prover. In: Gramlich, B., Miller, D., Sattler, U. (eds.) Automated Reasoning. pp. 111–117. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)
12. Czajka, Ł., Kaliszyk, C.: Hammer for coq: Automation for dependent type theory. Journal of Automated Reasoning **61**(1), 423–453 (Jun 2018)
13. Hindley, J.R., Seldin, J.P.: Lambda-Calculus and Combinators: An Introduction. Cambridge University Press, New York, NY, USA, 2 edn. (2008)
14. Hylton, P., Kemp, G.: Willard van orman quine. In: Zalta, E.N. (ed.) The Stanford Encyclopedia of Philosophy. Metaphysics Research Lab, Stanford University, spring 2019 edn. (2019)
15. Kerber, M.: How to prove higher order theorems in first order logic pp. 137–142 (01 1991)
16. Kovács, L., Voronkov, A.: First-order theorem proving and Vampire. In: International Conference on Computer Aided Verification. pp. 1–35. Springer (2013)
17. 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)
18. Meng, J., Paulson, L.C.: Translating higher-order clauses to first-order clauses. Journal of Automated Reasoning **40**(1), 35–60 (Jan 2008). https://doi.org/10.1007/s10817-007-9085-y, https://doi.org/10.1007/s10817-007-9085-y
19. Miller, D.A.: Proofs in higher-order logic. Ph.D. thesis, University of Pennsylvania (1983)
20. Miller, D.A.: A compact representation of proofs. Studia Logica **46**(4), 347–370 (1987)
21. Nieuwenhuis, R., Rubio, A.: Paramodulation-based theorem proving. In: Robinson, A., Voronkov, A. (eds.) Handbook of Automated Reasoning, vol. I, chap. 7, pp. 371–443. Elsevier Science (2001)
22. Paulson, L.C., Blanchette, J.C.: Three years of experience with sledgehammer, a practical link between automatic and interactive theorem provers. IWIL-2010 **1** (2010)
23. Reger, G., Suda, M., Voronkov, A.: Unification with abstraction and theory instantiation in saturation-based reasoning. In: Beyer, D., Huisman, M. (eds.) TACAS 2018. LNCS, vol. 10805, pp. 3–22. Springer International Publishing (2018)
24. Schulz, S.: E — a brainiac theorem prover. AI Communications **15**(2, 3), 111–126 (2002)
25. Shapiro, S.: Foundations without foundationalism : a case for second-order logic. Oxford logic guides ; 17, Clarendon (1991)
26. Steen, A.: Extensional Paramodulation for Higher-Order Logic and its Effective Implementation Leo-III. Ph.D. thesis, Freie Universität Berlin (2018)
27. Steen, A., Benzmüller, C.: The higher-order prover Leo-III. In: International Joint Conference on Automated Reasoning. LNCS, vol. 10900, pp. 108–116. Springer (2018)
28. Stump, A., Sutcliffe, G., Tinelli, C.: StarExec, a cross community logic solving service. https://www.starexec.org (2012)
29. Sutcliffe, G.: The TPTP problem library and associated infrastructure, from CNF to TH0, TPTP v6.4.0. Journal of Automated Reasoning **59**(4), 483–502 (2017)
30. Vukmirović, P., Bentkamp, A., Nummelin, V.: Efficient full higher-order unification (2019), http://matryoshka.gforge.inria.fr/pubs/hounif_paper.pdf, unpublished, http://matryoshka.gforge.inria.fr/pubs/hounif_paper.pdf

31. 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