





Superposition with Delayed Unification

Ahmed Bhayat¹  , Johannes Schoisswohl² , and Michael Rawson² 

¹ University of Manchester, Manchester, UK ahmed.bhayat@manchester.ac.uk

² TU Wien, Vienna, AT

{michael.rawson,johannes.schoisswohl}@tuwien.ac.at

Abstract. Classically, in saturation-based proof systems, unification has been considered atomic. However, it is also possible to move unification to the calculus level, turning the steps of the unification algorithm into inferences. For calculi that rely on unification procedures returning large or even infinite sets of unifiers, integrating unification into the calculus is an attractive method of dovetailing unification and inference. This applies, for example, to AC-superposition and higher-order superposition. We show that first-order superposition remains complete when moving unification rules to the calculus level. We discuss some of the benefits this has even for standard first-order superposition and provide an experimental evaluation.

1 Introduction

Unification is a key feature in many proof calculi, particularly those based on the saturation framework. It acts as a filter, reducing the number of inferences that need to be carried out by instantiating terms only to the degree necessary. However, many unification algorithms have large time complexities and produce large, or even infinite, sets of unifiers. This is the case, for example, for AC-unification, which can produce a doubly exponential number of unifiers [10], and higher-order unification, which can produce an infinite set of unifiers [20]. This motivates the study of how unification rules can be integrated into proof calculi to allow them to dovetail with standard calculus rules. One way to achieve this is to use the concept of unification with abstraction [17,13]. The general idea is that during the unification process, instead of solving all unification pairs, certain pairs are retained and added to the conclusion of an inference as negative *constraint* literals. Calculus-level unification inferences then work on such literals to solve these constraints and remove the literals in the case they are unifiable. Note how this differs from constrained resolution-style calculi such as [4,15] where the constraints are completely separate from the rest of the clause and are not subject to inferences.

To demonstrate the idea of dedicated unification inferences in combination with unification with abstraction, we provide the following example.

$$C_1 = f(g(a, x)) \not\approx t \quad C_2 = f(g(a, b)) \approx t$$

A standard superposition calculus would proceed by unifying $f(g(a, b))$ and $f(g(a, x))$ with the unifier $\sigma = \{x \rightarrow b\}$ and then rewriting C_1 with C_2 to derive $t\sigma \not\approx t\sigma$. Equality resolution on $t\sigma \not\approx t\sigma$ would then derive \perp . It is also possible to proceed by rewriting C_1 with C_2 *without* computing σ and instead add the constraint literal $g(a, x) \not\approx g(a, b)$ to the conclusion to derive $t \not\approx t \vee g(a, x) \not\approx g(a, b)$. A dedicated unification inference could then decompose the constraint literal resulting in $t \not\approx t \vee a \not\approx a \vee b \not\approx x$. Further unification inferences could bind x to b , and remove the trivial pairs $a \not\approx a$ and $t \not\approx t$ to derive \perp .

In this paper, we investigate moving unification to the calculus level for standard first-order superposition. Whilst this may seem like a regressive step, as we lose much of unification’s power to act as a filter on inferences and hence produce many more clauses, we think the investigation is valuable for two reasons.

Firstly, by showing how syntactic first-order unification can be lifted to the calculus level, we provide a roadmap for how more complex unification problems can be lifted to the calculus level. This may prove particularly useful in the higher-order case, where abstraction may expose terms to standard calculus rules that were unavailable before. Moreover, we note that in our calculus we do not turn the entire unification problem into a constraint, but rather a subproblem. Whilst this may be merely an interesting detail for first-order unification, for more complex unification problems, such a method could be used to eagerly solve simple unification subproblems whilst delaying complex subproblems by adding them as constraints.

Secondly, one of the most expensive operations in first-order theorem provers is the maintenance of indices. Indices are crucial to the performance of modern solvers, as they facilitate the efficient retrieval of terms unifiable or matchable with a query term. However, solvers typically spend a large amount of time inserting and removing terms from indices as well as unifying against terms in the indices. This is particularly the case in the presence of the AVATAR architecture [24] wherein a change in the model can trigger the insertion and removal of thousands of terms from various indices. By moving unification to the calculus level, we can replace complex indices with simple hash maps, since to trigger an inference we merely need to check for top symbol equality and not unifiability. Insertion and deletion become $O(1)$ time operations. However, for first-order logic, we do not expect the time gained to offset the downsides of extra inferences carried out and extra clauses created. Our experimental results back up this hypothesis (see Section 7). Our main contributions are:

- Designing a modified superposition calculus that moves unification to the calculus level (Section 3).
- Proving the calculus to be statically and dynamically refutationally complete (Section 5).
- Providing a thorough empirical evaluation of the calculus (Section 7).

2 Preliminaries

Syntax We consider standard monomorphic first-order logic with equality. We assume a signature consisting of a finite set of (monomorphically) typed function symbols and a single predicate, equality, denoted by \approx . A non-equality atom A can be expressed using equality as $A \approx \top$ where \top is a special function symbol [18]. Terms are formed in the normal way from variables and function symbols. We commonly use s, t or u or their primed variants to refer to terms. We write $s : \tau$ to show that term s has type τ . A term is ground if it contains no variables. We use the notation \bar{s}_n to refer to a tuple or list of terms of length n . More generally, we use the over bar notation to refer to tuples and lists of various objects. Where the length of the tuple or list is not relevant, we drop the subscript. By s_i we denote the i th element of the tuple \bar{s}_n . Literals are positive or negative equalities written as $s \approx t$ and $s \not\approx t$ respectively. We use $s \approx\!\!\approx t$ to refer to either a positive or a negative equality. Clauses are multisets of literals. A clause that contains no literals is known as the empty clause and denoted \perp .

A substitution is a mapping from variables to terms. We assume, w.l.o.g., that all substitutions are idempotent. We commonly denote substitutions using σ and θ and denote the application of a substitution σ to a term s by $s\sigma$. A substitution θ is grounding for a term s , if $s\theta$ is ground. The definition of grounding substitution can be extended to literals and clauses in the obvious manner. A substitution σ is a unifier of terms s and t if $s\sigma = t\sigma$. A unifier σ is more general than a unifier σ' if there exists a substitution ρ such that $\sigma\rho = \sigma'$. With respect to syntactic first-order unification, if two terms are unifiable then they have a single most general unifier up to variable naming [1].

A transitive irreflexive relation over terms is known as an ordering. The superposition calculus we present below is, as usual, parameterised by a simplification ordering on ground terms. An ordering \succ is a simplification ordering, if it possesses the following properties. It is total on ground terms. It is compatible with contexts, meaning that if $s \succ t$, then $u[s] \succ u[t]$. It is well-founded. Note that every simplification ordering has the subterm property. Namely, that if t is a proper subterm of s , then $s \succ t$. For non-ground terms, the only property that is required of the ordering is that it is stable under substitution. That is, if $s \succ t$ then for all substitutions σ , $s\sigma \succ t\sigma$. We extend the ordering \succ to literals in the standard fashion via its multiset extension. A positive literal $s \approx s'$ is treated as the multiset $\{s, s'\}$, whilst a negative literal $s \not\approx s'$ is treated as the multiset $\{s, s, s', s'\}$. The ordering is extended to clauses by its two-fold multiset extension. We use \succ to denote the ordering on terms and its multiset extensions to literals and clauses.

Semantics An interpretation is a pair $(\mathcal{U}, \mathcal{J})$, where \mathcal{U} is a set of typed universes and \mathcal{J} is an interpretation function, such that for each function symbol $f : \tau_1 \times \dots \times \tau_n \rightarrow \tau$ in the signature, $\mathcal{J}(f)$ is a concrete function of type $\mathcal{U}_{\tau_1} \times \dots \times \mathcal{U}_{\tau_n} \rightarrow \mathcal{U}_{\tau}$. A valuation ξ is a function that maps each variable $x : \tau$ to a member of \mathcal{U}_{τ} . For a given interpretation \mathcal{M} and valuation ξ , we use $\llbracket t \rrbracket_{\mathcal{M}}^{\xi}$ to represent the denotation of t in \mathcal{M} given ξ . A positive literal $s \approx t$ is true in an interpretation

\mathcal{M} for valuation ξ if $\llbracket s \rrbracket_{\mathcal{M}}^{\xi} = \llbracket t \rrbracket_{\mathcal{M}}^{\xi}$ and false otherwise. A negative literal $s \not\approx t$ is true in an interpretation \mathcal{M} for valuation ξ if $s \approx t$ is false. A clause C holds in an interpretation \mathcal{M} for valuation ξ if one of its literals is true in \mathcal{M} for ξ . An interpretation \mathcal{M} *models* a clause C if C holds in \mathcal{M} for every valuation. An interpretation models a clause set, if it models every clause in the set. A set of clauses M entails a set of clauses N , denoted $M \models N$, if every model of M is also a model of N .

3 Calculus

Intuitively, what we are aiming for with our calculus, is that whenever standard superposition applies a substitution σ to a conclusion with the side condition “ σ is a unifier of terms t_1 and t_2 ”, our calculus adds a constraint $t_1 \not\approx t_2$ to the conclusion. The calculus then has further inference rules that mimic the steps of a first-order unification algorithm and work on negative literals. Our presentation below does not quite follow this intuition. Instead, if the unification problem is trivial we solve it immediately. If it is non-trivial, we carry out a single step of unification and add the resulting sub-problems as constraints. Our reasons for doing this are two-fold.

1. Adding the entire unification problem $t_1 \not\approx t_2$ as a constraint can lead to a constraint literal that is larger, with respect to \succ , than any literal occurring in the premises. This causes difficulties in the completeness proof.
2. More pertinently, keeping in mind our planned applications to more complex logics, we wish to show that delayed unification remains complete even when only selected sub-problems of the original unification problem are added as constraints. In the context of higher-order logic, for example, this could allow for the eager solving of simple unification sub-problems whilst only the most difficult are added as constraints. See Section 6 for further details.

Wherever we present a clause as a subclause C' and a literal l (e.g. $C' \vee l$), we denote the entire clause by the same name as the subclause without the dash (e.g. we refer to the clause $C' \vee l$ by C). As in the classical superposition calculus, our calculus is parameterised by a *selection function* that is used to restrict the number of applicable inferences in order to avoid the search space growing unnecessarily. A selection function *sel* is a function that maps a clause to a subset of its negative literals. We say that literal l is σ -*eligible* in a clause $C' \vee l$ if it is selected in C ($l \in \text{sel}(C)$), or there are no selected literals and $l\sigma$ is maximal in $C\sigma$. Strict σ -eligibility is defined in a like fashion, with maximality replaced by strict maximality. Where σ is empty, we sometimes speak of eligibility instead of σ -eligibility. In what follows, \mathcal{CS} is a multiset of literals that we refer to as *constraints*.

$$\frac{D' \vee f(\bar{t}_n) \approx t' \quad C' \vee s[f(\bar{s}_n)] \approx s'}{C' \vee D' \vee s[t'] \approx s' \vee \mathcal{CS}} \text{ SUP}$$

$$\frac{D' \vee x \approx t' \quad C' \vee s[f(\bar{s}_n)] \approx s'}{(C' \vee D' \vee s[t'] \approx s')\sigma} \text{VSUP}$$

where $\sigma = \{x \rightarrow f(\bar{s}_n)\}$, and $\mathcal{CS} = t_1 \not\approx s_1 \vee \dots \vee t_n \not\approx s_n$. Both rules share the following side conditions. Let t stand for either $f(\bar{t}_n)$ or x . For SUP, the substitution σ mentioned in the side conditions is of course empty.

- $t \approx t'$ is strictly σ -eligible.
- $s[f(\bar{s}_n)] \approx s'$ is strictly σ -eligible if positive and σ -eligible if negative.
- $t\sigma \not\approx t'\sigma$ and $s[f(\bar{s}_n)]\sigma \not\approx s'\sigma$.
- $C\sigma \not\approx D\sigma$

$$\frac{C' \vee f(\bar{t}_n) \approx v' \vee f(\bar{s}_n) \approx v}{C' \vee v \not\approx v' \vee f(\bar{s}_n) \approx v \vee \mathcal{CS}} \text{EQFACT} \quad \frac{C' \vee u' \approx v' \vee u \approx v}{(C' \vee v \not\approx v' \vee u \approx v)\sigma} \text{VEQFACT}$$

for EQFACT, $\mathcal{CS} = t_1 \not\approx s_1 \vee \dots \vee t_n \not\approx s_n$. For VEQFACT, either u or u' must be a variable and σ is the most general unifier of u and u' . The side conditions for EQFACT are:

- $f(\bar{s}_n) \approx v$ be eligible in C .
- $f(\bar{s}_n) \not\approx v$ and $f(\bar{t}_n) \not\approx v'$.

The side conditions for VEQFACT are:

- $u \approx v$ be σ -eligible in C .
- $u\sigma \not\approx v\sigma$ and $u'\sigma \not\approx v'\sigma$.

The calculus also contains the following resolution / unification inferences. We refer to these as unification inferences, because each inference represents carrying out a single step of the well-known Robinson unification algorithm [11].

$$\frac{C' \vee f(\bar{s}_n) \not\approx f(\bar{t}_n)}{C' \vee \mathcal{CS}} \text{DECOMPOSE} \quad \frac{C' \vee x \not\approx t}{C'\sigma} \text{BIND} \quad \frac{C' \vee s \not\approx s}{C'} \text{REFLDEL}$$

where for BIND, $\sigma = \{x \rightarrow t\}$ and x does not occur in t . For DECOMPOSE, $f(\bar{s}_n) \neq f(\bar{t}_n)$ and $\mathcal{CS} = t_1 \not\approx s_1 \vee \dots \vee t_n \not\approx s_n$. All three inferences require that the final literal be σ -eligible in $C\sigma$ (for DECOMPOSE and REFLDEL, σ is empty). We provide some examples to show how the calculus works.

Example 1. Consider the unsatisfiable clause set:

$$C_1 = f(x, g(x)) \not\approx t \quad C_2 = f(g(b), y) \approx t$$

A SUP inference between C_1 and C_2 results in clause $C_3 = t \not\approx t \vee x \not\approx g(b) \vee g(x) \not\approx y$. A REFLDEL inference on C_3 results in the clause $C_4 = x \not\approx g(b) \vee g(x) \not\approx y$. An application of BIND on C_4 with $\sigma = \{x \rightarrow g(b)\}$ results in $C_5 = g(g(b)) \not\approx y$. Another application of BIND, then leads to \perp .

Example 2. Consider the unsatisfiable clause set:

$$C_1 = x \approx c \quad C_2 = f(a, c) \not\approx t \quad C_3 = f(c, c) \approx t$$

A VSUP inference between C_1 and C_2 results in clause $C_4 = f(c, c) \not\approx t$. A SUP inference between C_3 and C_4 results in the clause $C_5 = t \not\approx t \vee c \not\approx c \vee c \not\approx c$. A triple application of REFLDEL starting from C_5 derives \perp .

Note 1. We abuse terminology and use *inference* and *inference rule* to refer both to schemas such as shown above, as well as concrete instances of such schemas. Given an inference ι , we refer to the tuple of its premises by $prems(\iota)$, to its maximal premise by $mprem(\iota)$, and to its conclusion by $concl(\iota)$.

4 Redundancy Criterion

We utilise Waldmann et al.'s framework [25] for proving the completeness of our calculus. Hence, our redundancy criterion is based on their intersected lifted criterion. In instantiating the framework, we roughly follow Bentkamp et al. [6]. Let the calculus defined above be referred to as *Inf*. We introduce a ground inference system *GInf* that coincides with standard superposition [3]. That is, it contains the well known three inferences, SUP, EQFACT and EQRES. We refer to these inferences by GSUP, GEQFACT and GEQRES to indicate that they are only applied to ground clauses. Following the notation of the framework, we write $Inf(N)$ ($GInf(N)$) to denote the set of all *Inf* (*GInf*) inferences with premises in a clause set N . We introduce a grounding function \mathcal{G} that maps terms, literals and clauses to the sets of their ground instances. For example, given a clause C , $\mathcal{G}(C)$ is the set $\{C\theta \mid \theta \text{ is a grounding substitution}\}$. We extend the function \mathcal{G} to clause sets by letting $\mathcal{G}(N) = \bigcup_{C \in N} \mathcal{G}(C)$ where N is a set of clauses.

A ground clause C is redundant with respect to a set of ground clauses N if there are clauses $C_1, \dots, C_n \in N$ such that for $1 \leq i \leq n$, $C_i \prec C$ and $C_1, \dots, C_n \models C$. The set of all ground clauses redundant with respect to a set of ground clauses N is denoted $GRed_{C_l}(N)$.

A clause C is redundant with respect to a set of clauses N , if for every $D \in \mathcal{G}(C)$, D is redundant with respect to $\mathcal{G}(N)$ or there is a clause $C' \in N$ such that $D \in \mathcal{G}(C')$ and $C \sqsupset C'$ where \sqsupset is the strict subsumption relation. That is $C \sqsupset C'$ if C is subsumed by C' , but C' is not subsumed by C . The set of all clauses redundant with respect a set of clauses N is denoted $Red_{C_l}(N)$.

In order to define redundant inferences, we have to pay careful attention to selection functions. For non-ground clauses, we fix a selection function *sel*. We then let $\mathcal{G}(sel)$ be a set of selection functions on ground clauses with the following property. For each $gsel \in \mathcal{G}(sel)$, for every ground clause C , there exists a clause D such that $C \in \mathcal{G}(D)$ and the literals selected in C by *gsel* correspond to those selected in D by *sel*. We write $GInf^{gsel}$ to show that the ground inference system *GInf* is parameterised by the selection function *gsel*. Let ι be an inference in *Inf*. We extend the grounding function \mathcal{G} to a family of grounding functions \mathcal{G}^{gsel}

for each $g_{sel} \in \mathcal{G}(sel)$. Each function $\mathcal{G}^{g_{sel}}$ maps terms, literals and clauses as above, and maps members of Inf to subsets of $GInf^{g_{sel}}$ as follows.³

Definition 1 (Ground Instance of an Inference). *Let ι be of the form $C_1, \dots, C_n \vdash E \vee \mathcal{CS}$. An inference $\iota_g \in GInf^{g_{sel}}$ is in $\mathcal{G}^{g_{sel}}(\iota)$ if it is of the form $C_1\theta, \dots, C_n\theta \vdash E\theta$ for some grounding substitution θ . In this case, we say that ι_g is the θ -ground instance of ι . Note that we ignore the constraints in the definition of ground instances.*

A ground inference $C_1, \dots, C_n, C \vdash E$ with maximal premise C is redundant with respect to a clause set N if for $1 \leq i \leq n$, $C_i \in GRed_{Cl}(N)$ or $C \in GRed_{Cl}(N)$ or there exist clauses $D_1, \dots, D_m \in N$ such that for $1 \leq i \leq m$, $D_i \prec C$ and $D_1, \dots, D_m \models E$. The set of all ground inferences redundant with respect to a set N is denoted $GRed_I^{g_{sel}}(N)$.

An inference ι is redundant with respect to a clause set N if for every $g_{sel} \in \mathcal{G}(sel)$ and for every $\iota' \in \mathcal{G}^{g_{sel}}(\iota)$, $\iota' \in GRed_I^{g_{sel}}(\mathcal{G}(N))$. In words, every ground instance of the inference is redundant with respect to $\mathcal{G}(N)$. We denote the set of all redundant inferences with respect to a set N as $Red_I(N)$.

A clause set N is saturated up to redundancy by an inference system Inf if every member of $Inf(N)$ is redundant with respect to N .

Note 2. Given the definition of clause redundancy above, the REFLDEL inference can be utilised as a *simplification* inference. That is, the conclusion of the inference renders the premise redundant.

5 Refutational Completeness

To prove refutational completeness we utilise the above mentioned framework of Waldmann et al. [25]. In particular, we use Theorem 14 from the paper to lift completeness from the ground level to the non-ground level. We bring Theorem 14 here for clarity and to keep the paper self contained. We then present it in our notation. Let $GRed = (GRed_I^{g_{sel}}, GRed_{Cl})$ and $Red = (Red_I, Red_{Cl})$

Theorem 14 (from Waldmann et al. [25]). *If $(GInf^q, Red^q)$ is statically refutationally complete w.r.t. \models^q for every $q \in Q$ and if for every $N \subseteq \mathbf{F}$ that is saturated w.r.t. $FInf$ and $Red^{\cap \mathcal{G}}$ there exists a q such that $GInf^q(\mathcal{G}^q(N)) \subseteq \mathcal{G}^q(FInf(N)) \cup Red_I^q(\mathcal{G}^q(N))$, then $(FInf, Red^{\cap \mathcal{G}})$ is statically refutationally complete w.r.t. $\models_{\mathcal{G}}^{\cap}$.*

Theorem 14 (from Waldmann et al. in our Notation). *If $(GInf^{g_{sel}}, GRed)$ is statically refutationally complete w.r.t. \models for every $g_{sel} \in \mathcal{G}(sel)$ and if for every clause set N that is saturated w.r.t. Inf and Red there exists a g_{sel} such that $GInf^{g_{sel}}(\mathcal{G}^{g_{sel}}(N)) \subseteq \mathcal{G}^{g_{sel}}(Inf(N)) \cup Red_I(\mathcal{G}^{g_{sel}}(N))$, then (Inf, Red) is statically refutationally complete w.r.t. $\models_{\mathcal{G}}$.*

³ When a grounding function $\mathcal{G}^{g_{sel}}$ acts on a clause, literal or term, we commonly drop the g_{sel} superscript as the selection function plays no role in the grounding of these.

Thus, in our context, the set Q is $\mathcal{G}(sel)$, the ground inference system $GInf^q$ maps to $GInf^{gsel}$, the ground redundancy criterion Red^q is $(GRed_I^{gsel}, GRed_{Cl})$ and the ground entailment relation \models^q maps to standard entailment on first-order clauses. Moreover, the non-ground inference system $FInf$ maps to Inf and the redundancy criterion $Red^{\mathcal{G}}$ maps to (Red_I, Red_{Cl}) . Note, that this final mapping is not exact, as the criterion $Red^{\mathcal{G}}$ does not allow for a tiebreaker ordering, such as the strict subsumption relation, to be utilised in the definition of non-ground redundancy. However, this mismatch can easily be repaired since Theorem 16 of the framework paper extends the result of Theorem 14 to the case where tiebreaker orderings are used.

As our ground inference systems $GInf^{gsel}$ are ground superposition systems, static refutational completeness with respect to standard entailment and standard redundancy is a famous result. See for example [2]. What remains for us to prove in order to apply Theorem 14 and show the static refutational completeness of Inf , is:

1. For every $gsel \in \mathcal{G}(sel)$, the grounding function \mathcal{G}^{gsel} is a grounding function in the sense of the framework.
2. For every clause set N saturated up to redundancy by Inf , there exists a $gsel \in \mathcal{G}(sel)$ such that $GInf^{gsel}(\mathcal{G}(N)) \subseteq \mathcal{G}^{gsel}(Inf(N)) \cup GRed_I^{gsel}(\mathcal{G}(N))$. In words, there exists a ground selection function such that every ground inference with that selection function and premises in $\mathcal{G}(N)$ is either the instance of a non-ground inferences with premises in N or is redundant with respect to $\mathcal{G}(N)$.

Lemma 1. *For every $gsel \in \mathcal{G}(sel)$, the grounding function \mathcal{G}^{gsel} is a grounding function in the sense of the framework.*

Proof. We need show that properties (G1) – (G3) defined by Waldmann et al. hold for grounding functions. These properties are:

- (G1) for every $\perp \in \mathbf{F}_\perp$, $\emptyset \neq \mathcal{G}(\perp) \subseteq \mathbf{G}_\perp$;
- (G2) for every $C \in \mathbf{F}$, if $\perp \in \mathcal{G}(C)$ and $\perp \in (G)_\perp$ then $C \in \mathbf{F}_\perp$;
- (G3) for every $\iota \in FInf$, if $\mathcal{G}(\iota) \neq undef$, then $\mathcal{G}(\iota) \subseteq Red_I(\mathcal{G}(concl(\iota)))$.

As properties (G1) and (G2) relate to the grounding of terms and clauses, and our grounding of these is fully standard we skip these. We prove (G3), which in our terminology is: for every $\iota \in Inf$, $\mathcal{G}^{gsel}(\iota) \subseteq GRed_I^{gsel}(\mathcal{G}(concl(\iota)))$. This can be achieved by showing that for every $\iota' \in \mathcal{G}^{gsel}(\iota)$, there exist clauses $\bar{C} \in \mathcal{G}(concl(\iota))$ such that $\bar{C} \models concl(\iota')$ and for each $C_i \in \bar{C}$, $C_i \prec mprem(\iota')$. In what follows, let θ be the substitution by which ι' is a grounding of ι .

If \mathcal{CS} is the empty set in $concl(\iota)$, then $concl(\iota)\theta = concl(\iota')$ and hence $concl(\iota)\theta \models concl(\iota')$. Moreover, $concl(\iota)\theta \in \mathcal{G}(concl(\iota))$ and thus $concl(\iota)\theta \prec mprem(\iota')$.

On the other hand, if \mathcal{CS} is not empty, let $u = f(\bar{t}_n)$ and $u' = f(\bar{s}_n)$ be the two terms within $prems(\iota)$ from which the constraints are created. By the existence of ι' , we have that $u\theta = u'\theta$, and hence that $t_i\theta = s_i\theta$ for $1 \leq i \leq n$. Hence, every

literal in $\mathcal{CS}\theta$ has the form $t \not\approx t$ and is trivially false in every interpretation. Thus, we still have $\text{concl}(\iota)\theta \models \text{concl}(\iota')$. Moreover, by the subterm property of the ordering \succ we have that $t_i\theta \not\approx s_i\theta$ is smaller than the maximal / selected literal of $\text{mprem}(\iota')$ for $1 \leq i \leq n$ and hence that $\text{concl}(\iota)\theta \prec \text{mprem}(\iota')$. \square

Lemma 2. *let σ be the most general unifier of terms s and s' , and θ be any unifier of the same terms. Then for any term t , $(t\sigma)\theta = t\theta$.*

Proof. Since σ is the most general unifier, there must be a substitution ρ such that $\sigma\rho = \theta$. Hence $(t\sigma)\theta = (t\sigma)\sigma\rho = t\sigma\rho = t\theta$ where the second to last step follows from the fact that σ is idempotent. \square

Lemma 3. *For every clause set N saturated by Inf , there exists a $\text{gsel} \in \mathcal{G}(\text{sel})$ such that $\text{GInf}^{\text{gsel}}(\mathcal{G}(N)) \subseteq \mathcal{G}^{\text{gsel}}(\text{Inf}(N)) \cup \text{GRed}_I^{\text{gsel}}(\mathcal{G}(N))$.*

Proof. For every $D \in \mathcal{G}(N)$ there must exist a clause $C \in N$ such that $D \in \mathcal{G}(C)$. Let \gg be an arbitrary well-founded ordering on clauses. We let $C = \mathcal{G}^{-1}(D)$ denote the \gg -smallest clause such that $D \in \mathcal{G}(C)$. We then choose the $\text{gsel} \in \mathcal{G}(\text{sel})$ that for a clause $D \in \mathcal{G}(N)$ selects the corresponding literals to those selected by sel in $\mathcal{G}^{-1}(D)$. Given this gsel , we need to show that every inference with premises in $\mathcal{G}(N)$ is either the ground instance of an inference with premises in N , or is redundant with respect to $\mathcal{G}(N)$.

A SUP inference is redundant if the term t replaced in the second premise occurs at or below a variable. The proof is exactly the same as in the standard proof of the completeness of superposition [3], so we don't repeat it. All other inferences can be shown to be the ground instance of inferences from clauses in N .

Let $\iota \in \text{GInf}^{\text{gsel}}$ be the following GSUP inference with premises in $\mathcal{G}(N)$.

$$\frac{D'\theta \vee t\theta \approx t'\theta \quad C'\theta \vee s\theta[t\theta] \approx s'\theta}{C'\theta \vee D'\theta \vee s\theta[t'\theta] \approx s'\theta}$$

where $\mathcal{G}^{-1}(D\theta) = D = D' \vee t \approx t'$, $\mathcal{G}^{-1}(C\theta) = C = C' \vee s \approx s'$ and ι fulfils all the side conditions of GSUP. Let σ be any substitution. The literal $t\theta \approx t'\theta$ being strictly maximal in $D\theta$ implies that $t\sigma \approx t'\sigma$ is strictly maximal in $D\sigma$ due to the stability under substitution of \succ . The literal $s\theta[t\theta] \approx s'\theta$ being (strictly) eligible in $C\theta$ with respect to gsel implies that $s\sigma \approx s'\sigma$ is strictly eligible in $C\sigma$ with respect to sel . Let p be the position of $t\theta$ within $s\theta$ and let u be the subterm of s at p . Since the term $t\theta$ does not occur below a variable of C , such a position must exist. Moreover, u cannot be a variable since if it was $t\theta$ would occur at a variable of C . As θ is a unifier of u and t , it must be the case that either t is a variable, or u and t have the same top symbol. Further, $D\theta \prec C\theta$ implies that $C\sigma \not\prec D\sigma$, $t\theta \succ t'\theta$ implies that $t\sigma \not\prec t'\sigma$, and $s\theta[t'\theta] \succ s'\theta$ implies $s\sigma \not\prec s'\sigma$. Thus, if t is not a variable, there exists the following SUP inference ι' from clauses D and C .

$$\frac{D' \vee t \approx t' \quad C' \vee s[u] \approx s'}{C' \vee D' \vee s[t'] \approx s' \vee \mathcal{CS}}$$

We have that $(C' \vee D' \vee s[t'] \approx s')\theta = \text{concl}(\iota)$. That is, the grounding of the conclusion of ι' less the constraint literals is equal to the conclusion of ι . Thus, ι is the θ -ground instance of ι' as per Definition 1. If t is a variable x , then there exists the following VSUP inference ι' from clauses D and C .

$$\frac{D' \vee x \approx t' \quad C' \vee s[u] \approx s'}{(C' \vee D' \vee s[t'] \approx s')\sigma}$$

Where $\sigma = \{x \rightarrow u\}$ is the most general unifier of t and u . Thus, we can use Lemma 2 to show that $\text{concl}(\iota')\theta = \text{concl}(\iota)$ and again ι is the θ -ground instance of ι' .

Let $\iota \in \text{GInf}^{gsel}$ be the following GEQFACT inference with premise in $\mathcal{G}(N)$.

$$\frac{C'\theta \vee u'\theta \approx v'\theta \vee u\theta \approx v\theta}{C'\theta \vee v\theta \not\approx v'\theta \vee u\theta \approx v\theta}$$

where $u'\theta = u\theta$, $\mathcal{G}^{-1}(C\theta) = C = C' \vee u' \approx v' \vee u \approx v$ and ι fulfils all the side conditions of GEQFACT. Let σ be any substitution. The literal $u\theta \approx v\theta$ being maximal in $D\theta$ implies that $u\sigma \approx v\sigma$ is maximal in $D\sigma$. Since θ is a unifier of u' and u , at least one of them must be a variable, or they must share a top symbol. Moreover, $u\theta \succ v\theta$ implies that $u\sigma \not\approx v\sigma$ and $u'\theta \succ v'\theta$ implies that $u'\sigma \not\approx v'\sigma$. If neither u nor u' is a variable, there exists the following EQFACT inference ι' from C .

$$\frac{C' \vee u' \approx v' \vee u \approx v}{C' \vee v \not\approx v' \vee u \approx v \vee CS}$$

We have $(C' \vee v \not\approx v' \vee u \approx v)\theta = \text{concl}(\iota)$, making ι the θ -ground instance of ι' as per Definition 1. If either u or u' is a variable there exists the following VEQFACT inference ι' from C .

$$\frac{C' \vee u' \approx v' \vee u \approx v}{(C' \vee v \not\approx v' \vee u \approx v)\sigma}$$

Where σ is the most general unifier of u and u' . Thus, we can use Lemma 2 to show that $\text{concl}(\iota')\theta = \text{concl}(\iota)$. Finally, let $\iota \in \text{GInf}^{gsel}$ be the following GEQRES inference with premise in $\mathcal{G}(N)$.

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

where $s\theta = s'\theta$, $\mathcal{G}^{-1}(C\theta) = C = C' \vee s \not\approx s'$ and ι fulfils all the side conditions of GEQRES. Let σ be any substitution. The literal $s\theta \not\approx s'\theta$ being eligible with respect to $gsel$ in $C\theta$ implies that $s \not\approx s'$ is eligible in C with respect to sel . Since θ is a unifier of s and s' , at least one of them must be a variable, or they must share a top symbol. If $s = s'$, then there exists the following REFLDEL inference ι' from C .

$$\frac{C' \vee s \not\approx s}{C'}$$

Otherwise we have two options. If either s (or analogously s') is a variable, then there is the following BIND inference ι' from C .

$$\frac{C' \vee x \not\approx s'}{C'\sigma}$$

Otherwise s and s' must share a top symbol and there is the following DE-COMPOSE inference ι' from C .

$$\frac{C' \vee f(\bar{s}_n) \not\approx f(\bar{t}_n)}{C' \vee CS}$$

In the first case, we have $\text{concl}(\iota')\theta = \text{concl}(\iota)$. In the second case, σ is the most general unifier of s and s' , so we can use Lemma 2 to show that $\text{concl}(\iota')\theta = \text{concl}(\iota)$. In the last case, we have that $C'\theta = \text{concl}(\iota)$. Thus in all cases, ι is the θ -ground instance of ι' . \square

Using Lemmas 1 and 3 we can instantiate Theorem 14 to prove the static refutational completeness of *Inf*. There is a slight issue here, as Theorem 14 gives us refutational completeness with respect to Herbrand entailment. That is $N \models M$ if $\mathcal{G}(N) \models \mathcal{G}(M)$. We would like to prove completeness with respect to entailment as defined in Section 2 (known as Tarski entailment). This issue can easily be resolved by showing that the two concepts are equivalent with regards to refutations which can be achieved in a manner similar to Bentkamp et al. (Lemma 4.19 of [6]).

Theorem 1 (Static refutational completeness). *For a set of clauses N saturated up to redundancy by *Inf*, $N \models \perp$ if and only if $\perp \in N$.*

Theorem 17 of Waldmann et al.'s framework can be used to derive dynamic refutational completeness from static refutational completeness. We refer readers to the framework for the formal definition of dynamic refutational completeness.

Theorem 2 (Dynamic refutational completeness). *The inference system *Inf* is dynamically refutationally complete with respect to the redundancy criterion (Red_I, Red_{CI}).*

6 Extending to Higher-Order Logic

We sketch how the ideas above can be extended to higher-order logic. This is ongoing research, and many of the technical details have yet to be fully worked out. Here, we provide a (very) informal description and then provide examples. The higher-order unification problem is undecidable and there can exist a potentially infinite number of incomparable most general unifiers for a pair of terms [12]. Existing higher-order paramodulation style calculi deal with this issue in two

main ways. One method is to abandon completeness and only unify to some pre-defined depth [22]. Another approach is to produce potentially infinite streams of unifiers and interleave the fetching of items from such streams with the standard saturation procedure[7]. Our idea is to solve easy sub-problems eagerly, such as when terms are first-order or in the pattern fragment [16], and add harder sub-problems as constraints. We then utilise dedicated inferences on negative literals to mimic the rules of Huet's well known (pre-)unification procedure [12]. We think that inferences similar to the following two, could be sufficient to achieve refutational completeness.

$$\frac{C' \vee x \bar{s}_n \not\approx f \bar{t}_m}{(C' \vee x \bar{s}_n \not\approx f \bar{t}_m)\{x \rightarrow \lambda \bar{y}_n. f(z_1 \bar{y}_n) \dots (z_m \bar{y}_n)\}} \text{IMITATE}$$

$$\frac{C' \vee x \bar{s}_n \not\approx f \bar{t}_m}{(C' \vee x \bar{s}_n \not\approx f \bar{t}_m)\{x \rightarrow \lambda \bar{y}_n. y_i(z_1 \bar{y}_n) \dots (z_p \bar{y}_n)\}} \text{PROJECT}$$

In both rules, each z_i is a fresh variable of the relevant type, and $x \bar{s}_n \not\approx f \bar{t}_m$ is selected in C . PROJECT has $k \leq n$ conclusions, one for each y_i of suitable type. We hope that through a careful definition of the selection function, along with the use of purification, we can avoid the need to apply unification inferences to flex-flex literals (negative literals where both sides of the equality have variable heads). Moreover, we are hopeful that the calculus we propose can remain complete without the need for inferences that carry out superposition beneath variables such as the FLUIDSUP rule of λ -superposition [7] and the SUBVARSUP rule of combinatory-superposition [9].

Example 3. Consider the unsatisfiable clause set:

$$C_1 = f y(x a)(x b) \not\approx t \quad C_2 = f c a b \approx t$$

A SUP inference between C_1 and C_2 results in clause $C_3 = t\sigma \not\approx t\sigma \vee x a \not\approx a \vee x b \not\approx b$ where $\sigma = \{y \rightarrow c\}$. Assume that the literal $x a$ is selected in C_3 . We can carry out either a PROJECT step on this literal or an IMITATE step. The result of a project step is $C_4 = (t\sigma \not\approx t\sigma \vee (\lambda z. z) a \not\approx a \vee x b \not\approx b)\{x \rightarrow \lambda z. z\}$. Applying the substitution and β -reducing results in $C_5 = t\sigma \not\approx t\sigma \vee a \not\approx a \vee b \not\approx b$ from which it is easy to reach a contradiction.

Example 4 (Example 1 of Bentkamp et al. [7]). Consider the unsatisfiable clause set:

$$C_1 = f a \approx c \quad C_2 = h(y b)(y a) \not\approx h(g(f b))(g c)$$

An EQRES inference on C_2 results in $C_3 = y b \not\approx g(f b) \vee y a \not\approx g c$. An IMITATE inference on the first literal of C_3 followed by the application of the substitution and some β -reduction results in $C_4 = g(z b) \not\approx g(f b) \vee g(z a) \not\approx g c$. A further double application of EQRES gives us $C_5 = z b \not\approx f b \vee z a \not\approx c$. We again carry out IMITATE on the first literal followed by an EQRES to leave us with

$C_6 = x b \not\approx b \vee f(x a) \not\approx c$. We can now carry out a SUP inference between C_1 and C_6 resulting in $C_7 = x b \not\approx b \vee c \not\approx c \vee x a \not\approx a$ from which it is simple to derive \perp via an application of IMITATE on either the first or the third literal. Note, that the empty clause was derived without the need for an inference that simulates superposition underneath variables, unlike in [7].

Example 5 (Example 2 of Bentkamp et al. [7]). Consider the unsatisfiable clause set:

$$C_1 = f a \approx c \quad C_2 = h(y(\lambda x. g(f x)) a) y \not\approx h(g c)(\lambda w x. w x)$$

An EQRES inference on C_2 results in $C_3 = y(\lambda x. g(f x)) a \not\approx g c \vee y \not\approx \lambda w x. w x$. Assuming that the second literal is selected,⁴ an EQRES inference results in $C_4 = (y(\lambda x. g(f x)) a \not\approx g c)\{y \rightarrow \lambda w x. w x\}$. Simplifying C_4 via applying the substitution and β -reducing, we achieve $g(f a) \not\approx g c$. Superposing C_1 onto this clause we end up with $C_5 = g c \not\approx g c$ from which the empty clause can easily be derived. Note again, that the empty clause has been derived without recourse to a FLUIDSUP-like inference.

7 Experimental Results

We implemented the calculus in the Vampire theorem prover [14]. We also implemented a variant of the calculus, that utilises fingerprint indices [19] to act as an imperfect filter. The completeness proof indicates that a superposition inference only needs to be carried out when the two terms can *possibly* unify. Therefore, we store terms in fingerprint indices, which act as fast imperfect filters for finding unification partners, and only carry out superposition inferences with terms returned by the index. This restricts, somewhat, the number of inferences that take place, at the expense of some loss of speed. Thus, it represents a midway path between eager unification and delayed unification. As a final twist, we implemented a version of the calculus that uses fingerprint indices as well as solving constraint literals of the form $x \not\approx t$ (where x is not a subterm of t) and $t \not\approx t$ eagerly. Thus, in this version of the calculus there is no need for the BIND and REFLDEL rules.

We compared each of these approaches with the standard superposition calculus implemented in Vampire. We refer to the standard calculus as VAMPIRE and the delayed inference calculus without fingerprint indices by VAMPIRE*.⁵ We refer to the delayed inference calculus with fingerprint indices by VAMPIRE[†].

⁴ Most orderings would select the first literal. In this case, we can still derive a contradiction, but the proof is longer.

⁵ Our implementation can be found at <https://github.com/vprover/vampire/tree/delayed-unification>. To run the new calculus, use option `-duc on`. To run the standard calculus, the option `duc` is set to `off`.

Finally, we refer to the calculus that eagerly solves some constraint literals by VAMPIRE^\ddagger .⁶

We tested these approaches against each other on benchmarks coming from CASC 2023 system competition [23]. As our new approach is not currently compatible with higher-order or polymorphic input, we restricted the comparison to monomorphic first-order problems. Namely, we used the 500 benchmarks in the FNE and FEQ categories. These are monomorphic, first-order benchmarks that either include equality (FEQ) or do not contain equality (FNE). All benchmarks in the set are theorems. The results can be seen in Table 1. All experiments were run on a node cluster located at The University of Manchester. Each node in the cluster is equipped with 192 gigabytes of RAM and 32 Intel[®] Xeon processors with two threads per core. Each configuration was given 100s of CPU time per problem and run in single core mode. VAMPIRE was run with options `--mode casc` which causes it to use a tuned portfolio of strategies. All other variants were run with options `--mode casc --forced_options duc=on` which forces the use of the new calculus on top of the aforementioned portfolio.

Approach	Solved	Uniques
VAMPIRE	430	110
VAMPIRE^*	238	0
VAMPIRE^\dagger	255	0
VAMPIRE^\ddagger	322	2

Table 1: Summary of experimental results

The calculi based on delayed unification perform badly in comparison to standard superposition. This is unsurprising, as syntactic first-order unification is already an efficient process. By replacing it with delayed unification, we gain little in terms of time, but pay a heavy penalty in terms of the number of inferences carried out. The use of fingerprint indices helps somewhat in mitigating this issue, but not a great deal. Eagerly solving trivial constraints shows more promise and is actually able to solve two problems that the standard calculus can not (within the time limit). These are the benchmarks `CSR036+3.p` and `LAT347+3.p`.

8 Related Work

The only other proof calculi that we are aware of that explicitly integrate unification rules at the calculus level, are the higher-order paramodulation calculi [8,22]

⁶ The code for both VAMPIRE^\dagger and VAMPIRE^\ddagger can be found at branch <https://github.com/vprover/vampire/tree/delayed-unif-with-fp>. VAMPIRE^\dagger was built from commit `c04a08feb5db3e7468a1fa` and VAMPIRE^\ddagger from commit `fa2f139302b6a7a6487e73`. Again, option `-duc on` is required for the new calculi to run.

and lazy paramodulation [21]. However, these calculi are paramodulation calculi and do not incorporate certain concepts of redundancy so crucial to the success of superposition provers. Moreover, the completeness proofs for these calculi are based on very different techniques to the Bachmair & Ganzinger style model building proofs commonly employed in the completeness proofs of superposition calculi.

There are other calculi that in some form do represent the folding of unification into the calculus, but the link between the unification rules and the calculus is less clear. For example, the recent work by one of the authors of this paper [13] relating to reasoning about linear arithmetic, moves theory reasoning relating to a number of equations from the unification algorithm to the calculus level. A different example, by another of this paper, is the combinatory-superposition calculus [9] which essentially folds higher-order combinatory unification into the calculus. In both cases, the relationship between the unification algorithm and the calculus rules is not obvious.

There are other methods of dovetailing unification with inference rules. For example, a unification procedure can be modified to return a stream of results. This stream can be interrupted in order to carry out further inferences and then returned to later. This is the approach taken by the higher-order Zipperposition prover [7] in order to handle the infinite sets of unifiers returned by higher-order unification. Conceptually, this is a very different solution to using constraints, since the intermediate terms created during unification are not available to the entire calculus as they are in our approach. Furthermore, from an implementation perspective, streams of unifiers are a far greater departure from the standard saturation architecture than the adding of constraints. Unification can also be partially delayed by preprocessing techniques such as Brand's modification method and its developments [5].

As mentioned in the introduction, abstraction resembles the basic strategy [4,15], where unification problems are added to the constraint part of a clause. Periodically, these constraints can be checked for satisfiability and clauses with unsatisfiable constraints removed. However, in the basic strategy, the constraints do not interact with the rest of the proof calculus. Moreover, redundancy of clauses can no longer be defined in terms of ground instances, but only in terms of ground instances that satisfy the constraints. This significantly affects the simplification machinery of superposition / resolution.

Unification with abstraction was first introduced, to the best of our knowledge, by Reger et al. in [17] in the context of theory reasoning. However, the concept was introduced in an ad-hoc fashion with no theoretical analysis of its impact on the completeness of the underlying calculus. Recently, the relationship between unification modulo an equational theory and unification with abstraction has been analysed [13] and a framework developed linking the two. It remains to explore whether the current work can fit into that framework.

9 Conclusion

We have developed a first-order superposition calculus that delays unification through the use of constraints, and proved its completeness. Whilst the calculus does not perform well in practice, we feel that the calculus and its completeness proof form a template that can be followed to prove the completeness of calculi that involve unification procedures more complex than syntactic first-order unification. For example unification modulo a set of equations E . Some of the crucial features of our approach are: (1) the carrying out of partial unification and adding the remaining unification pairs back as constraints, and (2) the ignoring of constraint literals in the definition of redundant inference. In particular, feature (1) may well be crucial in taming issues relating to undecidable unification problems. For example, in higher-order logic where unification is undecidable, it is common to run unification to a particular depth and then give up if termination has not occurred. Of course, this harms completeness. With our approach it should be possible to add the remaining unification pairs back as constraints and maintain completeness. In the future, we would like to generalise our approach into a framework that can be used to prove the completeness of a variety of calculi as long as the unification problem for the underlying terms meets certain conditions. We would also like to explore instantiating such a framework to prove the completeness of particular calculi of interest to us such as AC-superposition and higher-order superposition.

Acknowledgements.

We acknowledge funding from the ERC Consolidator Grant ARTIST 101002685, the TU Wien Doctoral College SecInt, and the FWF SFB project SpyCoDe F8504.

References

1. Baader, F., Snyder, W.: Unification theory. In: Robinson, A., Voronkov, A. (eds.) *Handbook of Automated Reasoning*, vol. II, chap. 8, pp. 447–533 (2001)
2. Bachmair, L., Ganzinger, H.: On restrictions of ordered paramodulation with simplification. In: *IJCAR*. LNCS, vol. 449, pp. 427–441. Springer (1990)
3. Bachmair, L., Ganzinger, H.: Rewrite-based equational theorem proving with selection and simplification. *Journal of Logic and Computation* **4**(3), 217–247 (1994)
4. Bachmair, L., Ganzinger, H., Lynch, C., Snyder, W.: Basic paramodulation and superposition. In: *CADE*. LNAI, vol. 607. Springer (1992)
5. Bachmair, L., Ganzinger, H., Voronkov, A.: Elimination of equality via transformation with ordering constraints (1997)
6. Bentkamp, A., Blanchette, J., Cruanes, S., Waldmann, U.: Superposition for lambda-free higher-order logic. *Logical Methods in Computer Science* **17** (2021)
7. Bentkamp, A., Blanchette, J.C., Tournet, S., Vukmirović, P., Waldmann, U.: Superposition with lambdas. In: *CADE*. LNCS, vol. 11716, pp. 55–73. Springer (2019)
8. Benzmüller, C., Sultana, N., Paulson, L.C., Theiß, F.: The higher-order prover LEO-II. *Journal of Automated Reasoning* **55**(4), 389–404 (2015)
9. Bhayat, A., Rege, G.: A combinator-based superposition calculus for higher-order logic. In: *IJCAR*. LNCS, vol. 12166, pp. 278–296. Springer (2020)

10. Domenjoud, E.: A technical note on AC-unification. The number of minimal unifiers of the equation $\alpha x_1 + \dots + \alpha x_p \doteq_{AC} \beta y_1 + \dots + \beta y_q$. *Journal of Automated Reasoning* **8** (1992)
11. Hoder, K., Voronkov, A.: Comparing unification algorithms in first-order theorem proving. In: Annual Conference on Artificial Intelligence. LNCS, vol. 5803, pp. 435–443. Springer (2009)
12. Huet, G.P.: A unification algorithm for typed λ -calculus. *Theoretical Computer Science* **1**(1), 27–57 (1975)
13. Korovin, K., Kovacs, L., Reger, G., Schoisswohl, J., Voronkov, A.: ALASCA: Reasoning in quantified linear arithmetic. In: TACAS. p. forthcoming. LNCS, Springer (2023)
14. Kovács, L., Voronkov, A.: First-order theorem proving and vampire. In: CAV. LNCS, vol. 8044, pp. 1–35. Springer (2013)
15. Nieuwenhuis, R., Rubio, A.: Basic superposition is complete. In: ESOP. LNCS, vol. 582, pp. 371–389. Springer (1992)
16. Nipkow, T.: Functional unification of higher-order patterns. In: LICS. pp. 64–74. IEEE Computer Society (1993)
17. Reger, G., Suda, M., Voronkov, A.: Unification with abstraction and theory instantiation in saturation-based reasoning. In: TACAS. LNCS, vol. 10805, pp. 3–22. Springer (2018)
18. Schulz, S.: E - a brainiac theorem prover. *AI Commun.* **15**(2,3), 111–126 (2002)
19. Schulz, S.: Fingerprint indexing for paramodulation and rewriting. In: IJCAR. LNCS, vol. 7364, pp. 477–483. Springer (2012)
20. Snyder, W., Gallier, J.: Higher-order unification revisited: Complete sets of transformations. *Journal of Symbolic Computation* **8**(1-2), 101–140 (1989)
21. Snyder, W., Lynch, C.: Goal-directed strategies for paramodulation. In: *Rewriting Techniques and Applications: 4th International Conference, RTA-91 Como, Italy, April 10–12, 1991 Proceedings* 4. pp. 150–161. Springer (1991)
22. Steen, A., Benzmüller, C.: The higher-order prover Leo-III. In: IJCAR. LNCS, vol. 10900, pp. 108–116. Springer (2018)
23. Sutcliffe, G., Suttner, C.: The state of CASC. *AI Communications* **19**(1), 35–48 (2006)
24. Voronkov, A.: AVATAR: The architecture for first-order theorem provers. In: CAV. LNCS, vol. 8559, pp. 696–710. Springer (2014)
25. Waldmann, U., Tourret, S., Robillard, S., Blanchette, J.: A comprehensive framework for saturation theorem proving. In: IJCAR. LNCS, vol. 12166, pp. 316–334. Springer (2020)