Superposition with Equivalence Reasoning and Delayed Clause Normal Form Transformation

Harald Ganzinger* Jürgen Stuber†

Abstract

This paper describes a superposition calculus where quantifiers are eliminated lazily. Superposition and simplification inferences may employ equivalences that have arbitrary formulas at their smaller side. A closely related calculus is implemented in the Saturate system and has shown useful on many examples, in particular in set theory. The paper presents a completeness proof and reports on practical experience obtained with the Saturate system.

1 Introduction

"Translating a conjecture into clause normal form before handing it over to the theorem prover is like shooting oneself into the foot before starting on a long hike." With these or similar words G. Huet alluded to the fact that clause normal form transformation may obscure the logical structure to an extent making subsequent proof search very difficult, if not impossible [10]. This paper describes an approach to automated first-order theorem proving based on superposition where a compromise is sought between preserving formula structure, in particular quantifiers, and performing logical reasoning, in particular formula simplification, efficiently on clausal data structures. More specifically, we will describe and prove refutationally complete a clausal inference system, where literals in clauses can be equations between terms or equivalences between arbitrary first-order formulas. Term equations give rise to the usual superposition inferences, and so do equivalences when the larger side in the formula ordering is an atomic formula. The smaller side of an equivalence can be any formula. If the larger side of an equivalence literal is a nonatomic formula and if that literal is selected for inference, the formula is simplified by an analytic expansion rule. This system allows one to reason with equivalences as they might arise from definitions or lemmas in a natural way, and in particular use orientable equivalences as simplification rules, even if they are guarded by side conditions in a clausal context.

A calculus closely related to what we describe below was implemented in the Saturate system [9] a few years ago and has proved to be useful in an encouraging

*MPI für Informatik, 66123 Saarbrücken, Germany. Email: hg@mpi-sb.mpg.de

†LORIA École des Mines de Nancy, 615 Rue du Jardin Botanique, 54600 Villers-lès-Nancy, France. Email: stuber@loria.fr

number of cases. We have carried out practical experiments on various formalizations of fragments of set theory and from a proof of the Church-Rosser property of the lambda calculus. Experience shows that the proof search becomes more goal-oriented and generates much fewer clauses until a proof is found. Although Saturate is comparatively slow, this allows it to be competitive with Vampire [15] and E-Setheo [16] on the set theory examples that were used in the first-order division of the last CASC competition [6].

The work in this paper can be viewed as an extension of the extended narrowing calculus [8, 18] whereby not only initial equivalences can be used for simplification, but also those that have side conditions (contextual simplification) and those generated dynamically during the saturation process. Our completeness proof reuses some of the constructions of the latter paper. [7] also presents a method for handling definitions more efficiently, attempting to recover equivalences from clauses and treating them with specific selection strategies for inferences. Admitting inferences on nonclausal formulas, the calculus described below is also related to the various calculi of nonclausal resolution and superposition [11, 12, 2, 4]. The main difference here is that we also admit quantifiers, and that the local nature of classical superposition inferences is better preserved, facilitating the development of efficient implementations of our calculus in a theorem prover. In [1] the author argues, by discussing some examples, that a tight integration of sequent calculus, tableaux, and resolution should be beneficial to proof search, but does not propose any concrete calculus in this regard.

The work started with the implementation efforts in the Saturate system, and now revisited in a theoretical setting, should also be viewed as a first step towards closing the large gap in logical notation used in interactive theorem provers for higher-order logic at one side, and present-day automated provers à la E-Setheo, SPASS, or Vampire at the other side of the spectrum. That gap needs to be closed for any successful integration of automated provers into interactive ones. To close that gap even further, first-order provers should also understand some restricted higher-order notation. Steps into this direction might be facilitated with our present results.

2 A Motivating Example

Consider the definition of extensionality on sets by

```
eq(A,B) <=> !X.(contains(A,X)<=>contains(B,X))
```

where `!` is the universal quantifier. A typical clause normal form of this equivalence (and saturated with respect to superposition by Saturate) would be this (with `s$3` denoting a Skolem function, and "*" denoting a clause in which the first negative literal is selected):

```
*   4(2) : eq(B,C),contains(B,D) -> contains(C,D)
*   5(2) : eq(B,C),contains(C,D) -> contains(B,D)
    9(4) : contains(B,s$3(C,B)),contains(C,s$3(C,B)),eq(C,B)
*  10(4) : contains(B,s$3(C,B)),contains(C,s$3(C,B)) -> eq(C,B)
   19(8) : eq(B,B)
```

Having these clauses in the context of other axioms for set theory can create a quite large search space with many redundancies. Treating equivalence equationally by rewriting and superposition is much better. This is the proof of the transitivity of `eq` (line 2) that Saturate finds (about 15 times faster than the purely clausal proof, and by purely deterministic steps) with an inference system such as the one proposed in this paper:

```
 1 : eq(A,B)==!C.(contains(A,C)<=>contains(B,C))                                  [input]
 2 + !A,B,C.(eq(C,B)and eq(B,A)=>eq(C,A)) ->                                      [input]
 3 + !A,B,C.(!D.(contains(C,D)<=>contains(B,D)) and
     !E.(contains(B,E)<=>contains(A,E))=>!F.(contains(C,F)<=>contains(A,F))) ->
                                                             [reduction of 2 by [1,1,1]]
 4 + !A,B.(!C.(contains(B,C)<=>contains(A,C)) and
     !D.(contains(A,D)<=>contains(s$4,D))=>!E.(contains(B,E)<=>contains(s$4,E))) ->
                                                                    [expansion from 3]
 5 + !A.(!B.(contains(A,B)<=>contains(s$3,B)) and
     !C.(contains(s$3,C)<=>contains(s$4,C))=>!D.(contains(A,D)<=>contains(s$4,D))) ->
                                                                    [expansion from 4]
 6 + !A.(contains(s$2,A)<=>contains(s$3,A)) and
     !B.(contains(s$3,B)<=>contains(s$4,B))=>!C.(contains(s$2,C)<=>contains(s$4,C)) ->
                                                                    [expansion from 5]
 7 + !A.(contains(s$2,A)<=>contains(s$3,A)) and !B.(contains(s$3,B)<=>contains(s$4,B))
                                                                    [expansion from 6]
 8 + !A.(contains(s$2,A)<=>contains(s$4,A)) ->                      [expansion from 6]
 9 + contains(s$2,A)==contains(s$3,A)                               [expansion from 7]
10 + contains(s$3,A)==contains(s$4,A)                               [expansion from 7]
11 + contains(s$2,A)==contains(s$4,A)                           [reduction of 9 by [10]]
12 + false                                                 [reduction of 8 by [11,L,L]]

Length = 12, Depth = 9, Search Depth = 0
```

Steps 3–10 expand the goal by rewriting with the definition of `eq` in 1, by applying analytic rules in the sequent calculus, and by skolemization. After this the proof finishes by rewriting using the equivalences 1, 9, 10, and 11, with "==" denoting structural equivalence (or equality on the propositional level). This proof is found without any search at all as each inference is simplifying. Note that some equivalences are in the input explicitly, and others come to the surface by proof search. This short proof was made possible by rewriting with equivalences such as clause 1 in which the larger side is a standard atom, but where the smaller side in the ordering can be any formula, possibly including quantifiers.

The fact that with the clausal form of the extensionality axiom the search space can be very large is demonstrated by the nontermination of provers including Saturate and SPASS on trivially consistent clause sets such as the one resulting by adding the definition of union

```
contains(union(A,B),X) <=> contains(A,X) or contains(B,X)
```

to the extensionality clauses. With the inference system proposed here, Saturate will orient both equivalences, the one for `eq` and the one for `union`, from left to right, and terminate immediately as no critical pair exists.

3 Language

We assume as given a first-order language consisting of a set of predicate symbols $\mathcal{P}$ and a set of function symbols $\mathcal{F}$, and an arity function $\alpha : \mathcal{P} \cup \mathcal{F} \to \mathbb{N}$.

$\mathcal{F}$ has to contain sufficiently many function symbols for skolemization. We also assume a countably infinite set of variables $\mathcal{X}$. We will implicitly use deBruijn notation for bound variables to obtain a canonical representation for formulas, however, we will usually write formulas in the standard way. In deBruijn notation bound variables are represented by natural numbers ≥ 1, the *deBruijn indices*, where n refers to the n-th quantifier above n. For example, $\forall x \exists y\, p(x, y)$ is written as $\forall \exists p(2, 1)$. We assume that deBruijn indices are modified appropriately whenever necessary, for example when quantifiers are removed. Terms will be built in the usual way from function symbols (which include constants as function symbols of arity 0), free variables from $\mathcal{X}$ and bound variables from $\mathbb{N}^{\geq 1}$. A *nonequational atom* is an atom of the form $p(t_1, \ldots, t_n)$ where p is a predicate symbol of arity n and $t_1, \ldots, t_n$ are terms. An *equation* is written $s \approx t$. An *atom* is either a nonequational atom or an equation. A *first-order formula* is built from nonequational atoms and equations, using the logical operators $\perp$ (false), $\top$ (true), $\vee$, $\wedge$, $\rightarrow$, $\approx$ (equivalence), $\forall$ and $\exists$. We use $\neg F$ as an abbreviation for $F \rightarrow \perp$. We say a formula is *trivial* if it is either $\top$ or $\perp$. Note that we use the same symbol for equality and for logical equivalence which, for soundness, implicitly requires a two-sorted approach. We let $\approx$ be symmetric, i.e., we do not distinguish $l \approx r$ and $r \approx l$, both for equations and for equivalences. We write $F =_s G$ to indicate that F and G are equal up to symmetry of $\approx$.

A (generalized) *literal* is an equation or an equivalence. Note that an arbitrary logical formula F can be expressed as the literal $F \approx \top$. Finally, a (generalized) *clause* is a multiset of literals. We write $\square$ for the empty clause, and we will omit the multiset braces around singletons. For example, we will write $L \cup C$ for the clause consisting of the literal L and the literals in C.

We call the signature consisting of $\mathcal{P}$ and $\mathcal{F}$ the *base signature*, while the full signature containing the base signature and all the logical operators will be the *extended signature*. A clause or formula is *closed* if it contains no free variables, and *ground* if it contains no variables (i.e. not even deBruijn indices). A *ground instance* of a clause, formula or term is an instance obtained by substituting terms over the base signature for free variables. So in general ground instances are not ground, only closed. Note that ground instantiation does not introduce new bound variables. We write $\{t_1/x_1, \ldots, t_n/x_n\}$ for the substitution that replaces x_i by t_i for $i \in \{1, \ldots, n\}$.

The *Herbrand universe* HU is the set of ground terms over the base signature, and the *Herbrand base* HB is the set of all nonequational ground atoms over the base signature plus all equations between terms in the Herbrand universe. A *(nonequational Herbrand) interpretation* I is a subset of the Herbrand base. A ground atom A is true in I if and only if $A \in I$. A quantified formula $\forall x\, F$ is true if $F\{t/x\}$ is true in I for all ground terms t. Analogously $\exists x\, F$ is true if there exists a ground instantiation of x that is true in I. Other logical operators are interpreted as usual, which extends the interpretation to all closed clauses. Clauses with free variables will be considered true in I if all its closed instances are true in I, i.e. free variables are considered to be universally quantified.

4 Ordering

In order to do the model construction and obtain refutational completeness we need a well-ordering on closed clauses with the following properties:

1. The ordering has to be a total simplification ordering on closed terms and formulas that is compatible with the symmetry of $\approx$.
   We can achieve this by an RPO over a total precedence that has multiset status for $\approx$ and lexicographic status for the other symbols.
2. The ordering must decrease when a quantified variable is instantiated by an arbitrary ground term, so deBruijn indices need to be larger than all ground terms over $\mathcal{F}$.
   In the precedence we let deBruijn indices be greater than function symbols.
3. $\top$ is the least and $\bot$ the next-to-least element.
   We put $\bot \succ \top$ at the bottom of the precedence.
4. To make the factoring inference decreasing we need that for s a term or a nonequational atom, $s \succ t_1$ and $s \succ t_2$ imply $s \approx t_1 \succ t_1 \approx t_2$.
   In the precedence we let $\approx$ be smaller than all function and predicate symbols.

The following precedence is suitable for our purposes:

$$\ldots \succ 2 \succ 1 \succ 0 \succ f \succ \approx \succ \rightarrow \succ \forall \succ \exists \succ \vee \succ \wedge \succ \bot \succ \top$$

Here numbers stands for the bound variables in deBruijn notation, and f for an arbitrary well-ordering on all function symbols. Predicate symbols may be placed arbitrarily above $\approx$ in the precedence. This particular choice of a precedence has the additional advantage that it orients the CNF transformation rules of Stuber [18], which can be useful for simplification and refinements of the calculus. We denote the RPO obtained from such a precedence by $\succ$.

We extend $\succ$ from terms and first-order formulas to literals by considering a literal $l \approx r$ as the multiset $\{l, r\}$ and comparing them by using the multiset extension of $\succ$. Clauses are multisets of their literals and are again naturally compared by the multiset extension of the literal ordering. We say a literal L is *maximal* in a clause $L \cup C$ if $L \succeq L'$ for all literals L' in C.

We will uniformly write $\succ$ for the orderings on terms, on formulas, on literals, and on clauses. When we compare a literal and a clause we consider the literal as a unit clause.

5 The Inference System

We begin by describing some properties that all our inferences will have. Inferences have a *main premise* C, a *conclusion* D, and zero or more *side premises* $C_1, \ldots, C_n$. Such an inference is written

$$\frac{C \qquad C_1 \quad \ldots \quad C_n}{D},$$

where we will always write the main premise first. The inference operates on the *main literal* in the main premise, which we will write as the first literal of the clause. A side premise C_i always has the form $l_i \approx r_i \ \cup \ C_i'$, and we assume that variables are renamed to be distinct from those of the other premises. Inferences on closed clauses, which we call *closed inferences*, will have the additional implicit side conditions that the main premise is greater than the side premises and the conclusion, that the main literal is maximal in the main premise, and that $(l_i \approx r_i) \succ C_i'$ and $l_i \succ r_i$ for each side premise C_i. For the actual nonground inferences we approximate the ordering, for example $\succ$ by $\not\preceq$. In inferences where a substitution (i.e. an mgu) is applied the comparison is made on the substituted clauses.

First we present the Elimination inference rules that expand clauses to clause normal form. To write them more concisely we adapt the unified notation of Smullyan to our literal syntax:

α	α_1	α_2
$F \land G \approx \top$	$F \approx \top$	$G \approx \top$
$F \lor G \approx \bot$	$F \approx \bot$	$G \approx \bot$
$F \to G \approx \bot$	$F \approx \top$	$G \approx \bot$

β	β_1	β_2
$F \land G \approx \bot$	$F \approx \bot$	$G \approx \bot$
$F \lor G \approx \top$	$F \approx \top$	$G \approx \top$
$F \to G \approx \top$	$F \approx \bot$	$G \approx \top$

γ	$\gamma(t)$
$\forall x\, F \approx \top$	$F\{t/x\} \approx \top$
$\exists x\, F \approx \bot$	$F\{t/x\} \approx \bot$

δ	$\delta(t)$
$\forall x\, F \approx \bot$	$F\{t/x\} \approx \bot$
$\exists x\, F \approx \top$	$F\{t/x\} \approx \top$

$\bot$ Elimination $\dfrac{\bot \approx \top \ \cup \ C}{C}$

α Elimination $\dfrac{\alpha \ \cup \ C}{\alpha_i \ \cup \ C}$ for $i \in \{1, 2\}$.

β Elimination $\dfrac{\beta \ \cup \ C}{\beta_1 \ \cup \ \beta_2 \ \cup \ C}$

γ Elimination $\dfrac{\gamma \ \cup \ C}{\gamma(z) \ \cup \ C}$

where z is a fresh variable.

δ Elimination $\dfrac{\delta \ \cup \ C}{\delta(f(x_1, \ldots, x_n)) \ \cup \ C}$

where f is a fresh skolem function and $x_1, \ldots, x_n$ are the free variables in δ.

$\approx \top$ Elimination $\dfrac{(l \approx r) \approx \top \ \cup \ C}{l \approx r \ \cup \ C}$

Positive Equivalence Elimination 1 $\dfrac{l \approx r \ \cup \ C}{l \approx \bot \ \cup \ r \approx \top \ \cup \ C}$

where l and r are nontrivial formulas, $l \not\preceq r$, and l is not a nonequational atom.

Positive Equivalence Elimination 2 $\dfrac{l \approx r \ \cup\ C}{l \approx \top \ \cup\ r \approx \bot \ \cup\ C}$

where l and r are nontrivial formulas, $l \not\preceq r$, and l is not a nonequational atom.

Negative Equivalence Elimination 1 $\dfrac{(l \approx r) \approx \bot \ \cup\ C}{l \approx \top \ \cup\ r \approx \top \ \cup\ C}$

where l and r are formulas.

Negative Equivalence Elimination 2 $\dfrac{(l \approx r) \approx \bot \ \cup\ C}{l \approx \bot \ \cup\ r \approx \bot \ \cup\ C}$

where l and r are formulas.

The second part of the calculus is a two-sorted superposition calculus, which is very similar to standard superposition.

Reflexivity Resolution $\dfrac{(s \approx t) \approx \bot \ \cup\ C}{C\sigma}$

where (i) s and t are terms, and (ii) σ is a most general unifier of s and t.

Negative Superposition $\dfrac{(s[l'] \approx t) \approx \bot \ \cup\ C \qquad l \approx r \ \cup\ D}{((s[r] \approx t) \approx \bot \ \cup\ C \ \cup\ D)\sigma}$

where (i) $s[l']$ and t are terms, (ii) l' is not a variable, (iii) σ is a most general unifier of l and l', and (iv) $s[l']\sigma \not\preceq t\sigma$.

Positive Superposition $\dfrac{s[l'] \approx t \ \cup\ C \qquad l \approx r \ \cup\ D}{(s[r] \approx t \ \cup\ C \ \cup\ D)\sigma}$

where (i) l' is not a variable, (ii) σ is a most general unifier of l and l', (iii) $s[l']\sigma \not\preceq t\sigma$, and (iv) $(s[l'] \approx t)\sigma \not\preceq C\sigma$.

$\approx$ Factoring $\dfrac{l \approx r \ \cup\ l' \approx r' \ \cup\ C}{((r \approx r') \approx \bot \ \cup\ l' \approx r' \ \cup\ C)\sigma}$

where (i) σ is a most general unifier of l and l', and (ii) $l\sigma \not\preceq r\sigma \not\prec r'\sigma$.

We let ES denote this set of inferences.

If we restrict the input to a set of clauses containing only literals of the forms $s \approx t$ and $(s \approx t) \approx \bot$ where s and t are terms, none of the Elimination inferences is applicable, and we recover a syntactic variant of the standard superposition calculus [14], with exactly the same ordering restrictions. However, if we consider clause sets containing only literals of the form $A \approx \top$ or $A \approx \bot$, we do not get the standard ordered resolution calculus. Instead we need factoring on both positive and negative literals and obtain a resolution inference derived from Positive Superposition that restricts both resolved literals to be strictly maximal. Nevertheless, a calculus extended by selection (see Section 7) can force negative literals to be treated by Negative Superposition, which leads to a syntactical variant of standard ordered resolution with selection.

6 Refutational Completeness

We prove the refutational completeness of ES in the presence of strong redundancy criteria using the reduction-of-counterexamples framework of Bachmair and Ganzinger [4]. We assume that the reader is familiar with this method.

We will construct a Herbrand model by well-ordered recursion over $\succ$ on closed clauses. We construct a rewrite system R that contains rules of the form $l \Rightarrow r$ where either l and r are ground terms, or where l is a nonequational ground atom and r is a closed formula. We will write $s \Rightarrow_R t$ or just $s \Rightarrow t$ when s rewrites to t, $\overset{+}{\Rightarrow}$ for the transitive closure of $\Rightarrow$, $\overset{*}{\Rightarrow}$ for the reflexive-transitive closure of $\Rightarrow$, and $s \Downarrow t$ if $s \overset{*}{\Rightarrow} u \overset{*}{\Leftarrow} t$. We will construct R in such a way that all left-hand sides of rules are irreducible by other rules, which implies confluence. To ensure termination we will require $l \succ r$ for all rules.

Suppose R is such a rewrite system. We define the *rewrite closure* of R, written R^*, as the smallest set of closed first-order formulas that satisfies the following conditions:

1. If $F \overset{+}{\Rightarrow} G$ and $G \in R^*$ then $F \in R^*$.
2. $\top \in R^*$.
3. $s \approx s \in R^*$ for all ground terms s.
4. If $F \in R^*$ and $G \in R^*$ then $F \wedge G \in R^*$.
5. If $F \in R^*$ or $G \in R^*$ then $F \vee G \in R^*$.
6. If $F \notin R^*$ or $G \in R^*$ then $F \rightarrow G \in R^*$.
7. If $F \in R^*$ iff $G \in R^*$ then $F \approx G \in R^*$.
8. If $F\{t/x\} \in R^*$ for all ground terms t then $\forall x\, F \in R^*$.
9. If $F\{t/x\} \in R^*$ for some ground term t then $\exists x\, F \in R^*$.

R^* can be understood as a minimal (first-order, equality) Hintikka set containing R. This definition is by induction over $\succ$. In particular in case (6), the only one involving negation, the formulas in the condition are smaller than the formula in the conclusion, so R^* is well-defined. By restricting R^* to the Herbrand base it becomes a Herbrand interpretation, and we obviously have $F \in R^*$ if and only if $\mathrm{HB} \cap R^* \models F$. In the following we will also write R^* for this Herbrand interpretation.

Lemma 1 *The set of term equations in R^* is a congruence.*

Lemma 2 *Let $l \Rightarrow r$ be a rule in R. Then $R^* \models l \approx r$.*

We assume as given a set N of clauses and build an interpretation, intended to be a model of N, from the ground instances of N. For each closed clause C we will define a set R_C of rewrite rules derived from clauses smaller than C, the Hintikka set $I_C = R_C^*$ derived from R_C, and a set Δ_C that contains either a rewrite rule derived from C or is empty. The definition of Δ_C depends on R_C and I_C; it is well-defined, since R_C and I_C only depend on Δ_D for D smaller than C. Formally we let

$$\Delta_C = \begin{cases} \{l \Rightarrow r\} & \text{if (i) } I_C \not\models C', \\ & \quad \text{(ii) } C = l \approx r \ \cup\ C' \text{ is a ground instance of a clause in } N, \\ & \quad \text{(iii) } l \text{ and } r \text{ are terms, or } l \text{ is a nonequational atom,} \\ & \quad \text{(iv) } (l \approx r) \succ C', \\ & \quad \text{(v) } l \succ r, \\ & \quad \text{(vi) } l \text{ is irreducible by } R_C, \text{ and} \\ & \quad \text{(vii) } (R \cup \{l \Rightarrow r\})^* \not\models C'; \text{ or} \\ \emptyset & \text{otherwise.} \end{cases}$$

$$R_C = \bigcup_{D \prec C} \Delta_D$$

$$I_C = R_C^*$$

To obtain an interpretation for all ground instances of N we let

$$R_N = \bigcup_C \Delta_C \qquad \text{and} \qquad I_N = R_N^*.$$

We say that a closed clause C *produces* $l \Rightarrow r$ in I_N if $l \Rightarrow r$ is in Δ_C for the set of clauses N. In that case we say that C is *productive* in I_N.

Lemma 3 *If $C = l \approx r \ \cup\ C'$ produces $l \Rightarrow r$ then C' is false in I_N.*

We now come to the proof of the reduction property of ES, which is the main part in the proof of refutational completeness. We call a closed clause C a *counterexample* for I_N if it is a ground instance of a clause in N and false in I_N. Since closed clauses are well-ordered by $\succ$, there exists a least counterexample whenever $I_N \not\models N$. Let C be a counterexample for I_N and let $\mathcal{I}$ be a closed inference with main premise C, side premises $C_1, \ldots, C_n$ and conclusion D. We say that $\mathcal{I}$ *reduces the counterexample* C (with respect to I_N) if $I_N \models \neg D$, $C \succ D$ and the side premises are productive in I_N. An inference system Calc has the *reduction property for counterexamples* (with respect to I) if there is a ground instance of an inference in Calc that reduces C for I_N for any set N of ground clauses such that I_N has the least counterexample $C \neq \Box$.

Lemma 4 *ES has the reduction property for counterexamples.*

Proof: Let N be a set of clauses, let C be the least counterexample in I_N and suppose $C \neq \Box$. Then $C = L \ \cup\ C'$ with $L = l \approx r$ maximal in C. Since C is false in I_N, we have $l \neq_s r$, and we may assume $l \succ r$ without loss of generality. C is not productive, as $l \Rightarrow r \in R_N$ would imply $I_N \models l \approx r$. We do case analysis on the possible causes, i.e. the condition on $\Delta_C \neq \emptyset$ that is violated:

(iii) Suppose l is a formula, but not a nonequational atom, i.e. l has a logical operator at its root. We do a case analysis over the form of l and r.

(iii.1) l cannot be $\top$, as $\top$ is minimal in $\succ$ and r must be smaller.

(iii.2) Suppose $l = \bot$. Then $r = \top$, and $\bot$ Elimination reduces C.

(iii.3–5) If L is an α, β, or γ-formula, then the corresponding elimination inference reduces C.

(iii.6) Suppose L is a δ-formula. L being false in I_N implies that for all ground terms t over the base signature the formula $\delta(t)$ is false in I_N. Let σ be the substitution used to derive C from a clause in N, and let $x_1, \ldots, x_n$ be the free variables in its literal corresponding to L. Then the ground instance

$$\frac{\delta \ \cup \ C'}{\delta(f(x_1\sigma, \ldots, x_n\sigma)) \ \cup \ C'}$$

of δ Elimination has a conclusion that is false and smaller than C, since bound variables are larger in the ordering than any ground term over the base signature. Hence it reduces C. Note that f is in the base signature and that freshness is not needed for the completeness proof.

(iii.7) Suppose $L = (s \approx t) \approx \top$. Then $\approx \top$ Elimination reduces C.

(iii.8) Suppose $L = (s \approx t) \approx \bot$.

(iii.8.1) Suppose s and t are terms. Then $I_N \models s \approx t$.

(iii.8.1.1) If $s = t$ then Reflexivity Resolution reduces C.

(iii.8.1.2) Otherwise without loss of generality $s \succ t$ and $s \Downarrow_{R_N} t$, so s is reducible by R_N. We have $C = \hat{C}\sigma$ for some clause $\hat{C}$ in N and a ground substitution σ, where $\hat{C} = (\hat{s} \approx \hat{t}) \approx \bot \ \cup \ \hat{C}'$ with, among others, $\hat{s}\sigma = s$. If the reduction step were in a variable position in $\hat{s}$, then we could reduce σ to τ and obtain a smaller ground instance $\hat{C}\tau$ of N that would also be a counterexample of I_N, a contradiction to the minimality of C.

s is reduced by some rule $l' \Rightarrow r'$ in R_N, which has been produced by some clause $D = l' \approx r' \ \cup \ D'$. Since l' is a subterm of s we have $s \succeq l$, which implies $l \succ l'$ and in turn $C \succ D$, so s is reducible already by R_C. We have the ground instance of Negative Superposition

$$\frac{(s[l'] \approx t) \approx \bot \ \cup \ C' \qquad l' \approx r' \ \cup \ D'}{(s[r'] \approx t) \approx \bot \ \cup \ C' \ \cup \ D'}$$

which reduces C. The conclusion is smaller than C due to monotonicity, and the main literal preserves its truth value since $l' \approx r'$ is valid in I_N and since $\approx$ is interpreted by a congruence in I_N.

(iii.8.2) Otherwise s and t are formulas. Since L is false in I_N the equivalence $s \approx t$ is true, so either both s and t are true or both are false. Thus one of the Negative Equivalence Elimination inferences

$$\frac{(s \approx t) \approx \bot \ \cup \ C'}{s \approx \top \ \cup \ t \approx \top \ \cup \ C'} \quad \text{and} \quad \frac{(s \approx t) \approx \bot \ \cup \ C'}{s \approx \bot \ \cup \ t \approx \bot \ \cup \ C'}$$

reduces C.

(iii.9) Otherwise both l and r are formulas distinct from $\top$ and $\bot$, and l is not a nonequational atom. In this case one of the Positive Equivalence Elimination inferences reduces C.

(iv) From now on we may assume that (iii) holds, so l is either a term or a nonequational atom. Suppose that L is not strictly maximal in C, so that there is another literal L' in C with $L =_s L'$. Let $L' = l' \approx r'$ with $l =_s l'$ and $r =_s r'$. The inference

$$\frac{l \approx r \;\cup\; l' \approx r' \;\cup\; C''}{(r \approx r') \approx \bot \;\cup\; l' \approx r' \;\cup\; C''}$$

is a ground instance of Equality Factoring. By property (4) of the ordering the literal $(r \approx r') \approx \bot$ is smaller than $l \approx r$, as l has a function or predicate symbol at the root and $l \succ r, r'$. By reflexivity of $\approx$ this literal is also false in I_N, so this inference reduces C.

(vi) From now on we may assume that (iv) holds, i.e. $L \succ C'$. Suppose L is reducible by R_C. Then $l = l[l']$ and $l' \Rightarrow r' \in R_C$ is produced by some clause $D = l' \approx r' \;\cup\; D'$ with $l' \succ r'$ and $(l' \approx r') \succ D'$. By the same argument as in (iii.8.1.2) the ground substitution that leads to this instance is irreducible by R_N. We have the ground instance of Positive Superposition

$$\frac{l[l'] \approx r \;\cup\; C' \qquad l' \approx r' \;\cup\; D'}{l[r'] \approx r \;\cup\; C' \;\cup\; D'}$$

which reduces C by the same argument as in (iii.8.1.2).

(vii) Otherwise $(R_C \cup \{l \Rightarrow r\})^* \models C'$. Then $C' = L' \;\cup\; C''$ and $(R_C \cup \{l \Rightarrow r\})^* \models L'$, where $L \succeq L'$. L' must contain l as a subterm to be affected by the rule $l \Rightarrow r$, so $L' = l' \approx r'$ with $l =_s l' \succ r \succeq r'$. For L' to become true in $(R_C \cup \{l \Rightarrow r\})^*$ we must have $r \approx r'$ true in $(R_C \cup \{l \Rightarrow r\})^*$. This implies $I_N \models r \approx r'$, because no clause greater or equal than C can produce a rule that affects r or r'. The inference

$$\frac{l \approx r \;\cup\; l' \approx r' \;\cup\; C''}{(r \approx r') \approx \bot \;\cup\; l' \approx r' \;\cup\; C''}$$

is a ground instance of Equality Factoring whose conclusion is smaller than C by the same argument as in (iv), so it reduces C. □

We call a closed clause C *redundant* in N if there exist ground instances $C_1, \ldots, C_k$ of clauses in N such that $C \succ C_i$ for $i = 1, \ldots, k$ and $C_1, \ldots, C_k \models C$. A (nonclosed) clause is *redundant* if all its ground instances are redundant.

For the reduction of a least counterexample by an inference redundant clauses are irrelevant. The least counterexample cannot be redundant, as being implied by smaller true clauses would contradict it being false. Any productive clause D used as a side premise cannot be redundant either, as it would also be implied by smaller true clauses, which contradicts the condition for productivity of being false in I_D. Hence redundant clauses can be ignored in inferences.

We may *simplify* a clause C to some clauses $D_1, \ldots, D_n$ in N if these clauses follow logically from $N \cup \{C\}$ and if C is redundant in $N \cup \{D_1, \ldots, D_n\}$. A closed inference

$$\frac{C \qquad C_1 \quad \ldots \quad C_n}{D}$$

is *redundant* in N if there exist ground instances $D_1, \ldots, D_k$ of clauses in N such that $C \succ D_i$ for $i = 1, \ldots, k$ and $D_1, \ldots, D_k, C_1, \ldots, C_n \models D$. Again, a (nonclosed) inference is *redundant* if all its ground instances are redundant. An important property of redundancy is that it is preserved when redundant clauses are removed. A set N of clauses is *saturated* with respect to ES if all inferences in ES with premises in N are redundant. The reduction property implies refutational completeness for ES:

Theorem 5 *Let N be a set of clauses that is saturated with respect to ES. If N is not satisfiable then $\square \in N$.*

For the δ Elimination inferences the conclusion is not a logical consequence of the premise, as Skolemization only preserves satisfiability. However, the conclusion implies the premise. Premise and conclusion become logically equivalent in the presence of the implication in the other direction:

$$\forall x_1, \ldots, x_n.(\exists x.F) \rightarrow F\{f(x_1, \ldots, x_n)/x\} \tag{1}$$

We call (1) the *skolem axiom* for the *skolem function symbol f* with respect to $\exists x.F$. We call a set of skolem axioms S *fresh* with respect to a set of propositions N if no skolem function symbol of S occurs in N, and there is only one skolem axiom for every skolem function symbol in S. A fresh set of skolem axioms is always obtained when fresh function symbols are used for skolemization.

Proposition 6 *Let N be a set of propositions and S a set of skolem axioms that is fresh with respect to N, and let I be a model of N. Then there exists a model I' of $N \cup S$.*

For a suitable set of skolem axioms this proposition isolates the argument that skolemization preserves satisfiability, and allows us to use the simpler notion of logical equivalence elsewhere.

A *theorem proving derivation* with respect to a set S of skolem axioms is a sequence of sets of clauses $N_0 \vdash N_1 \vdash \ldots$ such that for all steps $N_i \vdash N_{i+1}$, $i \geq 0$, either (1) $N_{i+1} = N_i \cup M$ for some set M of clauses such that $N_i \cup S \models M$, or (2) $N_{i+1} = N_i \setminus M$ for some set M of clauses such that each clause C in M is redundant in N_i. In case (1) we call the step a *deduction step* and in case (2) a *deletion step*. For such a derivation the set of *persistent clauses* N_∞ is defined as $N_\infty = \bigcup_{i \geq 0} \bigcap_{j \geq i} N_j$. Then one can easily prove that N_0 is satisfiable if and only if N_∞ is satisfiable.

A theorem proving derivation is called *fair* (with respect to ES) if all inferences in ES from clauses in N_∞ are redundant in N_i for some $i \geq 0$. Since the conclusions of ground inferences are always smaller than the main premise, an inference can be made redundant by a deduction step that adds its conclusion. Thus a fair derivation can always be obtained by considering inferences in a fair way, i.e. not delaying an inference ad infinitum, and adding their conclusion if they are not already known to be redundant by some suitable sufficient criterion.

Lemma 7 *Let $N_0 \vdash N_1 \vdash \ldots$ be a fair theorem proving derivation. Then N_∞ is saturated up to redundancy.*

Theorem 8 *Let $N_0 \vdash N_1 \vdash \ldots$ be a fair theorem proving derivation. If N_0 is not satisfiable then $\square \in N_\infty$.*

7 Refinements

As is the case with other superposition calculi we can apply various refinements.

Certain of our inferences, in particular all of the Elimination ones, are simplifications. Instead of using them exactly in the way the inference system suggests we may use other simplifications. For example, we may simplify $(A \approx F) \approx \bot$ to $A \approx \neg F$, as the two are logically equivalent and the second is smaller in $\succ$. On the other hand we may choose to expand some equivalences that are orientable, using Positive Equivalence Elimination as a simplification rule. We may use any equivalence that holds in the model and is orientable w.r.t. the ordering for rewriting inside a formula, for example the CNF transformation rules of Stuber [18], or equivalences in N. More generally we may even use clauses for contextual reduction of formulas.

Another modification is to add selection to the calculus. A formula F is in a *negative position* in a clause C if F being false in I implies that C is true in I. For example, the equation in the Negative Superposition inference is in a negative position. We may select some equation in any negative position in a clause, and then apply suitably generalized Reflexivity Resolution and Negative Superposition inferences at this position, instead of any other inference with this clause. We may also select a nonequational atom and use equivalences to do superposition in a similar way. This refinement is complete, since any minimal counterexample with a selected atom can be reduced in a similar way as by Reflexivity Resolution and Negative Superposition in the proof above. We may further generalize this to the selection of more than one position, which leads to Hyperparamodulation and Hyperresolution inferences.

Theorem proving with built-in theories over terms [3, 17] should pose no new fundamental problems, as should adding the basic restriction [5, 13].

8 Experiments

A calculus that differs from the one described before only with respect to minor technicalities related to clause representation had been implemented previously in the Saturate system. The expansion inference rules are implemented in a structure preserving manner by introducing new names for the subformulas as suggested in [19]. Skolem functions are associated with each quantified subformula at parse time so that if that subformula becomes produced in different contexts the same function is used. One can choose between eager and lazy expansion of logical symbols other than equivalences. In the presence of structural expansion, eager expansion does not create duplication, and contextual rewriting, as it is implemented in Saturate, can discharge contexts better if they do not contain any general formulas.

In Figure 1 we summarize Saturate's performance on the 40 problems from the TPTP category SET that were chosen for the CASC-18 competition in the first-order division [6]. We compare Saturate's results with the performance of Vampire 5.0 and E-Setheo csp02 at that competition ("–" means no proof found at the competition with a time limit of 600 seconds; "+" means proof found).

Problem	Saturate		V	ES	Problem	Saturate		V	ES
	#clauses	time/s				#clauses	time/s		
SET014+4	84	1.8	–	–	SET630+3	3049	116.8	+	+
SET017+1	–	–	+	+	SET640+3	293	6.0	+	+
SET067+1	138	5.8	+	+	SET646+3	–	–	+	+
SET076+1	161	4.4	+	–	SET647+3	3360	236.0	+	+
SET086+1	285	9.6	+	+	SET648+3	3345	251.1	+	+
SET096+1	214	8.1	+	+	SET649+3	–	–	+	+
SET108+1	–	–	+	+	SET651+3	–	–	+	+
SET143+3	107	2.0	+	+	SET657+3	456	16.0	+	+
SET143+4	71	1.1	+	+	SET669+3	–	–	+	+
SET171+3	152	2.9	–	–	SET671+3	–	–	+	–
SET580+3	95	1.7	+	+	SET673+3	–	–	+	–
SET601+3	207	5.3	–	–	SET680+3	784	46.2	+	+
SET606+3	93	1.3	+	+	SET683+3	143	4.6	+	–
SET607+3	77	1.1	+	+	SET684+3	–	–	+	–
SET609+3	136	2.1	–	–	SET686+3	–	–	+	+
SET611+3	325	11.0	+	+	SET716+4	193	8.8	–	–
SET612+3	145	9.9	–	–	SET724+4	380	18.1	–	–
SET614+3	126	9.5	–	–	SET747+4	–	–	–	–
SET615+3	203	11.1	–	–	SET764+4	241	6.2	+	+
SET624+3	172	10.2	+	+	SET770+4	174	5.0	–	–
solved	18		13	12	solved	11		16	12

Figure 1: Experimental results on the SET problems from the CASC-18 FOF competition (V: Vampire 5.0, ES: E-Setheo csp02)

Saturate was run with a clause limit of 10000 clauses and a time limit of 300 seconds. All results were obtained in auto mode. In particular the ordering for orienting equivalences was computed without user interaction. An iterative deepening strategy for proof search was employed.

Saturate is an experimental system written in Prolog and is comparatively slow. Therefore the runtimes (in seconds on a 2GHz notebook) are not too interesting. What we want to emphasize is the relatively small number of clauses generated on those problems where Saturate finds the proof. Vampire typically generates up to several orders of magnitude more clauses than Saturate. As an extreme case, for proving SET143+3 Vampire generates almost 8 million clauses.

One should also say that most of these problems are trivial, so we really do want to find the proof without much search, and the iterative deepening (proof depth) strategy for inference selection turned out to be quite useful here. We believe the reason why Vampire and E-Setheo do not perform well on some of these examples is due to their inability to suitably treat equivalences occurring in extensionality axioms for set equality and the subset relation. Another problem with some of the examples is that they consist of a relatively large axiomatization where only few axioms are needed in each case for the proof of the respective goal. Saturate also has problems with redundant axiomatizations and small changes can have a big effect on proof finding.

References

[1] Arnon Avron. Gentzen-type systems, resolution and tableaux. *Journal of Automated Reasoning*, 10(2):265–281, 1993.

[2] Leo Bachmair and Harald Ganzinger. Non-clausal resolution and superposition

with selection and redundancy criteria. In *Logic Programming and Automated Reasoning*, LNCS 624, pages 273–284. Springer, 1992.

[3] Leo Bachmair and Harald Ganzinger. Associative-commutative superposition. In *Proc. 4th Int. Workshop on Conditional and Typed Rewriting*, LNCS 968, pages 1–14, Jerusalem, 1994. Springer.

[4] Leo Bachmair and Harald Ganzinger. Resolution theorem proving. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, volume 1, chapter 2, pages 19–100. North Holland, 2001.

[5] Leo Bachmair, Harald Ganzinger, Christopher Lynch, and Wayne Snyder. Basic paramodulation. *Information and Computation*, 121(2):172–192, 1995.

[6] The CADE-18 ATP System Competition (CASC-18), 2002. `http://www.cs.miami.edu/~tptp/CASC/18/`.

[7] Anatoli Degtyarev and Andrei Voronkov. Stratified resolution. In *Proc. 17th Int. Conf. on Automated Deduction (CADE-17)*, LNAI 1831, pages 365–384, Pittsburgh, PA, USA, 2000. Springer.

[8] Gilles Dowek, Thérèse Hardin, and Claude Kirchner. Theorem proving modulo. Rapport de Recherche 3400, INRIA, 1998.

[9] Harald Ganzinger, Robert Nieuwenhuis, and Pilar Nivela. The Saturate system. System available at `http://www.mpi-sb.mpg.de/SATURATE/`, 1994.

[10] Gérard Huet. The zipper talk. Invited Lecture at FLoC'96, 1996.

[11] Z. Manna and R. Waldinger. A deductive approach to program synthesis. *ACM Transactions on Programming Languages and Systems*, 2(1), 1980.

[12] Neil V. Murray. Completely non-clausal theorem proving. *Artificial Intelligence*, 18(1):67–85, 1982.

[13] Robert Nieuwenhuis and Albert Rubio. Theorem proving with ordering and equality constrained clauses. *Journal of Symbolic Computation*, 19:321–351, 1995.

[14] Robert Nieuwenhuis and Albert Rubio. Paramodulation-based theorem proving. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, volume 1, chapter 7, pages 371–443. North Holland, 2001.

[15] A. Riazanov and Voronkov A. The design and implementation of Vampire. *AI Communications*, 15(2-3), 2002.

[16] Stephan Schulz. E-Setheo. `http://www4.informatik.tu-muenchen.de/~schulz/WORK/e-setheo.html`.

[17] Jürgen Stuber. Deriving theory superposition calculi from convergent term rewriting systems. In *Proc. 11th Int. Conf. on Rewriting Techniques and Applications (RTA 2000)*, volume 1833 of *LNCS*, pages 229–245, Norwich, UK, 2000. Springer.

[18] Jürgen Stuber. A model-based completeness proof of Extended Narrowing And Resolution. In *Proc. 1st Int. Joint Conf. on Automated Reasoning (IJCAR-2001)*, LNCS 2083, pages 195–210, Siena, Italy, 2001.

[19] G. S. Tseitin. On the complexity of derivation in propositional calculus, 1970. Reprinted in J. Siekmann and G. Wrightson (eds): Automation of Reasoning, Vol. 2, Springer, 1983, pp. 466–486.