MAX-PLANCK-INSTITUT FÜR INFORMATIK

Superposition with Simplification as a Decision Procedure for the Monadic Class with Equality

Leo Bachmair
Harald Ganzinger
Uwe Waldmann

MPI–I–93–204 February 1993

Im Stadtwald
W 6600 Saarbrücken
Germany

Authors' Addresses

Leo Bachmair
Department of Computer Science, SUNY at Stony Brook, Stony Brook, NY 11794, U.S.A, leo@cs.sunysb.edu
Harald Ganzinger, Uwe Waldmann
Max-Planck-Institut für Informatik, Im Stadtwald, D-W-6600 Saarbrücken, Germany, {hg,uwe}@mpi-sb.mpg.de

Publication Notes

This paper has been submitted for publication elsewhere and will be copyrighted if accepted.

Acknowledgements

The research described in this paper was supported in part by the German Science Foundation (Deutsche Forschungsgemeinschaft) under grant Ga 261/4-1, by the German Ministry for Research and Technology (Bundesministerium für Forschung und Technologie) under grant ITS 9102/ITS 9103 and by the ESPRIT Basic Research Working Group 6028 (Construction of Computational Logics). The responsibility for the contents of this publication lies with the authors.

Abstract

We show that strict superposition, a restricted form of paramodulation, can be combined with specifically designed simplification rules such that it becomes a decision procedure for the monadic class with equality. The completeness of the method follows from a general notion of redundancy for clauses and superposition inferences.

Keywords

Decidable First-Order Theories, Monadic Class with Equality, Superposition, Simplification, Paramodulation, Resolution

1 Introduction

The monadic class is a fragment of first-order logic the decidability of which, not only for the case with equality but also for quantification over predicates, is known since (Löwenheim 1915). A simpler proof has later been given by Ackermann (1954), among others. His syntactic method of transforming the given formula into some kind of solved form is surprisingly modern in style and may be usable in practice. Joyner Jr. (1976) has shown that ordered resolution, which is known to be refutationally complete for arbitrary first-order theories, can be specialized to yield a decision method for the monadic class *without* equality and without second-order quantifiers. (For an overview of more recent results along this line see Fermüller et al. 1992.) Resolution can be combined with an inference rule, called paramodulation, to yield a refutationally complete inference system for first-order clauses with equality. The purpose of this paper is to show that *superposition*, a restricted form of paramodulation, can be equipped with specific simplification techniques such that it becomes a decision procedure for the monadic class *with* equality. It should be emphasized that this generalization of inference system and logic is not at all a trivial exercise. In fact it turned out to be technically more involved than we expected.

From a theoretical point of view our result is of interest as it provides evidence of the usefulness of the superposition calculus as a general method of refutational theorem proving for first-order logic with equality. Based on earlier work (e.g., Bachmair and Ganzinger 1990) the superposition calculus as we use it here has been proposed and proven refutationally complete (for arbitrary first-order clauses) by Bachmair and Ganzinger (1991). It represents an improvement over the results obtained by Rusinowitch (1989) (also see Rusinowitch 1991) in two directions. The ordering restrictions are sharpened and the inference rules are additionally parameterized by *selection functions* so that a larger class of search strategies can be modelled. More importantly, the calculus provides a general notion of *redundancy* by which, at any state of the theorem proving process, inferences can be further restricted in terms of global properties of the current database of formulas. Simplification inferences which replace formulas by simpler ones do not affect refutational completeness whenever the simplified formula renders the original formula redundant. In this paper we demonstrate that with an appropriate setting of its three main parameters (ordering, selection function, simplification inferences), the superposition calculus can be tailored to the specifics of the problem at hand.

On the practical side this means that we can implement our decision method simply by taking an implementation of the superposition calculus such as the one described in (Nivela and Nieuwenhuis 1993) and add only what is needed to specify the parameter setting. Our main practical motivation for the work described here, however, stems from the observation that set constraints with subterm equality and the monadic class with equality are essentially the same logic (Bachmair, Ganzinger, and Waldmann 1992). Set constraints have attained quite some interest as a means of specifying type inference and flow analysis for programming languages (Heintze and Jaffar 1991, Aiken and Wimmers 1992). Applications of the monadic class to knowledge representation languages have been described by Fermüller et al. (1992).

Although there is quite some body of work on using resolution as a decision procedure only little is known at present about paramodulation in the case of equality. One

exception are the shallow equational theories as dealt with by Comon, Haberstrau, and Jouannaud (1992). In a certain sense, the present paper extends the results of these authors to the first-order case. It is, however, not possible to extend their result in full generality. First-order clauses over shallow equations, if no restrictions about variable occurrences apply, form a reduction class; any non-shallow equation can then be flattened with the help of auxiliary variables. While writing this paper we have seen the announcement of another related result by Fermüller and Salzer (1993) about deciding the Ackermann class with equality by ordered paramodulation.

2 The Monadic Class with Equality

First-order formulas are built from quantifiers, logical symbols, predicate symbols, function symbols, and variables. Constants are function symbols of arity zero. We shall consider only languages containing at least one constant.

The *monadic class with equality* is the class of first-order formulas that contain only unary (monadic) predicate symbols, with the exception of one binary predicate symbol denoting equality, and no function symbols.

If we *skolemize* a monadic formula in prenex form (and thereby introduce function symbols), the resulting quantifier-free formula can be characterized by the following syntactic properties: (i) all atoms are of the form $p(t)$ or $s \approx t$ and (ii) there exists a sequence $x_1, \ldots, x_m$ of distinct variables such that any term t is either a variable x_n, or of the form $f(x_1, \ldots, x_n)$, for some $n \leq m$.

For example, the monadic formula with equality

$$\exists a \forall x \exists f \forall y \exists g \ (p(x) \wedge q(y) \rightarrow a \approx g \vee f \approx y)$$

skolemizes into the formula

$$p(x) \wedge q(y) \rightarrow a \approx g(x, y) \vee f(x) \approx y.$$

Skolemization preserves the satisfiability of formulas. The resulting quantifier-free formulas can be further transformed to conjunctive normal form. We will represent them as sets of clauses.

Formally, a *clause* is a multiset of literals; a *literal* being either an atomic formula A or the negation thereof, $\neg A$. Clauses are usually written as disjunctions, e.g., $A \vee A \vee \neg B$ and $A \vee \neg B$ are two (different) clauses.

The class of clauses we shall consider is actually slightly larger than the one obtained from the monadic class. A clause C is said to be *flat* if (i) all its atoms are of the form $p(t)$ or $s \approx t$ and (ii) there exists a list of distinct variables $V = v_1, \ldots, v_k$ and a subset W of $\{v_1, \ldots, v_k\}$, such that each term in C which is not a variable is of the form $f(v_1, \ldots, v_m)\sigma$, where $0 \leq m \leq k$ and σ is a substitution that maps every variable in W to a constant and every other variable to itself. We say that x is an *extra variable* in a flat clause C, if x occurs in C, but not as an argument of a function symbol.

For example,

$$\neg p(x) \wedge q(y) \vee a \approx g(a, y) \vee f(b) \approx h(a, y, z)$$

is a flat clause (with $V = x', y, z$ and $W = \{x'\}$) and x is its only extra variable.

In a flat clause C, a variable that occurs as the i-th argument of a function symbol, will be the i-th argument of any other occurrence of a function symbol in C, the arity of which is at least i. The same variable can not occur in any other argument position of a function symbol in C, but may occur as argument of a predicate symbol. Thus, $p(f(b, y)) \vee q(g(x))$ is not a flat clause, since x occurs as first argument of g, but not of f.

Our decision procedure for flat clauses is based on the superposition calculus, which we describe next.

3 The Superposition Calculus

3.1 The Inference System

The superposition calculus is a refutationally complete inference system for arbitrary first-order clauses with equality. That is, it provides a semi-decision procedure for the unsatisfiability of sets of clauses.

Its inference rules are restricted versions of paramodulation, resolution, and factoring; parameterized by a total reduction ordering $\succ$ on ground expressions (that is, ground atoms and ground terms) and a *selection function* S, which assigns to each clause a (possibly empty) multiset of (occurrences of) *negative* literals. The literals in $S(C)$ are called *selected*. Selected literals, besides being negative, can be arbitrarily chosen. The calculus is described in full detail in Bachmair and Ganzinger (1991) and will not be repeated here.[1] The following example, of paramodulation into a negative literal, illustrates the main features of the inference rules.

$$\frac{\Gamma \vee s \approx t \quad \neg A[s'] \vee \Lambda}{\neg A[t]\sigma \vee \Gamma\sigma \vee \Lambda\sigma}$$

where σ is the most general unifier of s and s', and there exists a ground substitution τ such that for $\theta = \sigma\tau$ (i) $s\theta \approx t\theta$ is the strictly maximal literal in $\Gamma\theta \vee s\theta \approx t\theta$, (ii) $s\theta \succ t\theta$ and $s\theta$ does not occur in a negative literal of $\Gamma\theta$, (iii) $u\theta \succ v\theta$, if $A[s']$ is an equation $u[s'] \approx v$, (iv) either $\neg A\theta$ is a maximal (occurrence of a) literal in $\neg A\theta \vee \Lambda\theta$, or else $\neg A$ is a selected literal in $\neg A \vee \Lambda$, (v) $\Gamma \vee s \approx t$ has no selected literals, and (vi) s' is not a variable.

This inference rule combines the unification of s and the subterm s' of A (the "superposition" of s on A at s') with subsequent replacement of $s\sigma$ by $t\sigma$. We call the first premise the *active* and the second premise the *passive premise*, respectively, of the inference. When we speak of a *superposition inference* we mean an arbitrary rule of the calculus.[2]

The various restrictions imposed by the ordering and the selection function are crucial in the design of a decision procedure for flat clauses. In particular, note

[1] Strictly speaking, we use the calculus $\mathcal{E}_S$ of (Bachmair and Ganzinger 1991), which contains the equality factoring rule instead of the ordered factoring and merging paramodulation rules of the system $\mathcal{P}_S$. It assumes an extension of $\succ$ to ground literals such that, in particular, $L \succ L'$ whenever the maximal expression in L is greater than the maximal expression in L'.

[2] If an inference has only one premise, we consider this to be active in the case of a factoring inference, and passive, otherwise.

that clauses with selected literals cannot serve as active premises in superposition inferences, and that maximal terms are superposed either on maximal or selected literals of a passive premise, and a smaller term is never replaced by a larger one. Moreover, only the maximal sides of equations are replaced. The pattern of interplay between ordering restrictions and the selection function is the same for all inference rules of the calculus.

One difficulty that arises in applying superposition to flat clauses is that non-flat conclusions can be derived from flat premises. This problem can be addressed in part by the choice of ordering. Henceforth, we assume $\succ$ to be a lexicographic path ordering[3] based on a precedence relation that respects the arity of function symbols (greater arity implying higher precedence) and in which function symbols have higher precedence than predicate symbols. Apart from satisfying these restrictions, the precedence can be arbitrary (but has to be total). Some more subtle aspects of inferences, such as extra variables, that may also result in non-flat clauses, cannot be captured with orderings, though.

A clause C is called (uniformly) *reductive* (with respect to e) if e is an expression (that is, a term or a non-equational atom) in C such that for any ground substitution σ, $e\sigma$ is the maximal expression in $C\sigma$. For instance, flat clauses with no extra variables are reductive, as are flat clauses that contain only one variable and no function symbols. An example of the latter case is the clause $p(x) \vee q(x)$. In general, we have:

Lemma 1. *If C is flat and reductive with respect to e, then (i) e contains all variables which occur in C, (ii) if e is of the form $p(x)$, with predicate p and variable x, C does not contain any non-constant function symbols, and (iii) if C contains a non-constant function symbol, then e contains a function symbol with maximal arity in C.*

If a clause C is non-reductive, then it can either be *partitioned*, that is, written as $C'(X) \vee C''(Y)$ where the two subclauses C' and C'' share no variables; or contains an equation $x \approx x$; or contains a *connecting variable*, that is, an extra variable x and an equation (also called a *variable connection*) $x \approx t$ with t not containing x. For example, in $p(x) \vee x \approx y \vee q(y)$, the variable x connects the two variable-disjoint subclauses $p(x)$ and $q(y)$.

We call a superposition inference from flat clauses *flatness preserving* if its active premise is reductive and if its passive premise is either reductive and has no selected literal, or else contains one selected negative equation with a connecting variable. This terminology is justified by the following lemma:

Lemma 2. *The conclusion of any flatness preserving inference from flat clauses is again a flat clause.*

Proof. The proof makes essential use of the assumptions about $\succ$. We will consider the cases of paramodulation into a negative literal and of ordered resolution, the proofs for the other inference rules being similar. Suppose that the active premise $C \vee s \approx t$ of a paramodulation inference is flat and reductive with respect to s. Then

[3]Details about the lexicographic path ordering can be found in survey papers on rewriting such as (Dershowitz and Jouannaud 1990).

s contains all variables occurring in $C \vee s \approx t$ and is not a variable itself, in other words, s is a term $f(v_1, \ldots, v_m)\sigma$ where σ maps the variables in $W \subseteq \{v_1, \ldots, v_m\}$ to constants symbols and all other variables to themselves. The substitution ρ unifies s with a term s' in the passive premise $\neg A[s'] \vee \Lambda$. Every term in $\neg A[s'] \vee \Lambda$ that is not a variable is of the form $g(v'_1, \ldots, v'_j)\sigma'$, $0 \leq j \leq k$, where σ' maps the variables in $W' \subseteq \{v'_1, \ldots, v'_k\}$ to constants symbols and all other variables to themselves. As s' is not a variable, we have $s' = f(v'_1, \ldots, v'_m)\sigma'$. Without loss of generality, we may assume that ρ maps every variable v_i or v'_i either to a constant or to the (new) variable v''_i. Let W'' be the set $\{\, v''_i \mid v_i \in W \vee v'_i \in W' \,\}$. Then every term of the conclusion that is not a variable is of the form $g(v''_1, \ldots, v''_j)\theta$ where θ maps every variable in W'' to a constant and every other variable to itself.

In the case of an ordered resolution inference of the assumed kind the passive premise cannot have a selected literal. Hence the passive premise is a reductive clause. For both the active and the passive premise, we have to consider two cases: either the clause contains only one variable x and consists entirely of literals $p(x)$, or the maximal literal has the form $p(f(v_1, \ldots, v_m)\sigma)$, and f is a function symbol with maximal arity in the clause. In any of these cases the proof proceeds in a similar way as above.

The flatness preserving inferences will form the core of our decision procedure. Superposition inferences that do not preserve flatness can be shown to be unnecessary or redundant, in a sense made precise next.

3.2 Redundancy

The superposition calculus does not directly yield a decision procedure, but it is compatible with a rather general notion of redundancy (Bachmair and Ganzinger 1991), which we shall exploit in developing a decision procedure.

Redundancy is based on an ordering $\succ^{\mathrm{i}}$ of pairs (C, σ) of clauses C and ground substitutions σ such that $(C, \sigma) \succ^{\mathrm{i}} (D, \tau)$ if and only if either (i) $C\sigma \succ D\tau$ or else (ii) $C\sigma = D\tau$, C and D are flat clauses, and D is an instance of C, but not vice versa. In other words, if the two instances are the same, the pair in which the clause is more (!) instantiated is considered smaller. As this (partial) ordering is applied only to flat clauses, which are of bounded depth, it is well-founded.

If N is a set of clauses and C is a clause, then C is called *redundant in* N if for all ground substitutions σ there exist $n \geq 0$ clauses C_j in N and ground substitutions σ_j such that (i) $C_1\sigma_1, \ldots, C_n\sigma_n \models C\sigma$, and (ii) $(C, \sigma) \succ^{\mathrm{i}} (C_j, \sigma_j)$, for $1 \leq j \leq n$. A set N of clauses is called *saturated (up to redundancy)* if all clauses that can be derived (by an inference of the superposition calculus) from non-redundant premises in N are either contained in N or else are redundant in N.

The following theorem from (Bachmair and Ganzinger 1991), which holds for arbitrary reduction orderings and selection functions, plays a central role for the correctness of our decision procedure:[4]

[4] Actually in that paper we use a slightly less powerful ordering $\succ^{\mathrm{i}}$. But the paper also discusses why this does not make a difference. Also see (Bachmair and Ganzinger 1990) for an ordering $\succ^{\mathrm{i}}$ in which the subsumption ordering is employed in the usual way. The extension of $\succ$ to ground clauses is given by the multiset extension of the ordering on literals.

Theorem 3. *Let N be saturated up to redundancy. Then N is satisfiable if and only if it does not contain the empty clause.*

This notion of redundancy leaves us a sufficient amount of freedom to design simplification and deletion rules which will guarantee that finite sets of flat clauses can always be finitely saturated. In the following section we shall first describe the general scheme by which saturation proceeds, the fair theorem proving derivations.

3.3 Theorem Proving Derivations

A (clausal) theorem proving derivation from a set of clauses N is a finitely branching (possibly infinite) tree δ, the nodes of which are sets of clauses, such that N is the root of δ and for each non-leaf node N_0 in δ with children $N_1, \ldots, N_k$, either (i) $k = 1$ and $N_1 = N_0 \setminus \{C\}$, where C is a clause in N_0 which is redundant in N_1, or (ii) $k \geq 1$ and $N_i = N_0 \cup M_i$, $1 \leq i \leq k$, with nonempty sets M_i of clauses such that if N_0 is satisfiable then at least one of the $N_0 \cup M_i$ is satisfiable, too. In case (i) a redundant formula is *deleted*, while in case (ii) some formula is *added* and a *case analysis* is possibly done.

A theorem proving derivation from N is called *fair* if for any path $N = N_0, N_1, \ldots$ in the tree, with *limit* $N_\infty = \bigcup_j \bigcap_{k \geq j} N_k$, it is the case that each clause C that can be deduced by a superposition inference from non-redundant premises in N_∞ is contained in some set N_j.

Lemma 4. *Let δ be a fair theorem proving derivation from N. If $N, N_1, N_2, \ldots$ is a path in δ with limit N_∞ then N_∞ is saturated. Furthermore, N is satisfiable if and only if there exists a path in δ the limit N_∞ of which is satisfiable.*

Proof. If C is a clause in $(\bigcup_j N_j) \setminus N_\infty$ then it is redundant with respect to some set N_j and hence, redundant with respect to $\bigcup_j N_j$. From this we may moreover infer that all clauses in $(\bigcup_j N_j) \setminus N_\infty$ are redundant with respect to N_∞. Therefore, if C is the conclusion of an inference from non-redundant clauses in N_∞ and is not in N_∞, it must be redundant with respect to N_∞, by virtue of the fact that fairness implies that C is a clause in $\bigcup_j N_j$.

N is satisfiable if and only there exists a path $N = N_0, N_1, \ldots$ in δ such that each of the N_i is satisfiable. By compactness of first-order logic this is equivalent to the satisfiability of N_∞.

Corollary 5. *Let δ be a fair saturation proving derivation from N. Then N is satisfiable if and only there exists a path in δ the limit of which does not contain the empty clause.*

4 The Decision Procedure

We now describe how to construct a fair and finite derivation from any given finite set of flat clauses N_0 over some some finite vocabulary Σ. We assume that Σ contains some sufficiently large subset $\mathcal{C}$ of additional constant symbols which do not occur in

N_0. The relevance of these constants and what we mean by "sufficiently large" will become clear below. We design a calculus of *expansion rules* of the form

$$\frac{N}{N_1 \mid \cdots \mid N_k}$$

with each representing a finite derivation with root N and leaves $N_1, \ldots, N_k$. The expansion rules indicate how derivations may be expanded at leaves (by adding new subtrees).

For instance, flatness preserving superposition inferences are described by the following expansion rule:

Deduce

$$\frac{N}{N \cup \{C\}}$$

if C is the conclusion of a flatness preserving superposition inference from clauses in N.

A simple instance of deletion of redundant formulas is given by:

Delete

$$\frac{N \cup \{C\}}{N}$$

if (i) C is a tautology or (ii) N contains a clause C', where C' can be obtained from C by renaming variables. In the following, clauses that can be deleted in this way will be called *trivial* in N.

The third possibility, case split, allows us to partition clauses:

Partition

$$\frac{N \cup \{C \vee C'\}}{N \cup \{C\} \mid N \cup \{C'\}}$$

if C and C' do not share any variables.

The above expansion rules are clearly insufficient for constructing fair derivations. Fairness requires that *all* superposition inferences be applied to non-redundant clauses, yet "Deduce" covers only flatness-preserving inferences. We will therefore design further expansion rules, called *simplification rules*, that can be used to eliminate problematic formulas. These expansion rules represent derivations consisting of an addition—a node $N \cup \{C\}$ with a child $N \cup \{C, D\}$—followed by a corresponding deletion—a node $N \cup \{D\}$ inserted below $N \cup \{C, D\}$). In other words, the key is to find a suitable formula D in the presence of which an undesirable formula C is rendered redundant. In some cases, more than one formula needs to be added before a problematic formula becomes redundant.

Factor 1

$$\frac{N \cup \{L \vee L \vee C\}}{N \cup \{L \vee C\}}$$

Factor 2

$$\frac{N \cup \{\neg x \approx y \vee C\}}{N \cup \{C[y/x]\}}$$

where x is an extra variable in C.

Factor 3

$$\frac{N \cup \{\neg x \approx t \vee \neg y \approx t \vee C\}}{N \cup \{\neg y \approx t \vee C[y/x]\}}$$

where x is an extra variable in C.[5]

Before stating the remaining rules we introduce some constraint notation to describe certain sets of instances of a clause. By $c_1, \ldots, c_k :: C$, where the c_i are constraints of the form $t_i \prec a_i$ or $t_i \preceq a_i$, with terms t_i and constants a_i, and where C is a flat clause, we denote the family $C\sigma$ of clauses where σ ranges over the solutions of the constraints. By that we mean ground substitutions for the variables in t_i for which the constraints are satisfied with respect to the given reduction ordering.

A second kind of constrained formulas which we will use is of the form $x \succ a :: C$, where x is a variable, a a constant and C a flat clause containing no function symbols other than constants. The latter expression will be a shorthand for the family of clauses $C[f(x_1, \ldots, x_n)/x]$, where f ranges over constants and function symbols greater than a, and where the x_i are distinct variables not occurring in C. Note that since constants are smaller than any non-constant term, and since we consider finite vocabularies only, the set of clauses represented by a constrained clause is always finite and consists of flat clauses only. Hence, the constraint notation does not introduce any new concept.

Instantiate

$$\frac{N \cup \{C \vee x \approx a_1 \vee \ldots \vee x \approx a_n\}}{N \cup \{x \succ a_1 :: C \vee x \approx a_1 \vee \ldots \vee x \approx a_n,\ x \prec a_1 :: C \vee x \approx a_1 \vee \ldots \vee x \approx a_n\}}$$

if x is a variable, the a_i are constants, with a_1 the maximal one, and C contains only non-equational atoms of the form $p(x)$.

This last rule indicates how certain non-reductive flat clauses may be rendered redundant by adding suitable instances.

The final rule of our calculus, to be described next, covers more complicated flat clauses with connecting equations to which the previous rule does not apply. It is designed to remove the connections between the variable-disjoint subclauses C' and C'' of C, which is achieved using constants to split connecting equations. The full details of this process are somewhat technically involved as we have to deal with disjunctions of connecting equations, and as we have to make sure that the new clauses are smaller than the clause that is split.

[5]Note that replacing a clause $\neg x \approx t \vee C$, where x does not occur in t, by $C[t/x]$ is in general not an admissible simplification. In cases where x is instantiated with a term smaller than t, the corresponding instance of the second clause can be too large. These negative literals have to be "eliminated" by selecting them for superposition. Our more specialized "Factor" rules, however, are simplifications.

Restrict

$$\frac{N \cup \{C\}}{N \cup \{C'\} \mid N \cup \{C''\} \mid N \cup M}$$

if C is a non-trivial clause of the form

$$C' \vee C'' \vee x \approx t_1 \vee \ldots \vee x \approx t_n$$

such that (i) the variable x does not occur in C'' nor in the t_i, (ii) C' consists of pairwise distinct occurrences of non-equational atoms of the form $p(x)$, and (iii) at least one of the terms t_i is not a constant. Then M is the set of clauses consisting of

$$\begin{aligned}
x \succ a_1 \quad &:: \quad C' \vee x \approx a_1 \vee \ldots \vee x \approx a_k, && (1)\\
&\qquad C'' \vee a_i \approx t_1 \vee \ldots \vee a_i \approx t_n;\ 1 \leq i \leq k && (2)\\
x \prec a_1 \quad &:: \quad C' \vee C'' \vee x \approx t_1 \vee \ldots \vee x \approx t_n, && (3)\\
t_1 \preceq a_1, \ldots, t_n \preceq a_1 \quad &:: \quad C' \vee C'' \vee x \approx t_1 \vee \ldots \vee x \approx t_n && (4)
\end{aligned}$$

The a_i are constants with the following properties. If N contains (up to renaming of the variable x) a clause of the form $x \succ b_1 :: C' \vee x \approx b_1 \vee \ldots \vee x \approx b_l$, with constants b_i, then $k = l$ and $a_i = b_i$. Otherwise, $k = n$, and the a_i are taken to be constant symbols in $\mathcal{C}$ that are new, that is, do not occur on the path from the root of the derivation tree to $N \cup \{C\}$. We assume that a_1 is maximal among the a_i with respect to $\succ$.

With a finite supply of predicate symbols only finitely many clauses of the form required for C' can be constructed. Also, clauses of the form (1), once added, cannot be deleted in later expansion steps. Hence, in fact, a finite supply of constants $\mathcal{C}$ will suffice to allow "Restrict" be applied whenever needed. The practical problem, however, is that we do not know in advance how many constants we have to generate. Therefore the concept of constrained formulas not only provides a convenient notation for specifying certain sets of instances of formulas, it also suggests how to treat the unknown $\mathcal{C}$ lazily in an actual implementation of the expansion rules.[6] Whenever an application of "Restrict" produces new constants, any constrained formula that has been added in previous expansions further up in the derivation tree simply needs to be inspected again as to whether the enriched signature (with an enriched precedence on symbols) gives rise to additional solutions of the constraint. The additional instances can then be added. This handling of constraints is explicitly supported within the extension of the superposition calculus by ordering constraints (Nieuwenhuis and Rubio 1992).

These remarks on implementation issues (which are of no theoretical significance) complete the description of our calculus of expansion rules. In most cases it is fairly easy to determine which derivation is represented by an expansion rule. The most complicated rule is "Restrict," which represents a derivation with root $N \cup \{C\}$ and

[6]There is no point in fixing $\mathcal{C}$ in advance by exploiting theoretically known upper bounds for the cardinality of finite models for monadic formulas. Then nothing would be gained over trivial methods which simply enumerate all structures up to this size. Although these methods are in some theoretical sense optimal (Lewis 1980), they appear to be competely useless in practice.

three children $N \cup \{C, C'\}$, $N \cup \{C, C''\}$, and $N \cup \{C\} \cup M$, each of which in turn has one child, obtained by deleting C. The following two lemmas establish that this is indeed a derivation.

Lemma 6. *The clause C is redundant in, respectively, $N \cup \{C'\}$, $N \cup \{C''\}$, and $N \cup M$.*

Proof. While the first two cases are obvious, for the third case we have to show that each ground instance $C\sigma$ of C which is not covered by the two constrained clauses (3) and (4) in M follows from smaller ground instances of $N \cup M$. (Note that as the clauses represented by (3) and (4) are more instantiated than C, their ground instances are smaller than corresponding instances of C with respect to $\succ^{\mathrm{i}}$.) Let us, therefore, assume that $x\sigma \succ a_1$ and $t_j\sigma \succ a_1$ for some j. (Instances for which $x\sigma = a_1$ are tautologies, hence redundant.) Then

$$C\sigma \succ (C' \vee x \approx a_1 \vee \ldots \vee x \approx a_k)\sigma$$

and

$$C\sigma \succ (C'' \vee a_i \approx t_1 \vee \ldots \vee a_i \approx t_n)\sigma,$$

for $1 \leq i \leq k$. Moreover, suppose that I is an interpretation in which the clauses in M are satisfied. We have to show that $C\sigma$ is also satisfied in I. If $C'\sigma$ and $C''\sigma$ are false in I, then, from the validity of the instances of the clause (1) we may infer the existence of an i such that $x\sigma \approx a_i$ is true in I. Similarly, as the instances of the clauses (2) are satisfied in I, there exists a j_i such that $a_i \approx t_{j_i}\sigma$ is true in I. Altogether, $x\sigma \approx t_{j_i}\sigma$, and hence $C\sigma$, are true in I.

Lemma 7. *If $N \cup \{C\}$ is satisfiable, then one of the three sets $N \cup \{C, C'\}$, $N \cup \{C, C''\}$, or $N \cup \{C\} \cup M$ is also satisfiable.*

Proof. Let I be a model of $N \cup \{C\}$. If I is a model of neither C' nor C'', then there exist variable assignments σ and τ for the variables in C' and C'', respectively, such that both $C'\sigma$ and $C''\tau$ are false in I. Hence $C' \vee x \approx t_1\tau \vee \ldots \vee x \approx t_n\tau$ is valid in I, for all assignments to x, from which we may infer that there are at most n different values of x such that C' is false in I. If N does not contain a (possibly constrained) clause of the form $C' \vee x \approx b_1 \vee \ldots \vee x \approx b_l$, then the a_i are new constants which do not occur in $N \cup \{C\}$. Let then I' be the same as I, except that the interpretation of the a_i is changed into what is obtained by evaluating $t_i\tau$ in I. As the a_i do not occur in $N \cup \{C\}$ and as the domains of I and I' are identical, I' satisfies $N \cup \{C\}$. If the a_i are "old" constants, let I' be the same as I. By construction, I' satisfies $C' \vee x \approx a_1 \vee \ldots \vee x \approx a_k$. As the set of values for which C' is false in I' is exactly the set of values represented by the a_i, and as C is true in I', $C'' \vee a_i \approx t_1 \vee \ldots \vee a_i \approx t_n$ is satisfied in I', for any $1 \leq i \leq k$. Hence we have shown that if I is not a model of the first two alternatives, I' is a model of $N \cup \{C\} \cup M$.

In sum, we have the following result:

Theorem 8. *Any tree generated by any sequence of applications of expansion rules is a theorem proving derivation.*

From now on we shall restrict attention to theorem proving derivations which are generated by strategies in which "Delete", "Partition", "Factor", "Instantiate", "Restrict", and "Deduce" are applied in this order of (descending) priorities. Moreover we assume that no "Deduce" inference is computed twice on the same path in the tree. Theorem proving derivations of this form will be called *expansions*.

Lemma 9. *Let N be a finite set of flat clauses and consider an arbitrary expansion δ from N. Then δ is finite.*

Proof. Applying Lemma 2, we may infer that "Deduce" does not add any non-flat clause. The same property holds for the other expansion rules. Note that many of the restrictions to the expansion rules are meant to preserve the variable property of flat clauses. What remains to be shown is that formulas cannot become arbitrarily long. As simplification expansions such as "Partition" and "Factor 1" are applied eagerly, the only remaining problems are variable connections.

We have to show that the number of variable connections in clauses is bounded. We first observe that there cannot be an unbounded number of negative variable connections $x \approx t$ in a clause if the factoring rules are applied eagerly. This is due to the fact that if t is a non-variable term, its variables have to satisfy the variable restrictions for flat clauses. Therefore the number of different non-variable terms, and hence the number of different negative variable connections in a clause is bounded. We may, therefore, infer the same upper bound for the number of different variables in negative literals of a clause. Now if a (non-trivial) and fully factored clause contains more positive variable connections $x \approx t$ than can be expressed by the variables of the negative literals and the finite number of function symbols, in at least one of them the variable x does not appear in a negative equation. In this case the "Restrict" rule can be applied by which this positive variable connection is eliminated.

Lemma 10. *Given a finite set of flat clauses N there exists a fair theorem proving derivation from N.*

Proof. We show that there exists an expansion from N which is fair. For that purpose we have to argue that superposition inferences which cannot be computed by "Deduce" are redundant. "Deduce" makes the restriction that the active clause has to be reductive and that the passive clause is either reductive and has no selected literal, or else has a selected negative variable connection. The restrictions about selection can be obeyed by a strategy such that whenever a clause contains a negative variable connection, in which case it cannot be reductive, one such equation is selected. No other literals in clauses are selected. As a further consequence of this strategy, clauses with negative variable connections cannot not be active. It remains to be shown that all other non-reductive clauses can be simplified. A non-reductive clause either has more than one partition, in which case it can be split by "Partition", or else it contains an extra variable x and a variable connection $x \approx t$. As negative connections are dealt with by the selection strategy we are left with the case where all variable connections are positive. If $t = x$ we have a tautology which can be removed by "Delete". If t is not a constant, we may apply "Restrict" to split the connection. If there exists only one partition and if all variable connections have constants on their right sides, then the clause is dealt with by "Instantiate".

Theorem 11. *Superposition with simplification is a decision procedure for the monadic class with equality.*

Proof. Applying the superposition and expansion strategy as indicated by the previous lemmas yields a decision procedure for the monadic class with equality. A given formula is unsatisfiable if and only if all leaves of the tree constructed from the skolemized clausal normal form of the given formula contain the empty clause.

5 Conclusion

We have shown that the three parameters of the superposition calculus — ordering, selection function, and simplification inferences — can be set in a way such that saturation of a finite set of flat clauses is guaranteed to terminate and hence forms a decision procedure for the monadic class with equality. Although the technical details of the above proof have turned out to be more complex than we expected, being able to describe the full proof in a paper of this size we consider an achievement. It was made possible only by the fact that the criteria for what admissible settings the three parameters are allowed to have, can be defined in a self-contained and precise way without any reference to the (non-trivial) completeness proof for the superposition calculus itself

From this exercise we have learned several things. Being able to combine ordering constraints with unrestricted selection of negative literals is extremely useful in practice. In particular we do not see any other way of dealing with negative connecting equations in flat clauses. We have also seen that our notion of redundancy is sufficiently general to prove the admissibility of the required simplification inferences. This was made possible in some cases only through the ability to compare non-ground clauses with respect to a more unusual well-founded ordering, whenever certain ground instances are identical. Our previous papers have not explicitly pointed out the admissibility of such a practice. Finally we have used a constraint notation for certain instances of clauses. Although not a new concept theoretically, any practical implementation should be able to handle such constraints specifically. Nieuwenhuis and Rubio (1992) have shown how to extend or calculus to formulas with ordering constraints. For the present application an explicit treatment of constraints is particularly useful because of the dynamic creation of new constants by the "Restrict" rule.

What remains open at present is the computational complexity of our decision procedure. It is known that the problem itself is complete for NEXPTIME. More precisely, Lewis (1980) has shown that NTIME($c^{n/\log n}$) (for different constants c) is both a lower and an upper bound for the complexity of Monadic-Sat, which is the case without equality. In the case with equality, the best known upper bound is slightly higher but again in NEXPTIME (cf. e.g. Dreben and Goldfarb 1979). We conjecture that our procedure is no more complex than that.

References

W. Ackermann, 1954. *Solvable Cases of the Decision Problem.* North-Holland, Amsterdam.

A. Aiken and E.L. Wimmers, 1992. Solving Systems of Set Constraints (Extended Abstract). In *Proc. IEEE Symposium on Logic in Computer Science*, pp. 329–340.

L. Bachmair and H. Ganzinger, 1990. On Restrictions of Ordered Paramodulation with Simplification. In M. Stickel, editor, *Proc. 10th Int. Conf. on Automated Deduction, Kaiserslautern*, Lecture Notes in Computer Science, vol. 449, pp. 427–441, Berlin, Springer-Verlag.

L. Bachmair and H. Ganzinger, 1991. Rewrite-based equational theorem proving with selection and simplification. Technical Report MPI-I-91-208, Max-Planck-Institut für Informatik, Saarbrücken. Revised version to appear in the Journal of Logic and Computation.

L. Bachmair, H. Ganzinger and U. Waldmann, 1992. Set Constraints are the Monadic Class. Technical Report MPI-I-92-240, Max-Planck-Institut für Informatik, Saarbrücken.

H. Comon, M. Haberstrau and J.-P. Jouannaud, 1992. Decidable Problems in Equational Theories. In *Proc. IEEE Symposium on Logic in Computer Science*, pp. 255–265.

N. Dershowitz and J.-P. Jouannaud, 1990. Rewrite Systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science B: Formal Methods and Semantics*, chapter 6, pp. 243–309. North-Holland, Amsterdam.

B. Dreben and W.D. Goldfarb, 1979. *The Decision Problem. Solvable Classes of Quantificational Formulas.* Addison-Wesley Publishing Company, Inc.

C. Fermüller, A. Leitsch, T. Tammet and N. Zamov, 1992. *Resolution Methods for the Decision Problem.* To appear.

C. Fermüller and G. Salzer, 1993. Ordered Paramodulation and Resolution as Decision Procedure. Submitted.

N. Heintze and J. Jaffar, 1991. Set-based program analysis. Draft manuscript.

W.H. Joyner Jr., 1976. Resolution Strategies as Decision Procedures. *Journal of the Association for Computing Machinery*, Vol. 23, No. 3, pp. 398–417.

H.R. Lewis, 1980. Complexity Results for Classes of Quantificational Formulas. *Journal of Computer and System Sciences*, Vol. 21, pp. 317–353.

L. Löwenheim, 1915. Über Möglichkeiten im Relativkalkül. *Math. Annalen*, Vol. 76, pp. 228–251.

R. Nieuwenhuis and A. Rubio, 1992. Theorem proving with ordering constrained clauses. In *Automated Deduction — CADE'11*, Lecture Notes in Computer Science, vol. 607, pp. 477–491, Berlin, Springer-Verlag.

P. Nivela and R. Nieuwenhuis, 1993. Practical results on saturation of full first-order clauses: Experiments with the Saturate system. Submitted.

M. Rusinowitch, 1989. *Démonstration Automatique: Techniques de Réécriture.* Intereditions, Paris, France.

M. Rusinowitch, 1991. Theorem proving with resolution and superposition: An extension of the Knuth and Bendix completion procedure as a complete set of inference rules. *J. Symbolic Computation*, Vol. 11.