J Autom Reasoning (2007) 38:127–153
DOI 10.1007/s10817-006-9050-1

Superposition-based Equality Handling for Analytic Tableaux

Martin Giese

Received: 1 January 2006 / Accepted: 1 July 2006 /
Published online: 2 December 2006

Abstract We present a variant of the basic ordered superposition rules to handle equality in an analytic free-variable tableau calculus. We prove completeness of this calculus by an adaptation of the *model generation* technique commonly used for completeness proofs of superposition in the context of resolution calculi. The calculi and the completeness proof are compared to earlier results of Degtyarev and Voronkov. Some variations and refinements are discussed.

Key words superposition rules • equality handling • analytic tableaux

1 Introduction

Efficient equality handling for first-order tableaux or related calculi, such as matings or the connection method, has been problematic for a long time.

It is not strictly necessary to include a special treatment for the equality predicate in the definition of the semantics of a logic, nor is it theoretically necessary to employ any special treatment of equality in a calculus. Given a set of formulas in first-order logic with equality, one can add an axiomatization of equality to the problem, so that the combined set of formulas is satisfiable with respect to first-order logic *without* equality, exactly if the original set was satisfiable with respect to the logic with equality.

M. Giese (✉)
Johann Radon Institute for Computational and Applied Mathematics,
Austrian Academy of Sciences, Altenbergerstr. 69, A-4040 Linz, Austria
e-mail: martin.giese@oeaw.ac.at

This approach is completely unpractical, however. It is well-known that only very small problems can be solved with the axiomatic approach. For larger problems, the equality axioms allow so many redundant inferences that no theorem prover can cope with them. This is equally true for resolution provers as for tableaux provers.

Basically two ways exist for improving on the axiomatic approach. One way is to *transform* problems into a form that makes the equality axioms superfluous. The first such transformation was proposed by [13]; more recent developments are described by [37] or [6]. The other way to treat equality is to actually build it into the calculus and the proof procedure. This is the approach we investigate here.

The simplest way of building in equality is with *paramodulation* rules, that is, rules based on the replacement of equals, as one would do in a mathematical proof. Given a formula $\phi(s)$ containing an occurrence of a term s, and an equation $s \doteq t$, we can derive $\phi(t)$, which has the occurrence of s replaced by t. An integration of paramodulation into the resolution calculus was proposed by [45]. For sequent-based calculi, work goes back even earlier; see, for example, [30]. A simple paramodulation rule is also used to build equality into the free-variable tableaux described in Fitting's textbook [19].

Unfortunately, unrestricted paramodulation can still lead to a lot of redundant derivations, as equations can be applied in both directions. The key to reducing this redundancy is to use *term orderings* and to require that equations be applied only in a way that makes terms and formulas smaller with respect to such an ordering. The most prominent application of term orderings for equality reasoning is, of course, the Knuth–Bendix completion procedure [32]. A further benefit of using orderings is that they pave the way for the incorporation of redundancy criteria in the sense of [4], which are another successful ingredient of modern resolution-based theorem provers. A good overview of the state of the art in ordered paramodulation from a resolution perspective has been given by [41]. An overview of previous equality handling methods for tableaux and other sequent-based calculi has been compiled by [18].

The topic of this paper is a method for equality handling in free-variable tableaux that is based on ordered paramodulation. After giving an overview of the history of ordering-based equality handling for tableaux in Section 2, we define some basic terms in Section 3. In Section 4, we present the calculus. In Section 5 we prove completeness of that calculus, the main result of this article. This is followed in Sections 6, 7, and 8 by a discussion of several variations and refinements. An overview of related work is given in Section 9. The article concludes in Section 10 by identifying some possible areas for future work.

2 Ordering-based Equality Handling in Tableaux

It is generally believed that only techniques based on ordered rewriting can sufficiently reduce the search space of paramodulation-based equality reasoning to make it tractable. In the context of tableaux, it was also believed for a long time that the best approach to the integration of free variable tableaux and equality handling would be to search for *simultaneous rigid E-unifiers* [20] of disequations on the tableau and use these to close branches instead of usual unifiers. The overall idea was to solve the rigid E-unification problems by using ordered rewriting techniques.

Example 1 This example is taken from [16]. We construct a tableau from the following formula.

$$\begin{aligned}\forall x, y, u, z.(\quad & (a \doteq b \wedge \neg g(x, u, v) \doteq g(y, fc, fd)) \\ \vee \; & (c \doteq d \wedge \neg g(u, x, y) \doteq g(v, fa, fb)) \;)\end{aligned}$$

We use $\doteq$ to denote the equality predicate to distinguish it from the equality $=$ of the mathematical meta-level. After applying four γ-, one β-, and two α-expansions, we obtain two branches containing the following literals.

$$\begin{aligned}&\{a \doteq b\,,\; \neg g(X, U, V) \doteq g(Y, fc, fd)\} \\ &\{c \doteq d,\; \neg g(U, X, Y) \doteq g(V, fa, fb)\}\end{aligned}$$

Closing *one* of these branches constitutes what is known as a *rigid E-unification* problem. The task is to find instantiations for the rigid variables X, Y, U, V, such that the two sides of the negated equation are equal with respect to the equational theory $E = \{a \doteq b\}$, resp. $E = \{c \doteq d\}$, defined by the positive equational literals on the branch. In general, these equations might also contain free variables; and since the variables are rigid, all occurrences of these variables must be instantiated to the same terms.

This situation is in contrast to (nonrigid) *E-unification*, where the variables in the equational theory are understood to be universally quantified, and equations might therefore be applied several times with different instantiations.

One gets a *simultaneous rigid E-unification* problem, if one tries to solve several rigid E-unification problems with a *single* instantiation. For instance, the simultaneous rigid E-unification problem produced by the two branches above has the following solution:

$$\left[X/fa,\; Y/fb,\; U/fc,\; V/fd\right].$$

The inequation in the first branch becomes

$$\neg g(fa, fc, fd) \doteq g(fb, fc, fd)$$

under that instantiation, so equality of the two sides does indeed follow from the equation $a \doteq b$, and similarly for the second branch.

Unfortunately, simultaneous rigid E-unification was later shown to be undecidable by [15].[1] The outlined plan could thus be implemented only by using incomplete procedures for E-unification. In particular, procedures were used that produce complete sets of solutions to rigid E-unification problems for each branch and try to join these solutions to obtain a solution for the simultaneous problem, as, for instance, in the work of [11]. Experiments with such a setting showed that the *combination* of a first-order theorem prover and an incomplete solver for the branch-wise rigid E-unification problems seemed to be complete despite the incompleteness of the unification machinery, though nobody knew exactly why. Even if the needed simultaneous unifier was not found at a given point in the proof construction, some

[1]This result invalidated a number of prior attempts at a completeness proof that were based on the opposite assumption.

simultaneous unifier would always be found after the tableau was expanded a little more [9, 11, 42].

We will try to give an intuition of this effect without actually attempting to present a procedure that solves rigid E-unification problems. We do assume the reader to be familiar with rewriting techniques in general, however.

Example 2 For the two branches of the previous example, an E-unification procedure based on ordered rewriting would fail to find a simultaneous unifier. For the first branch, it would find a unifier

$$[X/Y,\ U/fc,\ V/fd],$$

but that will not lead to a closure of the second branch. There is no incentive for instantiation of X and Y to fa and fb if the other branch is not taken into account, and it is desirable to handle branches as independently as possible.

However, if the tableau is further expanded by repeating the same steps as before on each of the two branches, the situation changes. If we consider only literals, we now get a tableau with four branches.

$$\{a \doteq b, \neg g(X, U, V) \doteq g(Y, fc, fd),\ \neg g(X_1, U_1, V_1) \doteq g(Y_1, fc, fd)\}$$
$$\{a \doteq b,\ c \doteq d,\ \neg g(X, U, V) \doteq g(Y, fc, fd),\ \neg g(U_1, X_1, Y_1) \doteq g(V_1, fa, fb)\}$$
$$\{c \doteq d,\ a \doteq b,\ \neg g(U, X, Y) \doteq g(V, fa, fb),\ \neg g(X_2, U_2, V_2) \doteq g(Y_2, fc, fd)\}$$
$$\{c \doteq d, \neg g(U, X, Y) \doteq g(V, fa, fb),\ \neg g(U_2, X_2, Y_2) \doteq g(V_2, fa, fb)\}$$

The structure of the tableau, showing only the literals at the nodes where they are introduced, is given in Figure 1. Each branch now contains two negated equations. To close the tableau, the prover has to choose one of these from each branch and then find a simultaneous solution for the four rigid E-unification problems. It turns out that the right choice is to take the second negated equation for the leftmost and rightmost branch, and the first for the two middle branches. Let us consider the corresponding rigid E-unification problems from left to right. First, we have

$$\{a \doteq b,\ \neg g(X_1, U_1, V_1) \doteq g(Y_1, fc, fd)\}.$$

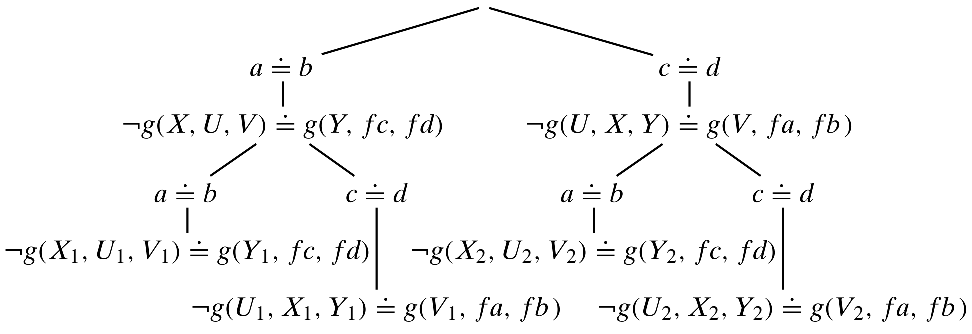
$a \doteq b$
$c \doteq d$
$\neg g(X, U, V) \doteq g(Y, fc, fd)$
$\neg g(U, X, Y) \doteq g(V, fa, fb)$
$a \doteq b$
$c \doteq d$
$a \doteq b$
$c \doteq d$
$\neg g(X_1, U_1, V_1) \doteq g(Y_1, fc, fd)$
$\neg g(X_2, U_2, V_2) \doteq g(Y_2, fc, fd)$
$\neg g(U_1, X_1, Y_1) \doteq g(V_1, fa, fb)$
$\neg g(U_2, X_2, Y_2) \doteq g(V_2, fa, fb)$

Figure 1 A tableau with equality.

Here, the unification

$$[X_1/Y_1,\ U_1/fc,\ V_1/fd]$$

is found. On the next branch

$$\{a \doteq b,\ c \doteq d,\ \neg g(X, U, V) \doteq g(Y, fc, fd)\} \quad ,$$

the negated equation may be rewritten by using the equations. Assuming a term ordering where c is larger than d, this will give

$$\neg g(X, U, V) \doteq g(Y, fd, fd) \quad .$$

The corresponding unifier is joined to the other one, giving

$$[X_1/Y_1,\ U_1/fc,\ V_1/fd, X/Y,\ U/fd,\ V/fd] \quad .$$

The next branch

$$\{c \doteq d,\ a \doteq b,\ \neg g(U, X, Y) \doteq g(V, fa, fb)\}$$

also allows rewriting. If we assume a to be greater than b in the ordering, we will get

$$\neg g(U, X, Y) \doteq g(V, fb, fb) \quad .$$

The unifier of these terms can also be joined to the previously collected one, giving

$$[X_1/Y_1,\ U_1/fc,\ V_1/fd,\ X/fb,\ Y/fb,\ U/fd,\ V/fd] \quad .$$

For the last branch, no rewriting is possible, and a unifier like the one for the first branch is produced. Putting all together, we have found the following simultaneous E-unifier.

$$\begin{array}{ccc} [& X_1/Y_1,\ U_1/fc,\ V_1/fd, & \\ & X/fb,\ Y/fb,\ U/fd,\ V/fd, & \\ & U_2/V_2,\ X_2/fa,\ Y_2/fb &] \end{array}$$

To summarize, although there is a simultaneous E-unifier for the tableau after only one γ-expansion, a procedure using rewriting and syntactic unification independently on the branches will not find it. This procedure is thus incomplete for simultaneous rigid E-unification. After two more γ-expansions, however, this same procedure finds a unifier that closes the tableau.

The procedure we applied in this example is actually part of a typical procedure to compute per-branch solutions of rigid E-unification problems. The question whether this approach leads to a complete tableau calculus for first-order logic with equality remained open for some years.

In 1997, Degtyarev and Voronkov finally showed completeness for such a combination of tableaux and rigid E-unification [16, 17] and thus for a tableau calculus with integrated superposition-based equality handling.

One might have expected that all problems would be solved after this discovery. A number of publications would follow, providing variations on the theme, like what is known as 'basic ordered paramodulation' in the resolution community, or a version with universal variables (see e.g. [28]), or hyper tableaux [7] with equality. Curiously enough, this has not happened! We surmise that the reason rests with the complexity of Degtyarev and Voronkov's completeness proof: It is over ten pages long, and very technical, although the proof of one of the used theorems is not even included in those papers.[2]

In this article, we describe a calculus similar to the one of [17], though we prefer to integrate the superposition process into the tableau calculus instead of defining a separate calculus for rigid E-unification. We then show the completeness of this calculus using an adaptation of the technique called *model generation*, well known for resolution calculi. This technique was first introduced by Bachmair and Ganzinger [2–4]) to show completeness of resolution calculi with strict superposition. Nieuwenhuis and Rubio have adapted the model generation technique for resolution calculi with constraint propagation (Rubio [46]; Nieuwenhuis and Rubio [39–41]), which is nearer to our application. For that reason, we will follow Nieuwenhuis and Rubio in our notation and presentation.

Apart from being significantly shorter than the proof of Degtyarev and Voronkov, our completeness proof has the advantage of requiring only few additional ingredients not known from resolution. This should make it easy to produce tableau versions of variants known for resolution, like basic ordered paramodulation (see Section 8) or hyper resolution, resp. hyper tableaux.

3 Preliminaries

We start by introducing a few concepts and notations, most of which are fairly standard.

We assume a fixed signature consisting of function symbols with fixed arity, constant symbols being considered as functions of arity zero. We also require an infinite supply of variable symbols. Terms over this signature are defined in the usual way. A ground term is a term that does not contain variables.

We admit only a single binary predicate symbol '$\doteq$' denoting equality. The equality symbol is handled in a symmetric way, namely, two formulas $s \doteq t$ and $t \doteq s$ are considered identical. A literal is either an equation $s \doteq t$ or a negated equation $\neg s \doteq t$. We do not include other predicates for two reasons. First, it is well-known that other predicates can be simulated with equality, by introducing a constant tt and representing a literal $p(t_1, \ldots, t_n)$ by an equation $p(t_1, \ldots, t_n) \doteq tt$, while taking care, for example, through the use of two different sorts, that tt cannot appear in any other equations. Second, it is conceptually easy to modify our calculus and proofs to work with nonequality predicates, but doing so makes them harder to read.

We limit our exposition to problems given in *Skolemized negation normal form*, for reasons we explain at the beginning of Section 4.

Definition 1 A *formula (in Skolemized negation normal form)* is

- a literal, or
- a conjunction $\phi \wedge \psi$ of two formulas, or
- a disjunction $\phi \vee \psi$ of two formulas, or
- a universally quantified formula $\forall x.\phi$ for some formula ϕ and a variable x.

[2]Theorem A.15 in [16] is taken from [14].

In other words, in contrast to the usual definition of formulas, we allow negations only in front of equations, and we do not use existential quantifiers. The reason for this restriction is that handling negation at arbitrary positions in a formula complicates our proofs by adding a dual case for every operator, without adding any insight. Similarly, the choice of a δ-rule for existential quantifiers is completely orthogonal to the problem of equality handling.

Substitutions are often defined as mapping a finite number of variables to arbitrary terms. In this article, we sometimes need substitutions that act on infinitely many variables simultaneously. Also, all variables are mapped to ground terms. We therefore consider substitutions to be functions from the set of all variables to the set of ground terms.

We will talk about models and interpretations in our completeness proofs. Contrary to the approach usually taken in discussions of full first-order logic, namely, to use a carrier set and an interpretation function to evaluate terms, we will follow [41] in defining an interpretation to be a congruence relation on ground terms. In other words, we take a quotient of the set of all ground terms as carrier set, and interpret terms to their congruence class. We do not need to define an interpretation for predicates, as we have only one predicate $\doteq$, which is interpreted as equality of equivalence classes. Herbrand's theorem guarantees that for our purposes, this definition is equivalent to defining interpretations with arbitrary carrier sets. Validity of a formula ϕ in an interpretation I will be written $I \models \phi$. Interpretations will often be described by sets of rewrite rules in the way captured by the following definition.

Definition 2 A *(ground) rewrite rule* is an ordered pair $l \Rightarrow r$ of ground terms. If R is a set of rewrite rules, the *interpretation induced by* R is the minimal congruence R^* on ground terms, such that $l R^* r$ for all $l \Rightarrow r \in R$.

The following notions are needed to formalize the application of equations on some subterm of a literal.

Definition 3 A *position* is a sequence of numbers designating subterms. Specifically, $s|_p$ is the subterm of s at position p, that is,

- $s|_\lambda = t$, where λ is the empty sequence.
- $f(s_1, \ldots, s_n)|_{k.q} = s_k|_q$ for $1 \le k \le n$.

Here $s[r]_p$ denotes the result of replacing the subterm at position p in s by r, that is,

- $s[r]_\lambda = r$, and
- $f(s_1, \ldots, s_n)[r]_{k.q} = f(s_1, \ldots, s_{k-1}, s_k[r]_q, s_{k+1}, \ldots, s_n)$ for $1 \le k \le n$.

We also need to fix a suitable rewrite ordering. In contrast to the usual definitions, we do not need to order terms with variables.[3]

[3]Our calculus uses free variables, but these will always be instantiated before we need to compare them.

Definition 4 A *total ground reduction ordering* is a total ordering $\succ$ on the set of ground terms, which is

– well-founded, that is, there is no infinite chain $t_0 \succ t_1 \succ t_2 \succ \cdots$, and
– monotonic, that is, $u[s]_p \succ u[t]_p$ for all ground terms u, s, t with $s \succ t$ and positions p in u.

A total ground reduction ordering $\succ$ is extended to a total well-founded ordering $\succ_l$ on ground literals as follows. Every ground literal is assigned a multiset by $m(s \doteq t) := \{s, t\}$ and $m(\neg s \doteq t) := \{s, s, t, t\}$. Then $L \succ_l L'$ iff $m(L) \ggcurly m(L')$, where $\ggcurly$ is the multiset extension of $\succ$.

We will need the following well-known fact in our proofs.

Proposition 1 *Any total ground reduction ordering $\succ$ enjoys the subterm property, that is, $u \succ t$ for all proper subterms t of a ground term u.*

It will also be useful to keep the following properties of the literal ordering $\succ_l$ in mind, which follow immediately from this definition.

Proposition 2 *If $s \succ t$ and $s' \succ t'$, then*

$$\begin{aligned}
s \doteq t \succ_l s' \doteq t' \quad &\textit{iff} \quad s \succ s' \textit{ or } (s = s' \textit{ and } t \succ t') \\
s \doteq t \succ_l \neg s' \doteq t' \quad &\textit{iff} \quad s \succ s' \\
\neg s \doteq t \succ_l s' \doteq t' \quad &\textit{iff} \quad s \succeq s' \\
\neg s \doteq t \succ_l \neg s' \doteq t' \quad &\textit{iff} \quad s \succ s' \textit{ or } (s = s' \textit{ and } t \succ t').
\end{aligned}$$

We assume a fixed total ground reduction ordering $\succ$, and corresponding literal ordering $\succ_l$, throughout the remainder of this article. In our examples, we occasionally use the lexicographic path ordering [29] on ground terms.

Definition 5 Let $>$ be a strict total ordering on the function and constant symbols of a signature, called *precedence*. We inductively define the *lexicographic path ordering* (LPO) $\succ$ on the set of ground terms as follows. Let $s = f(s_1, \ldots, s_m)$ and $t = g(t_1, \ldots, t_n)$. Then $s \succ t$, iff

– $s_i \succeq t$ for some $i \in \{1, \ldots, m\}$, or
– $f > g$ and $s \succ t_j$ for all $j \in \{1, \ldots, n\}$, or
– $f = g$, and there is some $j \in \{1, \ldots, n\}$ such that $s_i = t_i$ for $i < j$, $s_j \succ t_j$, and $s \succ t_i$ for $i > j$.

It is well-known that the LPO is a total ground reduction ordering for any (strict, total) precedence.

Our equality-handling rules will use constrained formulas (in fact only constrained literals) as we construct our tableaux. For this article, a constraint is a quantifier-free first-order formula that can use two predicate symbols with fixed interpretation, namely, '$\equiv$' representing (syntactic) equality and and '$\succ$' for the reduction ordering.

To make constraints optically different from the usual formulas, we denote conjunction as '&' in constraints. Disjunction and negation will not be needed. A substitution satisfies a constraint if the constraint is true under the given fixed interpretation when its free variables are assigned values according to the substitution. A constraint is satisfiable if there is a substitution that satisfies it.

Example 3 The constraint

$$X \equiv f(Y) \,\&\, Y \succ X$$

is not satisfiable under any total ground reduction ordering, due to the subterm property. The constraint

$$X \succ a \,\&\, b \succ X$$

may or may not be satisfiable, depending on whether there is a ground term between a and b in the chosen term ordering.

A family of practical algorithms for checking the satisfiability of constraints interpreted over recursive path orderings (RPOs) has been given by [38]. Satisfiability of constraints interpreted over Knuth-Bendix orderings has been investigated by [34].

A *constrained formula* is a pair $\phi \ll C$ of a formula and a constraint. The intention is that the formula may be used to close a branch only if the instantiation of the free variables of a tableau satisfies the constraint.

We occasionally refer to *rigid* versus *universal* variables. Rigid variables are the free variables introduced by the γ-rule of a free variable tableau calculus. They are called rigid because all occurrences of such variables in a tableau have to be instantiated by the same terms. This is very different from the situation in a resolution calculus, where each new clause is implicitly universally quantified and variables in a clause may be instantiated by a different m.g.u. in each resolution step.

In certain cases this restriction can be lifted in tableau calculi, namely, where it is sound to pick different instantiations for multiple occurrences of a free variable. If that is the case, one calls the free variable "universal" for the formula in which it occurs; see, for example, [28]. As has been pointed out by [11], using universal variables is crucial for efficient equality handling in tableaux.

We start by presenting our calculus without universal variables, but in Section 7 we will describe how they may be added.

4 The Calculus

We describe here an analytic free-variable tableau calculus to refute formulas in Skolemized negation normal form (SNNF). One reason for considering SNNF, instead of simply clauses as we did in our previous work, is that tableaux or sequent calculi are often applied in situations where using a clause normal form is not an option, for instance in modal or dynamic logics, but also when a proof tree has to be presented to a human, for example, for some kind of interaction. A second reason for considering SNNF is that we wish to compare our work to that of [17], which is also formulated for SNNF.

The reason for not treating full first-order logic, with existential quantifiers, implication, equivalence, and negation at arbitrary positions is mainly one of presentation. It is well-known how to deal with these additions; they are obviously orthogonal to the question of equality handling, and they would only make this article longer than necessary.

In an analytic free-variable tableau calculus for SNNF formulas, there are of course the usual α-, β-, and γ-rules. In addition, we want rules to perform superposition between literals on a branch. In principle, for example, in a Smullyan-style calculus without free variables, the rule should look like

$$\frac{\begin{array}{c}(\neg)s \doteq t\\ l \doteq r\end{array}}{(\neg)s[r]_p \doteq t}$$

where p is a position in s, $s|_p = l$, $s \succ t$ and $l \succ r$.

That is, we apply ordered equations on maximal sides of literals.

In a calculus with free variables, the condition $s|_p = l$ becomes a unification problem, and the ordering conditions also depend on the instantiation of free variables. In their calculus,[4] Degtyarev and Voronkov annotate the whole tableau with a constraint to which these unification and ordering conditions are added. They require that constraint to be satisfiable at all times. A consequence of using such a global constraint is that backtracking is required for each superposition application that adds to the global constraint. To avoid this, we will work with *constrained literals* $(\neg)s \doteq t \ll C$, where the constraint C accumulates the unification and ordering conditions of superposition steps needed to derive *this* literal.

Furthermore, Degtyarev and Voronkov delete the literal $(\neg)s \doteq t$ from the branch in a superposition step. Of course, this also can be done only if the procedure backtracks over every superposition step. Our calculus is nondestructive: all rules just add formulas to the branches. We will, however, take up the discussion of destructive rules in Section 6.

Formally, our tableaux are trees where nodes are labeled with constrained formulas. In the terminology of [27], this is a *constrained formula tableau calculus*. To refute a formula ϕ, we start from an initial tableau that has only one node, labeled with ϕ. When no constraint is written, we mean the empty constraint that is satisfied by any substitution. We may use the following rules to expand the tableau:

$$\alpha\ \frac{\phi \wedge \psi}{\begin{array}{c}\phi\\ \psi\end{array}} \qquad\qquad \beta\ \frac{\phi \vee \psi}{\phi \quad | \quad \psi}$$

$$\gamma\ \frac{\forall x.\phi}{\phi[x/X]}$$

where X is a new free variable.

[4]From now on, we always refer to the "tableau basic superposition" calculus $\mathcal{TBSE}$ of [16].

$$\text{SUP-P}\ \frac{\begin{array}{c} s \doteq t \ll A \\ l \doteq r \ll B \end{array}}{s[r]_p \doteq t \ll s|_p \equiv l \,\&\, s \succ t \,\&\, l \succ r \,\&\, A \,\&\, B}$$

where p is a position in s and $s|_p$ is not a variable.

$$\text{SUP-N}\ \frac{\begin{array}{c} \neg s \doteq t \ll A \\ l \doteq r \ll B \end{array}}{\neg s[r]_p \doteq t \ll s|_p \equiv l \,\&\, s \succ t \,\&\, l \succ r \,\&\, A \,\&\, B}$$

where p is a position in s and $s|_p$ is not a variable.

The superposition rules SUP-P and SUP-N are applied only if the constraint of the new literal is satisfiable. The two literals involved as premises in the SUP-P-rule are required to be distinct,[5] although one might differ from the other only by a renaming of variables.

Superposition rules are applied only between literals. It would be possible to allow certain applications of equalities in other formulas, but care needs to be taken with possible clashes with bound variables.

The superposition rules attach constraints to the literals on a branch. These constraints are *propagated* by the rules, which means that any literal derived from some constrained literals carries at least the constraints of the original literals, and possibly new ones stemming from the rule application.

A ground substitution σ closes a branch $\mathcal{B}$ of a tableau if there is a constrained negated equation $(\neg s \doteq t \ll A) \in \mathcal{B}$ such that $\sigma s = \sigma t$ (that is syntactic identity) and σ satisfies A. The whole tableau is closed if there is a single substitution σ that closes all branches of the tableau simultaneously.

The SUP-rules implement what is known as *rigid basic superposition*. The term "rigid" refers to the rigidity of the free variables of our tableau calculus. One talks of superposition when only ordered application of equations is allowed, and *only on the maximal side* of an equation, which in our case is enforced by the constraint $s \succ t$. The basicness restriction forbids application of equations on subterms of this literal introduced by the unifier μ. In other words, superposition steps at or below variable positions of $s[r]_p \doteq t$ are excluded. In our constraint-based calculus, we get this restriction automatically, because the unifier is encoded in the constraint instead of being actually applied, and because we forbid superpositions at or below variable positions.

Example 4 We give a simple example to demonstrate how the calculus works. We will show that the following formula is unsatisfiable.

$$\neg fb \doteq a \ \wedge\ \forall x.\forall y.\, fx \doteq gy \ \wedge\ \forall z.(fz \doteq a \vee gz \doteq a)$$

[5]This restriction cannot be imposed in calculi dealing with universal equations, such as the original Knuth–Bendix completion, unfailing Knuth–Bendix completion, or resolution saturation procedures. In the present work, we can require the literals to be distinct because we have *rigid* variables. With rigid variables, a term can't be unified with one of its proper subterms, so superposition of some $s \doteq t$ with $s \doteq t$ would be possible only at the top position, leading to the trivial equation $t \doteq t$. Superposition with the symmetric equation, that is, between $s \doteq t$ and $t \doteq s$, is excluded by the ordering constraints, which would require $s \succ t \,\&\, t \succ s$.

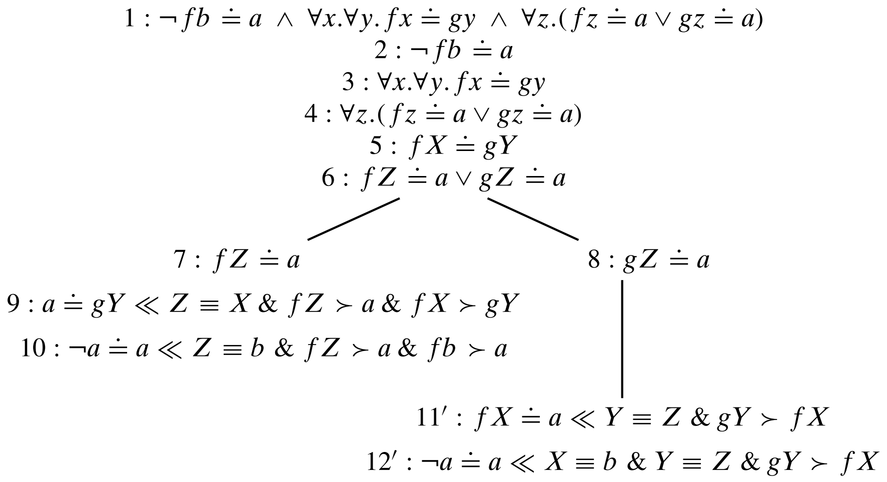
$1 : \neg fb \doteq a \;\wedge\; \forall x.\forall y.fx \doteq gy \;\wedge\; \forall z.(fz \doteq a \vee gz \doteq a)$
$2 : \neg fb \doteq a$
$3 : \forall x.\forall y.fx \doteq gy$
$4 : \forall z.(fz \doteq a \vee gz \doteq a)$
$5 : fX \doteq gY$
$6 : fZ \doteq a \vee gZ \doteq a$
$7 : fZ \doteq a$
$8 : gZ \doteq a$
$9 : a \doteq gY \ll Z \equiv X \;\&\; fZ \succ a \;\&\; fX \succ gY$
$10 : \neg a \doteq a \ll Z \equiv b \;\&\; fZ \succ a \;\&\; fb \succ a$
$11' : fX \doteq a \ll Y \equiv Z \;\&\; gY \succ fX$
$12' : \neg a \doteq a \ll X \equiv b \;\&\; Y \equiv Z \;\&\; gY \succ fX$

Figure 2 A tableau using the rigid basic superposition rules.

We will allow ourselves to gather several applications of α- and γ-rules into one step. The finished tableau is given in Figure 2, but we will explain how it is constructed, step by step. As term ordering, we choose a lexicographic path ordering (see Definition 5) with precedence $g > f > b > a$. Applying the α-rule twice on the initial formula gives us the following branch:

$$\begin{aligned}
&1 : \neg fb \doteq a \;\wedge\; \forall x.\forall y.fx \doteq gy \;\wedge\; \forall z.(fz \doteq a \vee gz \doteq a)\\
&2 : \neg fb \doteq a\\
&3 : \forall x.\forall y.fx \doteq gy\\
&4 : \forall z.(fz \doteq a \vee gz \doteq a)
\end{aligned}$$

We apply the γ-rule twice for the two quantifiers of 3 and once on 4 to get

$$\begin{aligned}
&5 : fX \doteq gY\\
&6 : fZ \doteq a \vee gZ \doteq a.
\end{aligned}$$

Between 2 and 5, a SUP-N application is possible by overlapping the left sides. The resulting literal is

$$\neg gY \doteq a \ll X \equiv b \;\&\; fb \succ a \;\&\; fX \succ gY \quad .$$

The constraint of this literal is unsatisfiable because it requires X to be instantiated with b, but every term gY is larger than fb under the LPO with the chosen precedence. Accordingly, this rule application is not allowed. Instead, we need to further expand the tableau, using the β-rule on 6. We now get two branches, with

$$7 : fZ \doteq a$$

on the left branch, and

$$8 : gZ \doteq a$$

on the right one. On the left branch, a superposition step of 7 on 5 leads to the literal

$$9 : a \doteq gY \ll Z \equiv X \,\&\, fZ \succ a \,\&\, fX \succ gY \quad .$$

This time, the constraint is satisfiable, for instance by instantiating X and Z to ga, and Y to a. But there is no further inference between literal 9 and the other literals on this branch. In particular, superposition between the right sides of 9 and 5 is not possible because of the ordering constraint $fX \succ gY$ of 9. Superposition of 5 on 7 would lead to the same literal 9. But superposition of 7 on 2 produces

$$10 : \neg a \doteq a \ll Z \equiv b \,\&\, fZ \succ a \,\&\, fb \succ a \quad ,$$

the constraint of which is satisfied whenever Z is instantiated to b. As this is a negated equation between two identical terms, this branch is closed for $\sigma(Z) = b$. No further superposition steps can be performed without expanding the left branch with further γ-instances, so we now turn to the right branch, which currently contains the following three literals.

$$\begin{array}{l} 2 : \neg fb \doteq a \\ 5 : fX \doteq gY \\ 8 : gZ \doteq a \end{array}$$

These allow only one superposition step, namely, between 8 and 5, producing the literal

$$11 : fX \doteq a \ll Y \equiv Z \,\&\, gY \succ fX \,\&\, gZ \succ a \quad ,$$

the constraint of which is easily seen to be satisfiable. As gZ is always larger than a under the given ordering, we can slightly simplify the constraint:

$$11' : fX \doteq a \ll Y \equiv Z \,\&\, gY \succ fX \quad ,$$

Literal $11'$ permits one further superposition step on 2, which gives us

$$12 : \neg a \doteq a \ll X \equiv b \,\&\, fb \succ a \,\&\, fX \succ a \,\&\, Y \equiv Z \,\&\, gY \succ fX \quad ,$$

where the constraint may again be simplified by removing the trivially satisfied conjuncts, to

$$12' : \neg a \doteq a \ll X \equiv b \,\&\, Y \equiv Z \,\&\, gY \succ fX \quad .$$

No other superposition steps are possible on this branch. As any term gY is larger than fb in our term ordering, the branch is now closed under any σ with $\sigma(X) = b$ and $\sigma(Y) = \sigma(Z)$. Hence, we can close the tableau with

$$[X/b, Y/b, Z/b] \quad .$$

Thanks to the ordering constraints, only few superposition steps were possible, which led to a very small search space.

Various proof procedures that use this calculus can be designed, differing in their use of backtracking and constraint handling:

- One gets a calculus similar to that of Degtyarev and Voronkov if one does not keep the constraints together with the literals but instead gathers them all in one global constraint G that is required to be satisfiable. This approach introduces a backtracking choice point for each rule application that adds to the global constraint. In addition, branch closure requires backtracking, as usual in free variable tableaux: whenever a negated equation $\neg s \doteq t$ appears on a branch, a backtracking point is introduced, and the constraint $s \equiv t$ is added to G. The procedure tries to close the other branches, always keeping G satisfiable, and keeping below a certain instantiation depth limit. If this fails, extension of the branch is continued. If no proof is found within a given depth limit, the whole procedure is restarted with an increased limit (iterative deepening). In contrast to the classic formulation of tableaux, the unifiers generated in superposition applications and branch closures should *not* be applied to the tableau, as this would yield possibilities for new, spurious rule applications on the other branches, weakening the 'basicness' property. Of course, rule applications on other branches that generate constraints incompatible with the global constraint G need not be considered in this scheme.
  The only difference from the calculus of Degtyarev and Voronkov is that they discard the first premise in the superposition steps. (Of course, such premises have to be reintroduced when the procedure backtracks over such a step). We will discuss destructive rules in Section 6.
- One can avoid the backtracking points introduced by the SUP-rules by keeping the constraints of literals. These are added to the global closure constraint G only when a branch is closed; accordingly, backtracking is needed only over branch closures.
- One can use the *incremental closure* technique introduced by Giese [22, 25] to avoid backtracking completely.

Although we prefer the third choice, our results are equally valid for a backtracking proof procedure.

5 Completeness

It is not hard to see that the calculus given in the previous section is sound: all the ordering constraints simply restrict the cases when equalities need to be applied, and if they are left out, the soundness proof is really quite standard. As Sections 1 and 2 have hinted, the interesting issue is completeness: to show that for any unsatisfiable formula, there is a closed tableau.

The completeness proof follows the usual lines. Assuming that there is no closed tableau for the initial formula, one constructs an infinite tableau by applying rules exhaustively – in particular, the γ-rule has to be applied infinitely often for each γ-formula on each branch. Then one chooses a ground instantiation σ for the free variables such that, after applying the substitution to the tableau, every branch contains a copy of every ground instantiation of each of the γ formulas. From the assumption, it follows that at least one branch $\mathcal{B}$ of the tableau is not closed by σ. From the literals on $\sigma\mathcal{B}$, an interpretation is constructed, which is then shown to be a model for the initial formula, contradicting its unsatisfiability.

Our proof differs from this standard approach only in the construction of the interpretation and in the proof that the initial formula is indeed satisfied by it. The proof is similar to the ones presented for problems in clause normal form in our previous work [23, 24].

First, we need the following notion.

Definition 6 Given a set $\mathcal{B}$ of constrained formulas, a ground instantiation σ for all free variables occurring in $\mathcal{B}$, and a set R of ground rewrite rules, the set of *variable-irreducible ground instances of* $\mathcal{B}$ *under* σ *with respect to* R, written $irred_R(\sigma, \mathcal{B})$, is the set of all ground literals $(\neg)\sigma l \doteq \sigma r$, where $((\neg)l \doteq r \ll A) \in \mathcal{B}$, A is satisfied by σ, and σX is irreducible by R for all variables X occurring in l or r.

Note that irreducibility is not required for the whole terms σl and σr, but only for the instantiations of variables occurring in them. Also, the instantiation of variables occurring only in the constraint A is allowed to be reducible. We are going to work only on variable-irreducible ground instances of the constrained literals on a branch. The reason will become clear later.

Our notion of variable-irreducible ground instances differs from that used by Nieuwenhuis and Rubio in the resolution setting in that we consider instances under a particular instantiation σ of the rigid variables, and not the set of all instantiations.

Note also that $irred_R(\sigma, \mathcal{B})$ contains only literals. The nonliteral formulas in $\mathcal{B}$ are not taken into account at all.

We can now define the model generation process, which constructs a ground rewrite system by induction with $\succ_l$ over variable-irreducible ground instances of literals on a branch. As usual for model-generation-style proofs, the rewrite relation that variable irreducibility refers to is being built during the induction.

Definition 7 Let $\mathcal{B}$ be a set of constrained formulas and σ a ground substitution on all variables in $\mathcal{B}$. For any ground literal L, we define $\mathrm{Gen}(L) = \{l \Rightarrow r\}$ and say that L *generates the rule* $l \Rightarrow r$, iff

1. $L \in irred_{R_L}(\sigma, \mathcal{B})$,
2. $L = (l \doteq r)$,
3. $R_L^* \not\models L$,
4. $l \succ r$, and
5. l is irreducible w.r.t. R_L,

where $R_L := \bigcup_{L \succ_l K} \mathrm{Gen}(K)$ is the set of all rules generated by smaller literals. Otherwise, we define $\mathrm{Gen}(L) := \emptyset$. The set of all rules generated by any ground literal is denoted $R_{\mathcal{B},\sigma} := \bigcup_K \mathrm{Gen}(K)$.

Note that only positive equation literals generate rules. When no confusion is likely concerning the set $\mathcal{B}$ and the substitution σ, we will just write R instead of $R_{\mathcal{B},\sigma}$.

We will use the following two useful lemmas, which are slightly reformulated versions of Lemma 3.2 of [41].

Lemma 1 *For any set of constrained formulas $\mathcal{B}$ and ground substitution σ, the generated set of rules $R = R_{\mathcal{B},\sigma}$ is convergent, that is, confluent and terminating. The subset R_L is also convergent for any ground literal L.*

Proof R terminates because $l \succ r$ for all rules $l \Rightarrow r \in R$ (condition 4). To show confluence, by Newman's lemma, one thus needs only to show local confluence, which follows from the fact that there can be no critical pairs in R. Assume $l \Rightarrow r \in R$ and $l' \Rightarrow r' \in R$ with $l|_p = l'$. Let $l \Rightarrow r$ be generated by a literal K. Then $l' \Rightarrow r'$ cannot be in R_K, for otherwise condition 5 would have prevented the generation of $l \Rightarrow r$. So $l' \Rightarrow r'$ is generated by a literal K' with $K' \succ_l K$. But then either $l' \succ l$, which is impossible because l' is a subterm of l, or $l' = l$ and $r \succ r'$, but then l' would be reducible by $l \Rightarrow r$, violating condition 5 for $\text{Gen}(K') = \{l' \Rightarrow r'\}$.

For arbitrary ground literals L, $R_L \subseteq R$, so R_L is also terminating, and R_L cannot contain critical pairs either. Hence, R_L is also convergent. □

Lemma 2 *For all ground literals L, if $R_L^* \models L$, then $R^* \models L$.*

Proof Let $R_L^* \models L$.

Case 1 $L = (s \doteq t)$. R contains at least all the rewrite rules of R_L, i.e. $R \supseteq R_L$. Thus, the equation must also hold in R^*.

Case 2 $L = (\neg s \doteq t)$. According to Lemma 1, R_L is convergent, so s and t have distinct normal forms $s' \preceq s$ and $t' \preceq t$ w.r.t. R_L. Now consider rules $l \Rightarrow r \in R \setminus R_L$. By definition of R_L, their generating literals $l \doteq r$ must be larger than L in the literal ordering (they can't be equal because L is a negated equation). By the definition of $\succ_l$, this implies that $l \succ s \succeq s'$ and $l \succ t \succeq t'$. So rules in $R \setminus R_L$ cannot further rewrite s' or t'; hence these are the normal forms of s and t also w.r.t. R. And as they are distinct, $R^* \models \neg s \doteq t$. □

We can now show the central property of the model R^* constructed in Definition 7, namely, that it satisfies all the irreducible instances (w.r.t. R) of literals in $\mathcal{B}$ under certain conditions.

Lemma 3 *(Model Generation) Let $\mathcal{B}$ be a set of constrained formulas and σ a ground substitution for the free variables in $\mathcal{B}$ such that*

- $\mathcal{B}$ is closed under the application of the SUP-P and SUP-N rules, and
- there is no literal $\neg s \doteq t \ll A \in \mathcal{B}$ such that $\sigma s = \sigma t$ (syntactically) and σ satisfies A.

Then $R^* \models L$ for all $L \in irred_R(\sigma, \mathcal{B})$.

Proof Assume that this were not the case. Then there must be a *minimal* (w.r.t. $\succ_l$) L in $irred_R(\sigma, \mathcal{B})$ with $R^* \not\models L$. We distinguish two cases, according to whether L is an equation or a negated equation:

Case 1 $L = (s \doteq t)$. If $s = t$ syntactically, then clearly $R^* \models L$, so we may assume that $s \succ t$. As $R_L \subseteq R$, we certainly have $L \in irred_{R_L}(\sigma, \mathcal{B})$. Also, due to Lemma 2, we already have $R_L^* \not\models L$. But $\mathrm{Gen}(L) = \emptyset$, because otherwise the rule $s \Rightarrow t$ would be in R, implying $R^* \models L$. As conditions 1 through 4 for L generating a rule are fulfilled, condition 5 must be violated. This means that there is a rule $l \Rightarrow r \in R_L$ that reduces s, so $s|_p = l$ for some position p in s. Now let L be the variable-irreducible (w.r.t. R) instance of a constrained literal $L_0 = (s_0 \doteq t_0 \ll A) \in \mathcal{B}$. Similarly, let $l \Rightarrow r$ be generated by a literal $K = (l \doteq r) \prec_l L$ that is the variable-irreducible (w.r.t. R_K) instance of a constrained literal $K_0 = (l_0 \doteq r_0 \ll B) \in \mathcal{B}$. It turns out that p must be a nonvariable position in s_0, because otherwise, since $s = \sigma s_0$, we would have $p = p'p''$ with $s_0|_{p'} = X$ and $\sigma X|_{p''} = l$; thus σX would be reducible by $l \Rightarrow r \in R$, contradicting the variable-irreducibility of L.[6] From all this, it follows that an application of the SUP-P-rule between the literals $L_0, K_0 \in \mathcal{B}$ is possible.

$$\textsc{sup-p}\ \frac{\begin{array}{c} s_0 \doteq t_0 \ll A \\ l_0 \doteq r_0 \ll B \end{array}}{s_0[r_0]_p \doteq t_0 \ll s_0|_p \equiv l_0 \mathbin{\&} s_0 \succ t_0 \mathbin{\&} l_0 \succ r_0 \mathbin{\&} A \mathbin{\&} B}$$

As $\mathcal{B}$ is required to be closed under SUP-rule applications, the resulting literal, call it L_0', must be in $\mathcal{B}$. Now $L' := (s[r]_p \doteq t) = \sigma L_0'$ is a variable-irreducible (w.r.t. R) instance of L_0': indeed, σ obviously satisfies the new constraint. Furthermore, σX is irreducible by R for any variable X occurring in s_0 or t_0. For an X occurring in r_0, σX is known to be irreducible by rules in R_K. But for rules $g \Rightarrow d \in R \setminus R_K$, we have $g \succeq l \succ r \succeq \sigma X$, so g cannot be a subterm of σX. This shows that σX is irreducible by R for all variables X in L_0', so $L' \in irred_R(\sigma, \mathcal{B})$. Moreover, since l and r are in the same R^*-equivalence class, replacing l by r in s does not change the (non)validity of $s \doteq t$, namely, $R^* \not\models L'$. And finally, by monotonicity of the rewrite ordering $\succ$, $L \succ_l L'$. This contradicts the assumption that L is the minimal element of $irred_R(\sigma, \mathcal{B})$, which is not valid in R^*.

Case 2 $L = (\neg s \doteq t)$. If $s = t$ syntactically, then the second precondition of this lemma is violated, so we may assume $s \succ t$. Due to Lemma 2, $R_L^* \not\models L$, namely, $R_L^* \models s \doteq t$. According to Lemma 1, R_L is convergent. Validity of $s \doteq t$ in R_L^* then means that s and t have the same normal form w.r.t. R_L. This normal form must be $\preceq t$, and thus $\prec s$. Therefore, s must be reducible by some rule $l \Rightarrow r \in R_L$ with $s|_p = l$ for some position p. As in case 1, let L be the variable-irreducible (w.r.t. R) instance of a constrained literal $L_0 = (\neg s_0 \doteq t_0 \ll A) \in \mathcal{B}$, and let $l \Rightarrow r$ be generated by a literal $K = (l \doteq r) \prec_l L$ that is the variable-irreducible (w.r.t. R_K) instance of a constrained

[6]This is the place where the use of variable-irreducible instances is necessary. Otherwise, the combination of constraint inheritance and the nonvariable-position condition would give problems. This idea is also used by [41], but they need a slightly more complicated notion of variable irreducibility because they work with clauses.

literal $K_0 = (l_0 \doteq r_0 \ll B) \in \mathcal{B}$. Again as in case 1, p must be a nonvariable position in s_0. It follows that an application of the SUP-N rule is possible between L_0 and K_0.

$$\mathit{sup\text{-}n}\quad \frac{\begin{array}{c}\neg s_0 \doteq t_0 \ll A\\ l_0 \doteq r_0 \ll B\end{array}}{\neg s_0[r_0]_p \doteq t_0 \ll s_0|_p \equiv l_0 \,\&\, s_0 \succ t_0 \,\&\, l_0 \succ r_0 \,\&\, A \,\&\, B}$$

We can now show, in complete analogy with case 1, that $L' := (\neg s[r]_p \doteq t) \in \mathit{irred}_R(\sigma, \mathcal{B})$, $R^* \not\models L'$ and $L \succ_l L'$, contradicting the assumption that L is minimal in $\mathit{irred}_R(\sigma, \mathcal{B})$ with $R^* \not\models L$. □

We now have all the necessary tools to show that our calculus is complete in the sense that there exists a finite closed tableau for any unsatisfiable set of clauses. We are going to show a little more, namely that a closed proof will be found if we simply expand the tableau in a fair way without requiring backtracking. Of course, this property is partly due to the fact that we postpone the instantiation of free variables by defining a closed tableau as one where a substitution closing all branches simultaneously exists. If we closed branches one at a time, we would have to backtrack over branch closures, but not – contrary to what is the case in the $\mathcal{TBSE}$ calculus of Degtyarev and Voronkov – over every application of the superposition rules. To state the completeness theorem, we need the following definition of a fair proof procedure.

Definition 8 A *proof procedure* is a procedure that takes a SNNF formula ϕ and builds a sequence of tableaux $\mathcal{T}_0, \mathcal{T}_1, \mathcal{T}_2, \ldots$ from ϕ, where $\mathcal{T}_0$ is the initial tableau containing only ϕ, and each $\mathcal{T}_{i+1}$ results from the application of an α, β, γ, or SUP rule on one of the branches of $\mathcal{T}_i$. A proof procedure *finds a proof for* ϕ if one of the $\mathcal{T}_i$ is closed.

The *limit* of a sequence of tableaux constructed by a proof procedure is the supremum of the sequence under the initial subtree ordering.

A proof procedure is *fair* if, for any sequence of tableaux it constructs that does not contain a closed tableau, the following holds: If $\mathcal{T}$ is the limit of the sequence of constructed tableaux, then

- The γ-rule is applied infinitely often for every γ formula on every branch of $\mathcal{T}$.
- Every possible application of the SUP-rules between two literals on a branch of $\mathcal{T}$ is eventually performed on that branch.

Theorem 1 *Let ϕ be an unsatisfiable closed formula. Then a fair proof procedure finds a proof for ϕ.*

Proof Assume that the procedure does not find a proof. Then it constructs a sequence of tableaux $\mathcal{T}_0, \mathcal{T}_1, \mathcal{T}_2, \ldots$ with a limit $\mathcal{T}$. $\mathcal{T}$ has at least one open branch for any instantiation of the free variables in $\mathcal{T}$. For assume that under a certain σ all branches are closed. Then every branch contains a literal $\neg s \doteq t \ll C$, such that $\sigma s = \sigma t$ and σ satisfies C. Make a new tableau $\mathcal{T}'$ by cutting off every branch below some occurrence of such a literal. Then σ still closes $\mathcal{T}'$ and $\mathcal{T}'$ has only branches of finite length and is finitely branching. Thus, by König's lemma, $\mathcal{T}'$ must be a finite closed tableau for ϕ. One of the tableaux $\mathcal{T}_i$ must contain $\mathcal{T}'$ as initial subtableau, and thus $\mathcal{T}_i$ is closed under σ, contradicting the assumption that the procedure finds no proof.

We now fix the instantiation σ. Specifically, σ should instantiate the free variables introduced by the γ-rule in such a way that every branch of $\sigma\mathcal{T}$ contains $\psi[x/t]$ for every formula $\forall x.\psi$ on the branch and every ground term t. This is possible because the fairness of the procedure guarantees that the γ-rule is applied infinitely often for each clause on every branch.

There must now be a branch $\mathcal{B}$ of $\mathcal{T}$ such that $\mathcal{B}$ is not closed by σ. We apply the model generation of Definition 7 on $\mathcal{B}$ and σ to obtain a set of rewrite rules $R = R_{\mathcal{B},\sigma}$. As $\mathcal{B}$ and σ obviously satisfy the preconditions of Lemma 3, every variable-irreducible instance of $\mathcal{B}$ is valid in R^*.

It now remains to show that ϕ is valid in R^* to contradict the assumption that ϕ is unsatisfiable. To do so, we first inductively define a subset $\mathcal{B}'$ of the branch $\mathcal{B}$, by defining that any $\psi \ll A \in \mathcal{B}$ is also in $\mathcal{B}$ if

- $\psi = \phi$ is the initial formula, or
- ψ is on $\mathcal{B}$ due to an α or β expansion of a formula in $\mathcal{B}'$, or
- $\psi = \psi_0[x/X]$ for some $\forall x.\psi_0 \in \mathcal{B}'$, and σX is irreducible w.r.t. R, or
- $\psi \ll A$ is a literal derived by SUP-rules from only formulas in $\mathcal{B}'$, and A is satisfied by σ.

This definition guarantees that the instantiation under σ of all free variables in formulas of $\mathcal{B}'$ are irreducible under R.

We can now show by induction on formula depth that $R^* \models \sigma\psi$ for all $\psi \ll A \in \mathcal{B}'$:

- If $\psi \ll A$ is a literal, then $\sigma\psi \in \mathit{irred}_R(\sigma, \mathcal{B})$, and therefore $R^* \models \sigma\psi$ due to Lemma 3.
- If $\psi = \psi_1 \wedge \psi_2$, then by fairness of the proof procedure, both ψ_1 and ψ_2 are on the branch. Since $\psi \in \mathcal{B}'$, also $\psi_1, \psi_2 \in \mathcal{B}'$. By the induction hypothesis, $R^* \models \sigma\psi_1$ and $R^* \models \sigma\psi_2$, and therefore $R^* \models \sigma\psi$.
- If $\psi = \psi_1 \vee \psi_2$, by fairness, one of the ψ_i must be on the branch, and also in $\mathcal{B}'$. By induction, $R^* \models \sigma\psi_i$, and therefore $R^* \models \sigma\psi$.
- If $\psi = \forall x.\psi_0$, we show that $\psi[x/t_0]$ is valid for arbitrary ground terms t_0. For a given t_0, let t be the normal form w.r.t. R of t_0. Since t and t_0 belong to the same equivalence class of R^*, it suffices to show that $R^* \models \sigma(\psi[x/t])$. Due to fairness and the construction of σ, $\mathcal{B}$ contains an instance $\psi[x/X]$ with $\sigma X = t$. σX is irreducible, so $\psi[x/X] \in \mathcal{B}'$, and therefore, by induction, $R^* \models \psi[x/X] \in \mathcal{B}'$.

Since the initial formula ϕ is in $\mathcal{B}'$, we have shown $R^* \models \sigma\phi$. But ϕ is closed, so the substitution has no effect: $R^* \models \phi$, contradicting the unsatisfiability of ϕ. □

This concludes the completeness proof for our calculus. A comparison to the model generation proofs of [41] reveals strong similarities, in particular for the model generation lemma. The main difference is that our rigid variables require taking the global closing substitution σ into account, for example, in the definition of $\mathit{irred}_R(\sigma, \mathcal{B})$. The proof of Theorem 1 is, of course, specific to tableaux, and the part concerning $\mathcal{B}'$ to nonclausal tableaux.

We note that our model construction, unlike that in most Hintikka-style completeness proofs for tableaux calculi, does not attempt to satisfy all formulas on a branch: it is sufficient to consider the variable irreducible literals, since the irreducible instantiations of the free variables play the role of canonical elements of the equivalence classes modulo R^*.

6 Destructive Rules

The $\mathcal{TBSE}$ calculus of [16] deletes the first premise $(\neg)s \doteq t$ when a superposition rule is applied. This seems to be a natural action: replacing an occurrence of l by the smaller r has simplified the formula. It has become redundant because the same information is contained in the new, simpler formula.

Deleting the first premise is not directly possible in our constrained formula calculus. In a sense, the new constraint $C = s|_p \equiv l \,\&\, s \succ t \,\&\, l \succ r \,\&\, A \,\&\, B$ indicates a condition under which the superposition has been applied. And only under this condition should the first premise be considered to be deleted.

A possible solution to this problem is to use *disunification constraints*: We add a negation operator ! to our constraint language, and we replace the original formula $(\neg)s \doteq t \ll A$ by $(\neg)s \doteq t \ll A \,\&\, !\,C$. This has the effect of excluding further rule applications involving that formula in exactly those cases in which the superposition step was allowed according to C. See [25], Section 7.5, for a discussion of this approach.

Another solution is to use *histories*, as introduced in Section 7.4 of [25]: Every literal is annotated with a set of formulas called the history, containing all the formulae that could have been deleted in the course of its creation. Histories are propagated by rule applications similarly to constraints. A superposition step or branch closure does not need to be considered if one of the involved formulas is contained in the history of the other. This restriction is slightly weaker than the one using disunification constraints, but it is easier to implement, and the calculus remains completely nondestructive, since the deletion information is always kept with the newly derived formulas.

A third solution is to actually perform the deletion, but to backtrack over the SUP-application. This is essentially what is done in the calculus of Degtyarev and Voronkov.

We remark that it is not completely clear whether deleting the first premise in a superposition application is really a good idea: In a free variable calculus, it is often possible to delete formulas, since they can be reintroduced by using further γ-expansions. Nothing forbids instantiating the new free variables to the same terms as the free variables in a deleted formula. But in such a case, the deletion just caused additional work. The real question is this:

> If we forbid using the same instantiation for the free variables introduced by two γ expansions of the same formula on the same branch, is completeness retained if we "delete" (possibly using disunification constraints or histories) the first premise of the superposition rules?

Whether this is the case is not stated by [16], nor is it easy to infer this from their proof. We have not been able to prove it using the model generation technique, because of technical difficulties discussed in [24], Sections 7.4 and 7.5.

In order to reason about destructive rules, it would be helpful to have an adaptation of the concept of saturation modulo redundancy of [4] to the case of free variable tableaux. Unfortunately, such an adaptation has not yet been undertaken.

7 Universal Variables

Beckert [11] has pointed out that the rigidity of the free variables in tableaux is particularly problematic in the presence of equality: Problems often contain universally quantified equalities such as associativity, commutativity, and idempotency as axioms. Typically, proofs require fairly many applications of these axioms. With rigid variables, each such application requires as many applications of the γ-rule as there are variables in the axiom and as many new free variables are introduced that need to be instantiated. In general, this leads to an unnecessarily large search space.

Using *universal variables* (see, e.g., [28]) is a well-known refinement of the free-variable tableau calculus that is particularly effective in this case. Roughly, if a variable X is universal for some formula ϕ in a tableau, then it is sound to rename that variable in this formula and therefore to instantiate it independently of other occurrences in different formulas or on other branches. Simple heuristics are used to detect universal variables in the formulas of a tableau.

For the superposition rules, one can show that if the universal variables in the premises are renamed to be different from any occurring rigid variables, then any of those universal variables that appear in the result are still universal.

Example 5 If we start with the two formulas

$$\forall x.\forall y.\forall z.\ (x \cdot y) \cdot z \doteq x \cdot (y \cdot z)$$
$$\forall x.\forall y.\ x \cdot y \doteq y \cdot x,$$

we can repeatedly apply the γ-rule to obtain

$$(X \cdot Y) \cdot Z \doteq X \cdot (Y \cdot Z)$$
$$X' \cdot Y' \doteq Y' \cdot X',$$

where all the free variables are universal for the respective formulas. Using the second formula to rewrite the subterm $X \cdot Y$ of the first one with the SUP-P rule leads to

$$(Y' \cdot X') \cdot Z \doteq X \cdot (Y \cdot Z) \ll X \equiv X' \,\&\, Y \equiv Y' \,\&\, X' \cdot Y' \succ Y' \cdot X',$$

where all variables are still universal. Therefore, we can rename the variables:

$$(V' \cdot U') \cdot W \doteq U \cdot (V \cdot W) \ll U \equiv U' \,\&\, V \equiv V' \,\&\, U' \cdot V' \succ V' \cdot U'.$$

Since the variables are universal, we might also simplify the constrained formula by keeping only one of U and U', resp. V and V':

$$(V \cdot U) \cdot W \doteq U \cdot (V \cdot W) \ll U \cdot V \succ V \cdot U.$$

Finally, under a lexicographic path ordering, we can replace the ordering constraint by an equivalent but simpler one:

$$(V \cdot U) \cdot W \doteq U \cdot (V \cdot W) \ \ll\ U \succ V.$$

Since all variables are universal for this formula, we can use it repeatedly for superposition on other formulas without generating further γ-instances of the original formulas.

If universal variables are used, we see that the behavior of our superposition rules for sets of universal formulas is essentially a variant of unfailing Knuth-Bendix completion [1] with constraints, similar to the calculi of [41]. The similarity would be even stronger if redundant formulas could be deleted in some way, in the style of [4], as was mentioned at the end of the last section.

Without universal variables, our calculus corresponds to the combination of a tableau calculus without equality and a solver for rigid E-unification problems, as described in Section 2. If we take universal variables as well as rigid variables into account, our calculus corresponds to the combination of tableaux with a solver for *mixed universal and rigid E-unification* as described by Beckert [10, 11].

8 Tableaux with Basic Ordered Paramodulation

At the beginning of this article, we claimed that the model generation completeness proof can easily be adapted to variants of the calculus. In this section, we demonstrate how variations of calculi and completeness proofs can be carried over from known results for resolution-based calculi.

A more restrictive form of equality handling is known in the resolution community as *basic ordered paramodulation* [5]. In comparison to basic superposition, the basicness restriction is strengthened: One forbids paramodulation below a position where a previous paramodulation step has taken place. The price to pay is that equations have to be applied on both sides of literals and not only the maximal side as for basic superposition. Still, basic ordered paramodulation seems to be effective in practice [36].

Using constrained literals, one can easily enforce this stronger basicness restriction by introducing a new free variable in the equality handling rules. The SUP-P rule becomes

$$\text{PAR-P}\quad \frac{\begin{array}{c} s \doteq t \ll A \\ l \doteq r \ll B \end{array}}{s[X]_p \doteq t \ll X \equiv r \,\&\, s|_p \equiv l \,\&\, l \succ r \,\&\, A \,\&\, B}$$

where p is a position in s, $s|_p$ is not a variable,
and X is a new (free) variable.

Note how the constraint forces X to be instantiated with r and that the restriction $s \succ t$ is gone.[7] The PAR-N-rule is exactly analogous. This modification is a straightforward

[7] It might seem that introducing a new free variable is not a good idea. But these are harmless because there is no need to search for their instantiation. It is determined by the instantiations of the free

adaptation of the formulation of basic ordered paramodulation using constraint inheritance given by [41].

How do we show completeness of our modified calculus? We cite [41]:

> The completeness proof is an easy extension of the previous results by the model generation method. It suffices to modify the rule generation by requiring, when a rule $l \Rightarrow r$ is generated, that both l and r are irreducible by R_C, instead of only l as before and to adapt the proof of Theorem 5.6 accordingly, which is straightforward.

All we need is a little "signature mapping" to apply this to our situation: their Theorem 5.6 corresponds closely to our Lemma 3. They have R_C instead of our R_L because they have to work with ground clauses, where we can use literals. Otherwise, this statement applies exactly to our case.

In the definition of the model generation process (Definition 7 on page 18), let us replace condition 5 by

5. *l and r are irreducible w.r.t.* R_L.

A close scrutiny of the proofs of Lemmas 1 and 2 satisfies us that they are still valid after this modification. And it is indeed quite straightforward to adapt the proof of Lemma 3 on page 16 f.: for case 1, we drop the assumption that $s \succ t$, and infer that condition 5 must be violated as before. As we take a symmetric view of equations, we can now assume that it is s that is reducible by some rule in R_L. One then shows as before that a PAR-P application is possible. Showing that L' is a variable-irreducible instance of some L'_0 is even simpler than before because we do not need to account for variables in r_0. To show that σX is irreducible, note that $\sigma X = r$, and as $l \doteq r$ generates a rule, condition 5 guarantees that r is irreducible. Similar modifications apply for case 2. All these actions correspond exactly to what needs to be done for resolution.

The only new and tableau-specific part is that σ has to provide an instantiation for the free variables X introduced in the paramodulation steps in such a way that the new constraints are satisfied. But fortunately, this is also easily done: in an induction over the superposition steps leading to the deduction of a literal, let $\sigma X := \sigma r_0$ for a free variable X introduced by a superposition with $l_0 \doteq r_0 \ll B$.

We think that this example is strong evidence in support of our claim that model generation completeness proofs are a good basis for adapting known results from resolution with superposition or paramodulation to a tableau setting.

9 Related Work

Using techniques based on ordered rewriting is not the only way to add efficient equality handling to a tableau prover. We believe that it is the most powerful way

variables in r. In fact, these new variables are universal as they are restricted to a fixed instantiation by the constraint.

because of the success of such techniques in resolution provers such as SPASS [47] or Vampire [43].

The other commonly employed technique is to transform problems in a way that makes equality axioms redundant [6, 13, 37]. These transformation techniques analyze the problem in order to find possible instances of paramodulation steps and then code the results of these paramodulations into the problem before the actual proof search starts. Classically, term orderings are not used, although [6] introduce ordering constraints in some cases. The great advantage of transformation techniques is that one can add equality handling to a prover by a preprocessing step, instead of changing the actual prover. On the other hand, we expect a direct integration of equality handling into the calculus to be more efficient.

Another technique for equality handling in sequent-based calculi has been proposed by [21] under the name of *lazy paramodulation*. In this approach, equations are applied only on top-level terms, but term orderings cannot be used. We do not know how this approach compares to a superposition-based calculus in practice.

Letz and Stenz [35] have published an overview on the integration of equality handling into Billon's disconnection calculus [12]. It seems, however, that most of the techniques from analytic tableaux or resolution are not easily applicable to this instance-based method. Baumgartner and Tinelli [8] have recently had more success with the integration of ordered rewriting and saturation up to redundancy into the model evolution calculus, another member of the instance-based family. The role of free variables in instance-based methods is, however, subtly different from that in "standard" free-variable tableaux, making it hard to compare the results in that setting to those of this article.

10 Conclusion

We presented a free-variable tableau calculus for formulas in Skolemized negation normal form with integrated equality handling using basic superposition rules with constraint propagation. We demonstrated how the completeness of such a calculus can be shown using model generation techniques known from resolution calculi with only a few additional tableau-specific ingredients. Although completeness of a similar calculus has previously been established by [16] using a different approach, our proof is much shorter, and we have demonstrated that it is easily adapted to related calculi.

One area for future research is to experiment with an implementation. In particular, it would be interesting to see how well a prover based on this calculus performs on non-Horn equality problems.

Another interesting direction would be to incorporate superposition-based equality handling into (a rigid variable version of) hyper tableaux [7]. The ideas from resolution should be applicable also to this case.

Still another important field for research is building in associativity and commutativity or other common equational theories. We expect that this can be done in the same way as has been done for resolution, for example by [40].

For a systematic approach to calculi that perform actual destructive simplification steps – a prerequisite for efficiency – an adaptation of the concepts of redundancy and saturation to free variable tableaux is needed. We are currently investigating such an adaptation and have obtained promising first results [26].

Acknowledgements I am grateful to the anonymous referees for their valuable comments. I also thank Bernhard Beckert, Reiner Hähnle, and Peter Schmitt for many fruitful discussions and good advice, leading to the results presented in this article.

References

1. Bachmair, L., Dershowitz, N., Plaisted, D.A.: Completion without failure. In: Aït-Kaci, H., Nivat, M. (eds.) Resolution of Equations in Algebraic Structures, vol. 2, pp. 1–30. Rewriting Techniques. New York: Academic (1989)
2. Bachmair, L., Ganzinger, H.: On restrictions of ordered paramodulation with simplication. In: Stickel, M.E. (ed.) Proceedings of CADE-10, Kaiserslautern, Germany, vol. 449 of LNCS, pp. 427–441. Springer, Berlin Heidelberg New York (1990)
3. Bachmair, L., Ganzinger, H.: Rewrite-based equational theorem proving with selection and simplificat ion. J. Log. Comput. **4**(3), 217–247 (1994)
4. Bachmair, L., Ganzinger, H.: Resolution theorem proving. In (Robinson and Voronkov, 2001), Chap 2, pp. 19–99, Amsterdam, The Netherlands (2001)
5. Bachmair, L., Ganzinger, H., Lynch, C., Snyder, W.: Basic paramodulation. Inf. Comput. **121**(2), 172–192 (1995)
6. Bachmair, L., Ganzinger, H., Voronkov, A.: Elimination of equality via transformation with ordering constraints. Research Report MPI-I-97-2-012, Max-Planck-Institut für Informatik, Im Stadtwald, D-66123 Saarbrücken, Germany (1997)
7. Baumgartner, P.: Hyper tableaux – the next generation. In: de Swart, H. (ed.) Proc. International Conference on Automated Reasoning with Analytic Tableaux and Related Methods, Oosterwijk, The Netherlands, pp. 60–76. Springer, Berlin Heidelberg New York (1998)
8. Baumgartner, P., Tinelli, C.: The model evolution calculus with equality. In: Nieuwenhuis, R. (ed.) Proc., 20th International Conference on Automated Deduction, CADE. vol. 3632 pp. 392–408, LNCS, pp. 392–408. Springer, Berlin Heidelberg New York (2005)
9. Beckert, B.: Ein vervollständigungsbasiertes Verfahren zur Behandlung von Gleichheit im Tableaukalkül mit freien Variablen. Diplomarbeit, Fakultät für Informatik, Universität Karlsruhe (1993)
10. Beckert, B.: Adding equality to semantic tableaux. In: Broda, K., D'Agostino, M., Goré, R., Johnson, R., Reeves, S. (eds.) Proceedings, 3rd Workshop on Theorem Proving with Analytic Tableaux and Related Methods, Abingdon. Imperial College, London, TR-94/5, pp. 29–41 (1994a)
11. Beckert, B.: A completion-based method for mixed universal and rigid E-unification. In: Bundy, A. (ed.) Proc. 12th Conference on Automated Deduction CADE, Nancy/France, pp. 678–692. Springer, Berlin Heidelberg New York (1994b)
12. Billon, J.-P.: The disconnection method: a confluent integration of unification in the analytic framework. In: Miglioli, P., Moscato, U., Mundici, D., Hi, M.O. (eds.) Theorem Proving with Tableaux and Related Methods, 5th International Workshop, TABLEAUX'96, Terrasini, Palermo, Italy, vol. 1071 of LNCS, pp. 110–126. Springer, Berlin Heidelberg New York (1996)
13. Brand, D.: Proving theorems with the modification method. SIAM J. Comput. **4**(4), 412–430 (1975)
14. Degtyarev, A., Voronkov, A.: Equality Elimination for Semantic Tableaux. Technical Report 90, Comp. Science Dept., Uppsala University, Sweden (1994)
15. Degtyarev, A., Voronkov, A.: The undecidability of simultaneous rigid E-unification. Theor. Comput. Sci. **166**(1–2), 291–300 (1996)
16. Degtyarev, A., Voronkov, A.: What you always wanted to know about rigid E-unification. Technical Report 143, Comp. Science Dept., Uppsala University, Sweden (1997)
17. Degtyarev, A., Voronkov, A.: What you always wanted to know about rigid E-unification. J. Autom. Reason. **20**(1), 47–80 (1998)
18. Degtyarev, A., Voronkov, A.: Equality reasoning in sequent-based calculi. In: Robinson, A., Voronkov, A. (eds.) Handbook of Automated Reasoning, vol. I, chap. 10, pp. 611–706. Elsevier, Amsterdam, The Netherlands (2001)
19. Fitting, M.C.: First-order Logic and Automated Theorem Proving, 2nd edn. Springer, Berlin Heidelberg New York (1996)

20. Gallier, J., Raatz, S., Snyder, W.: Theorem proving using rigid E-unification: equational matings. In: Proc. IEEE Symp. on Logic in Computer Science, pp. 338–346. IEEE Computer Society Press, Los Alamitos, CA (1987)
21. Gallier, J.H., Snyder, W.: Complete sets of transformations for general E-unification. Theor. Comput. Sci. **67**, 203–260 (1989)
22. Giese, M.: Incremental closure of free variable tableaux. In: Goré, R., Leitsch, A., Nipkow, T. (eds.) Proc. Intl. Joint Conf. on Automated Reasoning, Siena, Italy. Springer, Berlin Heidelberg New York (2001a)
23. Giese, M.: Model generation style completeness proofs for constraint tableaux with superposition. Technical Report 2001-20, Universität Karlsruhe TH, Germany (2001b)
24. Giese, M.: A model generation style completeness proof for constraint tableaux with superposition. In: Egly, U., Fermller, C.G. (eds.) Proc. Intl. Conf. on Automated Reasoning with Analytic Tableaux and Related Methods, Copenhagen, Denmark, vol. 2381 of LNCS, pp. 130–144. Springer, Berlin Heidelberg New York (2002a)
25. Giese, M.: Proof search without backtracking for free variable tableaux. Ph.D. thesis, Fakultät für Informatik, Universität Karlsruhe, Germany (2002b)
26. Giese, M.: Saturation up to redundancy for tableau and sequent calculi. In: Hermann, M., Voronkov, A. (eds.) Logic for Programming, Artificial Intelligence, and Reasoning, 13th Intl. Conf., LPAR 2006, Phnom Penh, Cambodia. (2006) (to appear)
27. Giese, M., Hähnle, R.: Tableaux + constraints. Also as Tech. Report RT-DIA-80-2003, Dipt. di Informatica e Automazione, Università degli Studi di Roma Tre, Italy (2003)
28. Hähnle, R.: Tableaux and related methods. In (Robinson and Voronkov), Chap. 3, pp. 100–178. Elsevier, Amsterdam, The Netherlands (2001)
29. Kamin, S., Lévy, J.-J.: Attempts for generalizing the recursive path orderings. Dept. of Computer Science, University of Illinois, Urbana, Illinois (1980) (available online at http://perso.ens-lyon.fr/pierre.lescanne/not_accessible.html)
30. Kanger, S.: A simplified proof method for elementary logic. Computer Programming and Formal Systems, pp. 87–94. (1963) (Reprinted as Kanger, 1983)
31. Kanger, S.: A simplified proof method for elementary logic. In: Siekmann, J.H., Wrightson, G. (eds.): Automation of Reasoning 1: Classical Papers on Computational Logic 1957–1966, pp. 364–371. Springer, Berlin Heidelberg New York (1983)
32. Knuth, D.E., Bendix, P.B.: Simple word problems in universal algebras. In: Leech, J. (ed.) Computational Problems in Abstract Algebra. Pergamon, Oxford, pp. 263–297 (1970) (Reprinted as Knuth and Bendix, 1983)
33. Knuth, D.E., Bendix, P.B.: Simple word problems in universal algebras. In: Siekmann, J.H., Wrightson, G. (eds.) Automation of Reasoning 2: Classical Papers on Computational Logic 1967–1970, pp. 342–376. Springer, Berlin Heidelberg New York (1983)
34. Korovin, K., Voronkov, A.: Knuth–Bendix constraint solving is NP-complete. In: Proc. Intl. Conf. on Automata, Languages and Programming (ICALP), vol. 2076 of LNCS, pp. 979–992 (2001)
35. Letz, R., Stenz, G.: Integration of equality reasoning into the disconnection calculus. In: Egly, U., Fermüller, C.G., (eds.) Proc. Intl. Conf. on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX-2002), Copenhagen, Denmark. Springer, Berlin Heidelberg New York (2002)
36. McCune, W.: Solution of the Robbins problem. J. Autom. Reason. **19**(3), 263–276 (1997)
37. Moser, M., Steinbach, J.: STE-modification revisited. AR-Report AR-97-03, Institut für Informatik, München, Germany (1997)
38. Nieuwenhuis, R., Rivero, J.M.: Solved forms for path ordering constraints. In: Narendran, P., Rusinowitch, M. (eds.) Proc. 10th International Conference on Rewriting Techniques and Applications (RTA), Trento, Italy, vol. 1631 of LNCS, pp. 1–15. Springer, Berlin Heidelberg New York (1999)
39. Nieuwenhuis, R., Rubio, A.: Theorem proving with ordering and equality constrained clauses. J. Symb. Comput. **19**(4), 321–351 (1995)
40. Nieuwenhuis, R., Rubio, A.: Paramodulation with built-in AC-theories and symbolic constraints. J. Symb. Comput. **23**(1), 1–21 (1997)
41. Nieuwenhuis, R., Rubio, A.: Paramodulation-based theorem proving. In (Robinson and Voronkov, 2001), Chap. 7, pp. 371–443, Elsevier, Amsterdam (2001)
42. Petermann, U.: A complete connection calculus with rigid E-unification. In: MacNish, C., Pearce, D., Pereira, L.M. (eds.) Logics in Artificial Intelligence (JELIA), pp. 152–166. Springer, Berlin Heidelberg New York (1994)

43. Riazanov, A., Voronkov, A.: Vampire 1.1 (System description). In: Goré, R., Leitsch, A., Nipkow, T. (eds.) Proc. Intl. Joint Conf. on Automated Reasoning, Siena, Italy, vol. 2083 of LNCS, pp. 376–380. Springer, Berlin Heidelberg New York (2001)
44. Robinson, A., Voronkov, A. (eds.): Handbook of Automated Reasoning. Elsevier, Amsterdam, The Netherlands (2001)
45. Robinson, G.A., Wos, L.: Paramodulation and theorem-proving in first-order theories with equality. In: Meltzer, B., Michie, D. (eds.) Machine Intelligence 4, pp. 135–150. Edinburgh University Press, Edinburgh, Scotland (1969)
46. Rubio, A.: Automated deduction with ordering and equality constrained clauses. Ph.D. thesis, Technical University of Catalonia, Barcelona, Spain (1994)
47. Weidenbach, C.: Combining superposition, sorts and splitting. In: Robinson, A., Voronkov, A. (eds.) Handbook of Automated Reasoning, vol. II, Chap. 27, pp. 1965–2013. Elsevier, Amsterdam, The Netherlands (2001)