Model Generation Style Completeness Proofs for Constraint Tableaux with Superposition

Martin Giese

giese@ira.uka.de

Institut für Logik, Komplexität und Deduktionssysteme,
Universität Karlsruhe, Germany

Technical Report Nr. 2001–20

December 2001

Abstract

We present several calculi that integrate equality handling by superposition and ordered paramodulation into a free variable tableau calculus. We prove completeness of this calculus by an adaptation of the *model generation* [1, 13] technique commonly used for completeness proofs of resolution calculi. The calculi and the completeness proof are compared to earlier results of Degtyarev and Voronkov [8].

1 Introduction

Efficient equality handling for first order tableaux or related calculi, like matings or the connection method, has been problematic for a long time. It is generally believed that only techniques based on ordered rewriting can sufficiently reduce the search space of equality reasoning to make it tractable. It was also believed, that the best approach to the integration of free variable tableaux and equality handling would be to search for *simultaneous rigid E-unifiers* [9] of disequations on the tableau and use these to close branches instead of usual unifiers. So the overall idea was to solve the rigid E-unification problems using ordered rewriting techniques.

Unfortunately, simultaneous rigid E-unification was later shown to be undecidable [6].[1] The outlined plan could thus only be implemented using incomplete procedures for E-unification. Experimenting with such a setting however, it turned out, that the *combination* of a first order theorem prover and an incomplete solver for rigid E-unification problems seemed to be complete despite the incompleteness of the unification machinery, though nobody knew exactly why. In 1997, Degtyarev and Voronkov finally showed completeness for such a combination [7, 8], 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. [11]), or hyper tableaux [4] with equality. Curiously enough, this has not happened! We surmise that the reason for this is 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.

In this paper we present calculi similar to (a clausal version of) the one presented in [8], 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, see [1, 13]. In particular, we can use many ideas of Nieuwenhuis and Rubio that worked for them in the setting of resolution with constraint propagation. Apart from being significantly shorter than the proof of Degtyarev and Voronkov, our proof has the advantage of requiring only few additional ingredients not known from resolution. This makes it easy to produce tableau versions of variants known for resolution, like basic ordered paramodulation or hyperresolution resp. hyper tableaux.

[1] This was only shown in 1996, invalidating a number of attempts at a completeness proof that were based on the opposite assumption.

In Sect. 2 we review a couple of common notions and notations. Sect. 3 contains the simplest version of our calculus and the completeness proof. Sect. 4 discusses issues connected to a certain termination property. An alternative approach to the simulation of destructiveness in a non-backtracking calculus with constraints is briefly reviewed in Sect. 5. We try to demonstrate the versatility of the model generation technique for tableau completeness proofs in Sect. 6 by applying it to rigid basic ordered paramodulation. Finally, some possible fields for further research are identified in Sect. 7.

2 Preliminaries

We shall assume a fixed signature consisting of function symbols with fixed arity, constant symbols being considered as functions of arity zero, and a single binary predicate symbol '$\doteq$' denoting equality. The equality symbol is handled in a symmetric way, i.e. two formulae $s \doteq t$ and $t \doteq s$ are considered identical. A literal is either an equation $s \doteq t$ or a negated equation $\neg s \doteq t$. A clause is a finite set of literals.

An interpretation is a congruence relation on ground terms. Validity of equations, literals and clauses in a interpretation is defined as usual.[2]

Interpretations will be described by sets of rewrite rules $l \Rightarrow r$. The interpretation induced by a given set R of rewrite rules is the minimal congruence R^* on ground terms, such that lR^*r for all $l \Rightarrow r \in R$.

Furthermore a fixed total reduction ordering $\succ$ on ground terms will be assumed. This ordering is extended to a total well-founded ordering $\succ_l$ on ground literals as follows: A 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 $m(l) \succ\!\!\succ m(l')$, where $\succ\!\!\succ$ is the multiset extension of $\succ$. It is useful to keep the following properties in mind, which follow immediately from this definition:

If $s \succ t$ and $s' \succ t'$, then

$$\begin{array}{rll} s \doteq t \succ_l s' \doteq t' & \text{iff} & s \succ s' \text{ or } (s = s' \text{ and } t \succ t') \\ s \doteq t \succ_l \neg s' \doteq t' & \text{iff} & s \succ s' \\ \neg s \doteq t \succ_l s' \doteq t' & \text{iff} & s \succeq s' \\ \neg s \doteq t \succ_l \neg s' \doteq t' & \text{iff} & s \succ s' \text{ or } (s = s' \text{ and } t \succ t') \end{array}$$

A *position* is a sequence of numbers designating subterms. $s|_p$ is the subterm of s at position p and $s[r]_p$ is the result of replacing the subterm at position p in s by r.

A constraint is a first order formula that uses a certain fixed signature and is interpreted over the set of ground terms. There are two predicate symbols with fixed interpretation, namely '$\equiv$' representing (syntactic)

[2] This approach is also taken in [13]. Herbrand's Theorem guarantees that for our purposes this is equivalent to defining interpretations with arbitrary carrier sets.

equality and and $\succ$ for the reduction ordering. We denote conjunction as '&' in constraints. A substitution satisfies a constraint, if the constraint is true under the 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.

We shall 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.

There are cases where this restriction can be lifted in tableau calculi, i.e. where it is sound to pick different instantiations for multiple occurrences of a free variable. If that is the cases, one calls the free variable 'universal' for the formula in which it occurs, see [11]. As has been recognized in [5], using universal variables is crucial for efficient equality handling in tableaux. The calculi presented in this paper do *not* use universal variables, but we expect our results to be easily adaptable to calculi that use them.

3 A Simple Calculus

We first introduce a simple calculus and show its completeness. Variations of this calculus will be introduced in the following sections.

3.1 The Calculus

We describe a clausal free variable tableau calculus to refute sets of clauses. Let a set $\mathcal{C}$ of clauses be given. The calculus consists of three rules:

$$\textit{ext} \quad \frac{}{\theta L_1 \quad | \quad \cdots \quad | \quad \theta L_k}$$

where $C = \{L_1, \ldots, L_k\} \in \mathcal{C}$,
and θ renames each variable in C into a new (free) variable.

$$\textit{sup-p} \quad \frac{\begin{array}{c} s \doteq t \ll A \\ l \doteq r \ll B \end{array}}{s[r]_p \doteq t \ll s|_p \equiv l \mathbin{\&} s \succ t \mathbin{\&} l \succ r \mathbin{\&} A \mathbin{\&} B}$$

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

$$\textit{sup-n} \quad \frac{\neg s \doteq t \ll A \qquad l \doteq r \ll B}{\neg s[r]_p \doteq t \ll s|_p \equiv l \mathbin{\&} s \succ t \mathbin{\&} l \succ r \mathbin{\&} A \mathbin{\&} B}$$

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

The superposition rules *sup-p* and *sup-n* are only applied 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,[3] although one might be a renaming of the other.

Note that constraints are attached to the formulae on a branch, and that these constraints are propagated by the rules. When we don't write a constraint (as in the *ext*-rule) we mean the empty constraint that is satisfied by any substitution.

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 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 $s \succ t$. Finally the *basicness restriction* forbids application of equations on subterms created by unifiers introduced by previous superposition steps. In our case, this is achieved by deriving a literal $s[r]_p \doteq t \ll s|_p \equiv l \dots$ instead of determining a most general unifier μ of $s|_p$ and l and generating a literal $\mu(s[r]_p \doteq t)$, as would be done in a calculus without constraints.

In an implementation, closure of a tableau could be determined in different ways:

- 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 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

[3] We can require this because we have *rigid* variables. With rigid variables, a term can't be unified with one of its proper subterms, so superposition would only be possible at the top position, leading to a trivial equation.

branch is continued. If no proof is found up to 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.

- One can avoid the backtracking points introduced by the *sup*-rules by keeping the constraints of literals. These are only added to the global closure constraint G when a branch is closed, and accordingly backtracking is only needed over branch closures.

- One can use the Incremental Closure technique to avoid backtracking completely, see [10].

3.2 Completeness

The completeness proof follows the usual lines: Assuming that there is no closed tableau for a set of clauses, one constructs an infinite tableau by applying rules exhaustively—in particular, the *ext*-rule has to be applied infinitely often for each clause on each branch. Then one chooses a ground substitution σ for the free variables, such that after applying the substitution to the tableau, every branch contains at least one literal from every ground instance of each of the clauses. 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 clause set.

Our proof differs from this standard approach only in the construction of the interpretation and in the proof that the clause set is indeed satisfied by it.

First, we need the following notion:

Definition 1 *Given a set $\mathcal{B}$ of constrained literals, a ground substitution 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 $\mathit{irred}_R(\sigma, \mathcal{B})$, 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 for this will become clear later.

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. The tricky part here is that the rewrite relation that variable-irreducibility refers to is only just being built during the induction.

Definition 2 *Let $\mathcal{B}$ be a set of constrained literals 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 L* generates the rule *$l \Rightarrow r$, iff*

1. $L \in \mathit{irred}_{R_L}(\sigma, \mathcal{B})$,
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 previously generated rules. 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 equations generate rules. When no confusion is possible about the set $\mathcal{B}$ and the substitution σ, we will just write R instead of $R_{\mathcal{B},\sigma}$.

We have the following two useful lemmas, stolen outright from Nieuwenhuis and Rubio:

Lemma 1 *For any set of constrained literals $\mathcal{B}$ and ground substitution σ, the generated set of rules $R = R_{\mathcal{B},\sigma}$ is convergent, i.e. 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 only needs to show local confluence, which follows from the fact that there can be no critical pairs in R. For 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. $l' \Rightarrow r'$ 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 $\mathrm{Gen}(K') = \{l' \Rightarrow r'\}$.

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 $R \setminus R_L$ can not 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 Def. 2, 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 literals 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 $L \in \mathit{irred}_R(\sigma, \mathcal{B})$.

Proof. Assume that this were not the case. Then there must be a *minimal* (w.r.t. $\succ_l$) L in $\mathit{irred}_R(\sigma, \mathcal{B})$ 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 $L \in \mathit{irred}_{R_L}(\sigma, \mathcal{B})$. Also, due to Lemma 2, we already have $R_L^* \not\models L$. But 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 non-variable 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.[4] 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:

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

As $\mathcal{B}$ is required to be closed under 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 $L' \in \mathit{irred}_R(\sigma, \mathcal{B})$. 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 $s \doteq t$, i.e. $R^* \not\models L'$. And finally, by monotonicity of the rewrite ordering $\succ$, $L \succ L'$. This contradicts the assumption that L is the minimal element of $\mathit{irred}_R(\sigma, \mathcal{B})$ 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, $R_L^* \not\models L$, i.e. $R_L^* \models s \doteq t$. 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 literal $K_0 = (l_0 \doteq r_0 \ll B) \in \mathcal{B}$. Again as in case 1, p must be a non-variable position in s_0. It follows that an application of the *sup-n* rule is possible between L_0 and K_0:

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

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 $L \succ L'$, contradicting the assumption that L is minimal in $\mathit{irred}_R(\sigma, \mathcal{B})$ with $R^* \not\models L$. $\square$

[4]This is the place where the use of variable-irreducible instances is necessary. Otherwise, the combination of constraint inheritance and the non-variable-position condition would give problems. Of course, this idea is also stolen from Nieuwenhuis and Rubio.

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 to a global closure test. 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 calculus of Degtyarev and Voronkov – over every application of the superposition rules. In order to state the completeness theorem, we need the following definition of a fair proof procedure.

Definition 3 *A* proof procedure *is a procedure that takes a set of clauses $\mathcal{C}$ and builds a sequence of tableaux $\mathcal{T}_0, \mathcal{T}_1, \mathcal{T}_2, \ldots$ for $\mathcal{C}$ where $\mathcal{T}_0$ is the empty tableau, and each $\mathcal{T}_{i+1}$ results from the application of an ext or sup rule on one of the branches of $\mathcal{T}_i$. A proof procedure* finds a proof for $\mathcal{C}$*, if one of the $\mathcal{T}_i$ is closed. 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,*[5] *then*

- *The ext-rule is applied infinitely often for every clause 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 $\mathcal{C}$ be an unsatisfiable set of clauses. Then a fair proof procedure finds a proof for $\mathcal{C}$.*

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 under any ground substitution for the free variables in $\mathcal{T}$. For assume that under a certain σ all branches are closed. Then there is a literal $\neg s \doteq t$ on every branch with $\sigma s = \sigma t$. 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 $\mathcal{C}$. 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 ground substitution σ. Namely, σ should instantiate the free variables introduced by the *ext*-rule in such a way, that every branch of $\sigma\mathcal{T}$ contains at least one literal of each ground instance of every clause in $\mathcal{C}$. This is possible because by fairness of the procedure, the *ext*-rule is applied infinitely often for each clause on every branch.

[5] The limit of a sequence of tableau is defined as the supremum under the initial subtree ordering.

We have seen that 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 Def. 2 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 every clause in $\mathcal{C}$ is valid in R^* to contradict the assumption that $\mathcal{C}$ is unsatisfiable. We do this by showing that all ground instances of clauses in $\mathcal{C}$ are valid. Let τC be a ground instance of $C \in \mathcal{C}$, where τ is a ground substitution for the variables occurring in C. We now define a new substitution τ' such that $\tau' x$ is the normal form w.r.t. R of τx. This makes $\tau' x$ irreducible by R for all variables x of C. Now $\tau' C$ is obtained from τC by replacement of a number of subterms by other subterms equivalent under R^*. Thus $R^* \models \tau C$ iff $R^* \models \tau' C$. By construction of σ and $\mathcal{B}$, there must be a literal $L \in C$ such that $\theta L \in \mathcal{B}$ for some renaming of variables θ, and such that $\tau' L = \sigma\theta L$. As θL carries no constraint, this makes $\tau' L$ a variable-irreducible instance of θL, so $R^* \models \tau' L$ and accordingly $R^* \models \tau' C$. □

4 A Calculus with Histories

The calculus proposed by Degtyarev and Voronkov has an interesting property that the one described in the previous sections lacks: In their calculus, only a finite number of applications of the superposition rules is possible without intervening applications of the γ-rule, that corresponds to our *ext*-rule. That this is not the case for the calculus considered so far, as can be seen from the following example:

$$\begin{array}{r}
1 : fgx \doteq gx \\
2 : gx \doteq a \\
3(\textit{sup-p} \text{ of } 2 \text{ on } 1) : gx \doteq fa \ll gx \succ a \\
4(\textit{sup-p} \text{ of } 3 \text{ on } 1) : gx \doteq ffa \ll gx \succ fa \\
5(\textit{sup-p} \text{ of } 4 \text{ on } 1) : gx \doteq fffa \ll gx \succ ffa \\
\vdots
\end{array}$$

where the generated constraints have been suitably simplified. Indeed, all the generated constraints can be seen to be satisfiable, regardless of the reduction ordering chosen. Such a derivation is not possible in the calculus of Degtyarev and Voronkov, because they discard the formula that superposition takes place on. The cost of this is that backtracking over the order of applied superposition steps becomes necessary.

Our calculus can be 'fixed' to make superposition steps terminating. We shall see that backtracking is not required in our case.

The idea is as follows: instead of actually discarding rewritten literals, we label each literal L with a *history* h_L, which is a list of (references to occurrences of) literals that would have been discarded during the derivation of L in a destructive calculus. We constrain the superposition rules in a way that excludes rule applications between L and a literal in h_L.

A calculus that discards the rewritten literal would then have a positive superposition rule like this:

$$\frac{\begin{array}{c} L = (s \doteq t \ll A \cdot h_L) \\ K = (l \doteq r \ll B \cdot h_K) \end{array}}{s[r]_p \doteq t \ll s|_p \equiv l \,\&\, s \succ t \,\&\, l \succ r \,\&\, A \,\&\, B \cdot \{L\} \cup h_L \cup h_K}$$

where p is a position in s, $s|_p$ is not a variable and $L \notin h_K$ and $K \notin h_L$.

This would closely correspond to the calculus of Degtyarev and Voronkov, though one can show that it still allows certain derivations not possible in the destructive version. However, we are going to use a more restrictive calculus for the following reasons:

- As we show completeness of a more restrictive calculus, our completeness result is strictly stronger. It works in exactly the same way for a weaker restriction.
- The proof is not further complicated by the stronger restriction.
- The termination property is much easier to show in our calculus.
- In a way, the 'deeper reasons' behind our completeness proof become clearer with the more restrictive calculus.

We shall call the literals introduced by the *ext*-rule (as opposed to those introduced by applications of the superposition rules) *ext-literals*. Our calculus will record in the history of each literal which ext-literals were involved in its derivation. We allow each ext-literal to be used at most once in the derivation of a literal, which is easily formalized by requiring the histories of literals used in the superposition rules to be *disjoint*. In a destructive, backtracking formulation analogous to the one of Degtyarev and Voronkov, this would mean that *both* literals used in a superposition step are discarded.

Here are the three rules of our calculus with histories:

$$ext \quad \frac{}{L_1 \cdot \{L_1\} \quad | \quad \cdots \quad | \quad L_k \cdot \{L_k\}}$$

where $\{L_1, \ldots, L_k\} = \theta C$, with $C \in \mathcal{C}$
and θ renames each variable in C into a new (free) variable.

$$sup\text{-}p \quad \frac{\begin{array}{c} s \doteq t \ll A \cdot h_1 \\ l \doteq r \ll B \cdot h_2 \end{array}}{s[r]_p \doteq t \ll s|_p \equiv l \,\&\, s \succ t \,\&\, l \succ r \,\&\, A \,\&\, B \cdot h_1 \cup h_2}$$

where p is a position in s, $s|_p$ is not a variable and $h_1 \cap h_2 = \emptyset$.

$$\textit{sup-n} \quad \frac{\begin{array}{c}\neg s \doteq t \ll A \cdot h_1 \\ l \doteq r \ll B \cdot h_2\end{array}}{\neg s[r]_p \doteq t \ll s|_p \equiv l \,\&\, s \succ t \,\&\, l \succ r \,\&\, A \,\&\, B \cdot h_1 \cup h_2}$$

where p is a position in s, $s|_p$ is not a variable and $h_1 \cap h_2 = \emptyset$.

Again, the superposition rules *sup-p* and *sup-n* are only applied 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.

As in the simple calculus of the previous section, a ground substitution σ closes a branch $\mathcal{B}$ of a tableau, if there is a constrained negated equation $\neg s \doteq t \ll A \cdot h \in \mathcal{B}$ 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 simultaneously.

4.1 Completeness of the Calculus with Histories

For our completeness proof, we have to modify the proof given in the previous section slightly, to cope with the disjoint history restriction in the *sup*-rules. We require the following notions.

Definition 4 *Let $\mathcal{S}$ be a set of constrained literals with history and σ a ground substitution for the free variables in $\mathcal{S}$. Two literals $L, K \in \mathcal{S}$ are called* variants*, if they are equal up to renaming of free variables, if histories are not regarded.*[6] *They are called* copies *(under σ) if moreover the free variables are assigned the same ground terms under σ. $\mathcal{S}$ is called* rich *(under σ), if every literal $L \in \mathcal{S}$ has an infinite number of copies with* pairwise disjoint histories *in $\mathcal{S}$.*

For instance, $f(X) \doteq Y \ll X \equiv a \cdot \{L_1, L_2\}$ and $f(U) \doteq V \ll U \equiv a \cdot \{L_1, L_3\}$ are variants. They are also copies under σ if $\sigma X = \sigma U$ and $\sigma Y = \sigma V$.

As will become apparent in the proof of Theorem 2, we will do the model construction with only a subset of the literals on an open branch. To avoid confusion, we are going to denote the concerned sets of constrained literals with history $\mathcal{S}$ instead of $\mathcal{B}$.

The construction of a model from a set $\mathcal{S}$ works exactly as in Def. 2. The only new aspect is that the literals in $\mathcal{S}$ have histories: we simply forget those when applying a ground substitution. So $\mathit{irred}_R(\sigma, \mathcal{S})$ shall simply be a set of ground literals without history as before. Thus, for the generated set of ground rewrite rules $R = R_{\mathcal{S},\sigma}$, Lemmas 1 and 2 hold as before.

The differences are in the Model Generation Lemma (Lemma 3 for the simple calculus) and the actual completeness proof. Most of the Model

[6]The variable renaming also applies to the constraints.

Generation Lemma and its proof are actually identical to the simple version, but we shall repeat the proof here to make it more readable and also to make sure that we do not accidentally skip an important difference. The new parts are marked by a gray bar in the margin.

Lemma 4 (Model Generation) *Let $\mathcal{S}$ be a set of constrained literals with history and σ a ground substitution for the free variables in $\mathcal{S}$, such that*

- *$\mathcal{S}$ is closed under the application of the sup-p and sup-n rules, and*
- *there is no literal $\neg s \doteq t \ll A \cdot h \in \mathcal{S}$ such that $\sigma s = \sigma t$ (syntactically) and σ satisfies A.*
- *$\mathcal{S}$ is rich under σ.*

Then $R^* \models L$ for all $L \in \mathit{irred}_R(\sigma, \mathcal{S})$.

Proof. Assume that this were not the case. Then there must be a *minimal* (w.r.t. $\succ_l$) L in $\mathit{irred}_R(\sigma, \mathcal{S})$ 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 $L \in \mathit{irred}_{R_L}(\sigma, \mathcal{S})$. 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 $L_0 = (s_0 \doteq t_0 \ll A \cdot h_L) \in \mathcal{S}$. 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 $K_0 = (l_0 \doteq r_0 \ll B \cdot h_K) \in \mathcal{S}$. As $\mathcal{S}$ is rich, there are infinitely many copies under σ of L_0 with pairwise disjoint histories. Each of the finitely many elements of h_K can be contained in the history of at most one of these copies, and all the remaining ones have a history disjoint to h_K. So we may assume that L_0 and K_0 are chosen in a way that h_K and h_L are disjoint. Further, it turns out that p must be a non-variable 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. From all this, it follows that an application of the *sup-p*-rule between the literals $L_0, K_0 \in \mathcal{S}$ is possible:

$$\textit{sup-p}\quad \frac{\begin{array}{c} s_0 \doteq t_0 \ll A \cdot h_L \\ l_0 \doteq r_0 \ll B \cdot h_K \end{array}}{s_0[r_0]_p \doteq t_0 \ll s_0|_p \equiv l_0 \,\&\, s_0 \succ t_0 \,\&\, l_0 \succ r_0 \,\&\, A \,\&\, B \cdot h_L \cup h_K}$$

As $\mathcal{S}$ is required to be closed under rule applications, the resulting literal, call it L_0', must be in $\mathcal{S}$. 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 $L' \in \mathit{irred}_R(\sigma, \mathcal{S})$. 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 $s \doteq t$, i.e. $R^* \not\models L'$. And finally, by monotonicity of the rewrite ordering $\succ$, $L \succ L'$. This contradicts the assumption that L is the minimal element of $\mathit{irred}_R(\sigma, \mathcal{S})$ 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, $R_L^* \not\models L$, i.e. $R_L^* \models s \doteq t$. 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 $L_0 = (\neg s_0 \doteq t_0 \ll A \cdot h_L) \in \mathcal{S}$ 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 literal $K_0 = (l_0 \doteq r_0 \ll B \cdot h_K) \in \mathcal{S}$. Again as in case 1, p must be a non-variable position in s_0, and we can choose L_0 and K_0 with disjoint histories. It follows that an application of the *sup-n* rule is possible between L_0 and K_0:

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

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

The main point is that if $\mathcal{S}$ is rich, we can find enough copies of the required literals that some of them have disjoint histories. Now in the actual completeness proof, we have to extract a rich set of literals from an open branch in such a way that the validity of the irreducible instances of that set will imply the validity of each of the clauses in our clause set.

Using the definitions of a fair proof procedure from Def. 3, we can now show the following completeness theorem.

Theorem 2 *Let $\mathcal{C}$ be an unsatisfiable set of clauses. Then a fair proof procedure for the calculus with histories finds a proof for $\mathcal{C}$.*

Proof. Assume that the procedure does not find a proof. As in the proof of Theorem 1, we can conclude that it constructs in the limit an infinite tableau $\mathcal{T}$ which has at least one open branch under any ground substitution for the free variables in $\mathcal{T}$.

We now fix the ground substitution σ. Namely, σ should instantiate the free variables introduced by the *ext*-rule in such a way, that every branch of $\sigma\mathcal{T}$ contains *infinitely many* occurrences of literals of each ground instance of every clause in $\mathcal{C}$. This is possible because the *ext*-rule is applied infinitely often for each clause on every branch, and using a dovetailing process that lets each of the ground instantiations be used infinitely often.

There must now be a branch $\mathcal{B}$ of $\mathcal{T}$, such that $\mathcal{B}$ is not closed by σ. As there are infinitely many occurrences of literals of each ground instance of every clause on $\mathcal{B}$, and every clause is finite, for every ground instance τC of every clause, there must be at least one literal $L_{\tau C} \in \tau C$, such that there are infinitely many ext-literals $L' \in \mathcal{B}$ with $\sigma L' = L_{\tau C}$.

Collect all these ext-literals $L_{\tau C}$ on $\mathcal{B}$ in a set I. As we are dealing with ext-literals, the histories of literals in I are disjoint, so I is rich under σ. Now define $\mathcal{B}^\infty$ to contain all literals of I as well as all literals on $\mathcal{B}$ derived from literals in I alone.

As $\mathcal{B}$ is closed under *sup*-rule applications by fairness of the proof procedure, so is $\mathcal{B}^\infty$. Furthermore, $\mathcal{B}^\infty$ is rich, as can be seen by induction on the number n of literals in the history of a given literal L: For $n = 1$, L is an ext-literal, so $L \in I$. Hence there are infinitely many copies of L with pairwise disjoint histories. For $n > 1$, L must be derived by an application of a *sup*-rule from literals with a history smaller than n. The induction hypothesis guarantees an infinite number of copies with pairwise disjoint histories of these literals in $\mathcal{B}^\infty$. The same rule application is obviously possible between these copies, and as $\mathcal{B}^\infty$ is closed under *sup* applications, one easily sees that there must be infinitely many copies of L.

We apply the model generation of Def. 2 on $\mathcal{B}^\infty$ and σ to obtain a set of rewrite rules $R = R_{\mathcal{B}^\infty,\sigma}$. As $\mathcal{B}^\infty$ and σ satisfy the preconditions of Lemma 4, every variable-irreducible instance of $\mathcal{B}^\infty$ is valid in R^*.

It now remains to show that every clause in $\mathcal{C}$ is valid in R^* to contradict the assumption that $\mathcal{C}$ is unsatisfiable. We do this by showing that all ground instances of clauses in $\mathcal{C}$ are valid. Let τC be a ground instance of $C \in \mathcal{C}$, where τ is a ground substitution for the variables occurring in C. We now define a new substitution τ' such that $\tau' x$ is the normal form w.r.t. R of τx. This makes $\tau' x$ irreducible by R for all variables x of C. Now $\tau' C$ is obtained from τC by replacement of a number of subterms by other subterms equivalent under R^*. Thus $R^* \models \tau C$ iff $R^* \models \tau' C$. By construction of σ and $\mathcal{B}^\infty$, there must be a literal $L \in C$ such that $\theta L \in I \subset \mathcal{B}^\infty$ for some renaming of variables θ, and such that $\tau' L = \sigma\theta L$. As θL carries no constraint, this makes $\tau' L$ a variable-irreducible instance of θL, so $R^* \models \tau' L$ and accordingly $R^* \models \tau' C$. $\square$

4.2 Termination

Our termination proof is simpler than the one of Voronkov and Degtyarev because the calculus is more restrictive. Indeed, we can prove termination with the histories alone, without needing arguments about the ordering restrictions expressed in the constraints.

Theorem 3 *Starting from a finite tableau $\mathcal{T}$, only a finite number of sup-rule applications is possible without intervening applications of the ext-rule.*

Proof. As the *sup*-rules do not introduce new branches, it suffices to show this property for each of the finitely many branches of $\mathcal{T}$. The *sup*-rules combine the disjoint history sets of used literals, so the size of the history of the resulting literal is the sum of the sizes of the used literals' histories. In particular, only ext-literals have a history of size one.

We show by induction on n, that only finitely many literals with a history of at most n literals can be derived. For $n = 1$, this is the case, since we start out with only finitely many ext-literals, and we do not get any new ones. For $n > 1$, a literal must be the result of a *sup*-application between literals of history size less than n. By induction hypothesis, there can be only finitely many of those. Also, there are only finitely many ways to apply a *sup*-rule between two given literals, because the rule applications are determined by the position p at which the terms are overlapped.

No history can get larger than the initial number of ext-literals on the branch, so one can only derive a finite number of new literals altogether.

□

Our restriction to disjoint histories is so strong, that it prompts the question whether it is useful in practice. But, of course that question has to be asked of any restriction. Only experimentation can show which restriction is useful to ensure termination in practice. In fact, it is not even clear whether the termination property is of any practical value at all:

- At first sight, the termination property makes it easier to implement a fair proof procedure: One can apply the *sup*-rules exhaustively before resorting to further *ext*-expansions. However, one still needs a fair strategy to choose the next extension clause on a branch. If one can implement an intelligent procedure to do this, one should also be able to choose between extension and superposition. Or, vice versa, if it is sufficient to just put pending *ext*-expansions in a FIFO queue, why should it not be good enough to use the same queue for superposition steps?

- With the termination property, one can take a tableau prover without equality handling and add a 'background prover' that tries to close branches using superposition. The termination property guarantees that the background prover terminates. However, this approach is unpractical as it stands. There is a communication overhead between foreground and background provers. The background prover needs to be incremental, to avoid redoing all the superposition steps after each extension, which is hard to implement.

- In an efficient tableau prover, particularly in the presence of equality, one needs to take universal variables into account, see [5]. The superposition rules with universal variables correspond essentially to unfailing Knuth-Bendix completion [2], which does not terminate in general. UKBA behaves very well in practice, so it is probably not sensible to artificially introduce additional conditions to enforce termination.

- The regularity restriction (see e.g. [11]) requires literals introduced by rule applications to be new to their branches under the closing substitution. This is a very common and successful restriction to eliminate redundancy in proof search. The calculus with histories is not complete if we require regularity, see Sect. 4.3. It is not clear whether the calculus of Degtyarev and Voronkov is compatible with regularity. On the other hand, the simple calculus of Sect. 3 obviously *is*, since only one copy of each ground instance is needed.

To summarize, it seems that in an efficient implementation of a tableau calculus with superposition, the termination property is not really important, and maybe cannot even be sensibly maintained at all.

Of course, the termination property can be bestowed on any calculus by a simple trick: One takes an arbitrary fair strategy and codes it into the calculus. As every possible rule application gets scheduled at some point by a fair procedure, and extension with a clause is always possible, it follows, that only finitely many *sup*-applications are performed in between.

Admittedly, it is nonsense to code the whole proof procedure into the calculus. But only experimentation can show how far one should go.

4.3 Regularity

In this section, we are going to consider the problems with the regularity restriction mentioned in the previous section in a little more detail. The regularity restriction is a common restriction to reduce redundancy in the construction of tableau proofs. In a ground tableau calculus, it requires that no rule application introduces a literal on a branch that is already present on that branch. A version of this condition suitable for our calculi would be that first, closing instantiations σ which lead to two equal literals on a branch under instantiation are not considered, and second, rule applications that lead to a repetition of literals under *every* instantiation are forbidden.

The most important effect of the regularity restriction in a first order calculus is that it prevents expanding the same clause twice with the same instantiation on one branch. We shall call this the 'economic instantiation' property.

One can easily check that the simple calculus of Sect. 3 is compatible with the regularity restriction. Indeed, the completeness proof only requires *one* literal of every ground instance of each clause to be on a branch.

The situation is different for the calculus with histories. Regularity can still be required, if instances of literals with different histories are regarded as different in the regularity condition. But the condition would then be very weak, because the *ext*-rule introduces new literals with new histories each time, so we would not have the economic instantiation property. On the other hand, if we disregard histories in the regularity condition, one can easily see that the calculus is no longer complete. This is reflected by the requirement of having infinitely many copies of literals on a branch in the completeness proof.

So is there a restriction of our calculus which has the termination property and is compatible with a reasonable regularity restriction? The answer is yes, because the simple calculus is compatible with regularity and the trick mentioned at the end of the last section allows us to endow it with the termination property. We thus reformulate our question:

> Is there a *natural* restriction of our calculus which has the termination property and is compatible with a reasonable regularity restriction?

A natural restriction would be one, for instance, that somehow reflects discarding rewritten or otherwise redundant literals. A reasonable regularity restriction should at least entail the economic instantiation property.

It is not stated in [8] whether the calculus of Degtyarev and Voronkov is compatible with regularity. One should remember that their calculus is destructive, so a suitable regularity condition should demand not only that instances of literals present on a branch are not duplicated by rule applications, but also that instances of literals that *were* present but have been discarded are not duplicated. Otherwise we would not get the economic instantiation property.

Regard again the initial attempt at a calculus with histories mentioned at the beginning of Sect. 4 on page 12. We tried to find a completeness proof for that calculus (which corresponds closely to the one of [8]) that would be compatible with regularity. In particular, we considered a ground version of that calculus, that does not need constraints and works only on ground literals with histories. We were able to show completeness of that ground calculus by the model generation technique without requiring infinitely many copies of literals. The variable irreducibility restriction is not needed for the ground case, instead we restrict model generation to literals with maximal histories. From the validity of these in the generated model, we can then inductively infer the validity of all other literals. The restriction to literals with maximal history conflicts with the restriction to variable-irreducible instances which we need to cope with the non-variable-position restriction, leading to very tangled interdependencies which we were not able to resolve.

We thus currently have no answer to the question formulated above.

5 Using Negative Constraints

In Sect. 4, we used history lists to emulate the deletion of literals from a branch in a non-destructive way that eliminates the need to backtrack over *sup*-rule applications. Another possibility to model deletion of literals comes to mind, which we shall briefly discuss.

Assume that the following rule application is possible between two literals:

$$\textit{sup-p} \quad \frac{\begin{array}{c} s \doteq t \ll A \\ l \doteq r \ll B \end{array}}{s[r]_p \doteq t \ll A \,\&\, B \,\&\, C}$$

Where $C = (s|_p \equiv l \,\&\, s \succ t \,\&\, l \succ r)$ is the new constraint introduced by the rule application. One could now model the deletion of the rewritten literal $s \doteq t \ll A$ by *modifying* the constraint of that literal to $A \,\&\, !(B \,\&\, C)$:[7] this implies that for instantiations σ under which the superposition is possible, namely if σ satisfies $A \,\&\, B \,\&\, C$, the constraint of the deleted literal is no longer valid.

This method leads to a destructive calculus, since the constraints of literals are changed as the proof is expanded. Accordingly, more complicated techniques are required for a completeness proof, one needs to consider the persistent literals on a branch, etc.

Also, one needs a rather complicated fairness condition: It is not sufficient to require (as in resolution saturation procedures) that all superpositions between persistent literals of a branch are eventually executed: Assume for instance that a superposition with K is applicable on some literal L, and that this superposition is needed to close the proof. There are situations where it is possible to successively apply superposition steps on L with other literals leading to a sequence of constrained literals $L', L'', \ldots$ with constraints getting more and more restrictive, but without becoming unsatisfiable. Neither the literal L or any of its descendants is then persistent on the branch, so superposition with K might never take place.

[7]We use '!' to designate negation in constraints.

We have tried to fix the fairness condition (or rather the notion of persistency) to take account of this difficulty. But we still did not succeed in showing completeness of the resulting procedure. To give an intuition for the problem, let us mention that one has to restrict model generation to variable-irreducible literals in order to cope with the non-variable-position restriction, while literals on the branch might be rewritten by superposition with literals which later turn out not to be variable irreducible.

This state of affairs is rather unsatisfying. One can construct a *ground* calculus in which the *ext* rule introduces (guessed) ground instances of clauses instead of free variables, no constraints are needed, and the first premise of the superposition rules is actually deleted. We have been able to show completeness (without backtracking over superposition applications) of this destructive ground calculus. The proof even shows compatibility with regularity. But we have not been able to lift this proof due to the afore-mentioned problems with the non-variable-position restriction. The situation is thus rather similar to the one described in Sect. 4.3.

6 Tableaux with Basic Ordered Paramodulation

In this section, we shall try to demonstrate how variations of calculi and completeness proofs can be carried over from known results for resolution-based calculi.

There is a more restrictive form of equality handling known in the resolution community as *basic ordered paramodulation* [3]. 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 very effective in practice [12].

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:[8]

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

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

[8]We do not use the disjoint history restriction here in order to make things simpler to read. It should however be no problem to use that restriction with basic ordered paramodulation.

Note how the constraint forces X to be instantiated with r, and that the restriction $s \succ t$ is gone.[9] The *par-n*-rule is exactly analogous. This modification is a straightforward adaptation of the formulation of basic ordered paramodulation using constraint inheritance given by Nieuwenhuis and Rubio in [13].

How do we show completeness of our modified calculus? We cite [13]:

> 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.

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.

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: For case 1, one drops the assumption that $s \succ t$, and infers 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 *sup-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 this corresponds 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 constitutes strong evidence for our claim that our completeness proofs are a good basis for adapting known results from resolution with superposition or paramodulation to a tableau setting.

[9] It might seem that introducing a new free variable is not a good idea. But these ones are harmless, as there is no need to search for their instantiation. It is determined by the instantiations of the free variables in r. In a sense, they can be regarded as universal variables restricted by the constraint $X \equiv r$.

7 Conclusion and Future Research

We presented a number of free variable tableau calculi with integrated equality handling using ordered superposition/paramodulation rules with constraint propagation. We demonstrated how the completeness of such calculi can be shown using model generation techniques known from resolution calculi with only few additional tableau-specific ingredients.

We have shown how a termination property can be enforced for such calculi using a disjoint history restriction, and how completeness may be proved in presence of such a restriction. We have also briefly discussed the practical usefulness of the termination property in such calculi.

One area for future research is experimentation with an implementation. In particular, it would be interesting to see what impact various restrictions ensuring the termination property have both on performance of the prover and on implementation complexity.

An obvious extension of our results would be a version that permits predicates other than equality and that does not require problems to be in clausal form. We expect this to be quite straight-forward.

Universal variables are known to be important for efficiency. It is expected that the given calculi and proofs can easily be adapted to incorporate universal variables, but of course this has to be checked in detail. We also plan to investigate how superposition-based equality handling can be incorporated into hyper tableaux [4].

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 for resolution, see e.g. [14].

Finally, it would be (at least theoretically) interesting to find an answer to the question of Sect. 4.3, namely whether there is a natural restriction that ensures the termination property but is compatible with regularity.

Acknowledgments

I thank Bernhard Beckert, Reiner Hähnle and Peter Schmitt for many fruitful discussions which led to the results presented in this paper.

References

[1] L. Bachmair and H. Ganzinger. Resolution theorem proving. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 2, pages 19–99. Elsevier Science, 2001.

[2] Leo Bachmair, Nachum Dershowitz, and David A. Plaisted. Completion without failure. In H. Aït-Kaci and M. Nivat, editors, *Resolution*

of Equations in Algebraic Structures, volume 2: Rewriting Techniques, pages 1–30. Academic Press, New York, 1989.

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

[4] Peter Baumgartner. Hyper Tableaux — The Next Generation. In Harrie de Swart, editor, *Proc. International Conference on Automated Reasoning with Analytic Tableaux and Related Methods, Oosterwijk, The Netherlands*, number 1397 in LNCS, pages 60–76. Springer-Verlag, 1998.

[5] Bernhard Beckert. A completion-based method for mixed universal and rigid E-unification. In Alan Bundy, editor, *Proc. 12th Conference on Automated Deduction CADE, Nancy/France*, LNAI 814, pages 678–692. Springer-Verlag, 1994.

[6] A. Degtyarev and A. Voronkov. The undecidability of simultaneous rigid E-unification. *Theoretical Computer Science*, 166(1-2):291–300, October 1996.

[7] A. Degtyarev and A. Voronkov. What you always wanted to know about rigid E-unification. Technical Report 143, Comp. Science Dept., Uppsala University, 1997.

[8] A. Degtyarev and A. Voronkov. What you always wanted to know about rigid E-unification. *Journal of Automated Reasoning*, 20(1):47–80, 1998.

[9] J. Gallier, S. Raatz, and W. Snyder. Theorem proving using rigid E-unification: Equational matings. In *Proc. IEEE Symp. on Logic in Computer Science*, pages 338–346. IEEE Computer Society Press, 1987.

[10] Martin Giese. Incremental Closure of Free Variable Tableaux. In Rajeev Goré, Alexander Leitsch, and Tobias Nipkow, editors, *Proc. Intl. Joint Conf. on Automated Reasoning, Siena, Italy*, LNCS. Springer-Verlag, 2001.

[11] R. Hähnle. Tableaux and related methods. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 3, pages 100–178. Elsevier Science, 2001.

[12] William McCune. Solution of the robbins problem. *Journal of Automated Reasoning*, 19(3):263–276, December 1997.

[13] R. Nieuwenhuis and A. Rubio. Paramodulation-based theorem proving. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 7, pages 371–443. Elsevier Science, 2001.

[14] Robert Nieuwenhuis and Albert Rubio. Paramodulation with built-in AC-theories and symbolic constraints. *Journal of Symbolic Computation*, 23(1):1–21, January 1997.