

A Model Generation Style Completeness Proof for Constraint Tableaux with Superposition

Martin Giese

Institut für Logik, Komplexität und Deduktionssysteme,
Universität Karlsruhe, Germany
giese@ira.uka.de

Abstract. We present a calculus that integrates equality handling by superposition into a free variable tableau calculus. We prove completeness of this calculus by an adaptation of the *model generation* [1, 15] technique commonly used for completeness proofs of resolution calculi. The calculi and the completeness proof are compared to earlier results of Degtyarev and Voronkov [7].

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. [12]), or hyper tableaux [4] with equality. Curiously enough, this has not happened! We surmise that

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

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 [7], 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, 15]. 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. Some parts of this paper are elaborated in more detail in [11].

In Sect. 2 we review some common notions and notations. Our calculus is described in Sect. 3. The main result, namely completeness, is presented in Sect. 4. A discussion of a certain termination property and related issues follows in Sect. 5. We then 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 $m(l) \succ m(l')$, where \succ is the multiset extension of \succ . It is useful to keep the following properties in mind, which follow immediately from this definition:

² This approach is also taken in [15]. Herbrand’s Theorem guarantees that for our purposes this is equivalent to defining interpretations with arbitrary carrier sets.

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

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

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) equality 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.

A *constrained literal* is a pair $(\neg)s \doteq t \ll C$ of a literal and a constraint. The intention is that the literal may only be used if the instantiations of the free variables of a tableau satisfies the constraint.

We shall occasionally refer to *rigid* versus *universal* variables. Rigid variables are the free variables introduced by the rule for universal quantifiers 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 instantiate different occurrences of a free variable differently. If that is the case, one calls the free variable ‘universal’ for the formula in which it occurs, see [12]. 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 The Calculus

We shall describe a clausal free variable tableau calculus to refute sets of clauses. In such a calculus, there is of course the usual clause extension rule, which expands the tableau by renamings of the literals of one clause. In addition, we want rules to perform superposition between literals on a branch. In principle, e.g. in a ground calculus, the rule should look like this:

$$\frac{(\neg)s \doteq t \quad l \doteq r}{(\neg)s[r]_p \doteq t}$$

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,³ 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. In order to avoid this, we shall 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. We choose a non-destructive formulation to avoid backtracking: instead of actually deleting literals, we keep track with each new literal L of a set of literals that *would have been deleted* during the derivation of L in a destructive calculus. These sets, called *histories* can then be used to exclude rule applications, e.g. between L and K if L is in the history of K . We shall write a constrained literal with history as $(\neg)s \doteq t \ll C \cdot h$.

In fact, we go a step further: at each superposition step we delete (or rather simulate deletion of) not only the first premise but both premises, because it turns out that this stronger restriction does not complicate the completeness proof—in a sense, it even becomes more transparent—and it makes it easier to prove the termination property of Sect. 5. Deleting both premises amounts to requiring each literal to be used at most once in the derivation of any other literal. Call the literals introduced on the tableau by the extension rule and not by superposition *ext-literals*. In the history of a literal L , we shall record the ext-literals from which L is derived, and in superposition steps, we shall require these histories to be disjoint.

Formally, we start from an initial empty tableau and expand it by applying certain rules. Tableau nodes are labeled with *constrained literals with histories*, written $L = (\neg)s \doteq t \ll C \cdot h$. The history h is a set of (references to occurrences of) literals on the tableau. Let a set \mathcal{C} of clauses be given. Our calculus has the following three rules:

$$\begin{array}{c} \text{ext} \quad \frac{}{L_1 \cdot \{L_1\} \quad | \quad \dots \quad | \quad L_k \cdot \{L_k\}} \\ \text{where } \{L_1, \dots, L_k\} = \theta C, \text{ with } C \in \mathcal{C} \\ \text{and } \theta \text{ renames each variable in } C \text{ into a new (free) variable.} \end{array}$$

$$\begin{array}{c} \text{sup-}p \quad \frac{s \doteq t \ll A \cdot h_1 \quad l \doteq r \ll B \cdot h_2}{s[r]_p \doteq t \ll s|_p \equiv l \ \& \ s \succ t \ \& \ l \succ r \ \& \ A \ \& \ B \cdot h_1 \cup h_2} \\ \text{where } p \text{ is a position in } s, s|_p \text{ is not a variable and } h_1 \cap h_2 = \emptyset. \end{array}$$

³ From now on, we always refer to the ‘tableau basic superposition’ calculus *TBSE* of [7].

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

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

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,⁴ although one might be a renaming of the other.

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.

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 *basic strategy* is a restriction that was first introduced for calculi that work without constraints. In a superposition step of such calculi, a most general unifier μ of $s|_p$ and l is determined, and a literal $\mu(s[r]_p \doteq t)$ is generated. 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.

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 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

⁴ 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.

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 also drop the histories and discard literals used in a *sup*-rule application from the branch instead. Of course, when the procedure backtracks, they have to be reintroduced.

- 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].

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

4 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 a sufficient set of ground instances of literals from 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 $\text{irred}_R(\sigma, \mathcal{B})$, 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 $\text{Gen}(L) = \{l \Rightarrow r\}$ and say L generates the rule $l \Rightarrow r$, iff

1. $L \in \text{irred}_{R_L}(\sigma, \mathcal{B})$,
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 $R_L := \bigcup_{L \succ_l K} \text{Gen}(K)$ is the set of all previously generated rules. Otherwise, we define $\text{Gen}(L) := \emptyset$. The set of all rules generated by any ground literal is denoted $R_{\mathcal{B}, \sigma} := \bigcup_K \text{Gen}(K)$.

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 will need the following two useful lemmas, taken from Nieuwenhuis and Rubio [15]:

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 $\text{Gen}(K') = \{l' \Rightarrow r'\}$.

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. \square

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, $R^* \models \neg s \doteq t$. \square

Definition 3. 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.⁵ 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 $\sigma Y = \sigma V$.

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{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 $L \in \text{irred}_R(\sigma, \mathcal{S})$.

Proof. Assume that this were not the case. Then there must be a *minimal* (w.r.t. \succ_l) L in $\text{irred}_R(\sigma, \mathcal{S})$ 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 $L \in \text{irred}_{R_L}(\sigma, \mathcal{S})$. Also, due to Lemma 2, we already have $R_L^* \not\models L$. But $\text{Gen}(L) = \emptyset$, 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

⁵ The variable renaming also applies to the constraints.

between the literals $L_0, K_0 \in \mathcal{S}$ is possible:

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

As \mathcal{S} is required to be closed under rule applications, the resulting literal, call it L'_0 , must be in \mathcal{S} . Now $L' := (s[r]_p \dot{=} t) = \sigma L'_0$ 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 $L' \in \text{irred}_R(\sigma, \mathcal{S})$. 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 $s \dot{=} t$, 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 $\text{irred}_R(\sigma, \mathcal{S})$ which is not valid in R^* .

Case 2: $L = (\neg s \dot{=} t)$. 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. $R_L^* \models s \dot{=} t$. According to Lemma 1, R_L is convergent. Validity of $s \dot{=} t$ 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 $L_0 = (\neg s_0 \dot{=} t_0 \ll A \cdot h_L) \in \mathcal{S}$ and let $l \Rightarrow r$ be generated by a literal $K = (l \dot{=} r) \prec_l L$ that is the variable-irreducible (w.r.t. R_K) instance of a constrained literal $K_0 = (l_0 \dot{=} r_0 \ll B \cdot h_K) \in \mathcal{S}$. 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 :

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

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

We cope with the history restriction here by requiring that \mathcal{S} is rich, so 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.

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 \mathcal{TBSE} 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 4. A proof procedure is a procedure that takes a set of clauses \mathcal{C} and builds a sequence of tableaux $\mathcal{T}_0, \mathcal{T}_1, \mathcal{T}_2, \dots$ 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 in any run, it either finds a proof, or the following holds for the limit \mathcal{T} of the sequence of constructed tableaux:⁶

- The *ext*-rule is applied infinitely often for each clause on every branch of \mathcal{T} .
- Every possible application of the *sup*-rules between two literals on a branch of \mathcal{T} has been performed on that branch.

Theorem 1. 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. Then it constructs a sequence of tableaux $\mathcal{T}_0, \mathcal{T}_1, \mathcal{T}_2, \dots$ 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 *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 \mathcal{I} . As we are dealing with ext-literals, the histories of literals in \mathcal{I} are disjoint, so \mathcal{I} is rich under σ . Now

⁶ The limit of a sequence of tableaux is defined as the supremum under the initial subtree ordering.

define \mathcal{B}^∞ to contain all literals of \mathcal{I} as well as all literals on \mathcal{B} derived from literals in \mathcal{I} alone.

As \mathcal{B} is closed under *sup*-rule applications by fairness of the proof procedure, so is \mathcal{B}^∞ . Furthermore, \mathcal{B}^∞ 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 $L \in \mathcal{I}$. 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}^∞ . The same rule application is obviously possible between these copies, and as \mathcal{B}^∞ 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}^∞ and σ to obtain a set of rewrite rules $R = R_{\mathcal{B}^\infty, \sigma}$. As \mathcal{B}^∞ and σ 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 $\theta L \in \mathcal{I} \subset \mathcal{B}^\infty$ 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

5 Termination and Regularity

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 2. *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. \square

The flip side of enforcing the termination property using histories is that we lose regularity (see e.g. [12]): It is clear from our completeness proof that an arbitrary number of ext-literals with the same instantiation, though with different histories, might be needed.

The situation changes if one discards histories from our calculus. One then loses the termination property, but the calculus becomes compatible with regularity. Indeed, the completeness proof then works without the richness condition, so one literal of every instantiation of every clause on each branch is sufficient.

The question arises whether this happens if one deletes (or uses histories to simulate deletion of) only the first premise of the *sup*-rules, as is done in the calculus of Degtyarev and Voronkov. There is no indication in [7] whether their calculus and completeness proof are compatible with regularity. We have not been able to find a natural restriction of our calculus that enjoys the termination property *and* is compatible with regularity, see [11] for a discussion of the difficulties.

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?
- 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.
- As noted above, it is not clear whether there is a natural restriction that ensures termination and is compatible with regularity. But regularity is acknowledged to be an important refinement for automated theorem provers.

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.

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 at or 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 [13].

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:⁷

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

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.⁸ 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 [15].

How do we show completeness of our modified calculus? We cite [15]:

⁷ We do not use the disjoint history restriction here in order to make things simpler to read. It is no problem to use that restriction with basic ordered paramodulation.

⁸ 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 $X \equiv r$.

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.

7 Conclusion and Future Research

We presented a free variable tableau calculus 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 few additional tableau-specific ingredients. Though completeness of a similar calculus has previously been proven in [7], our proof is much shorter, and we have demonstrated that it is easily adapted to related calculi.

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. 5, namely whether there is a natural restriction that ensures the termination property but is compatible with regularity.

Acknowledgements

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 Robinson and Voronkov [16], chapter 2, pages 19–99.
- [2] Leo Bachmair, Nachum Dershowitz, and David A. Plaisted. Completion without failure. In H. Ait-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] Martin Giese. Model generation style completeness proofs for constraint tableaux with superposition. Technical Report 2001-20, Universität Karlsruhe TH, Germany, 2001. URL: <http://i11www.ira.uka.de/~giese/tr01-20.ps.gz>.
- [12] R. Hähnle. Tableaux and related methods. In Robinson and Voronkov [16], chapter 3, pages 100–178.
- [13] William McCune. Solution of the Robbins problem. *Journal of Automated Reasoning*, 19(3):263–276, December 1997.
- [14] R. Nieuwenhuis and A. Rubio. Paramodulation with built-in AC-theories and symbolic constraints. *Journal of Symbolic Computation*, 23(1):1–21, January 1997.
- [15] R. Nieuwenhuis and A. Rubio. Paramodulation-based theorem proving. In Robinson and Voronkov [16], chapter 7, pages 371–443.
- [16] Alan Robinson and Andrei Voronkov, editors. *Handbook of Automated Reasoning*. Elsevier Science B.V., 2001.