Hyper Tableaux with Equality

Peter Baumgartner[1], Ulrich Furbach[2], and Björn Pelzer[2]

[1] NICTA, Canberra, Australia
Peter.Baumgartner@nicta.com.au
[2] Universität Koblenz-Landau, Koblenz, Germany
{uli,bpelzer}@uni-koblenz.de

Abstract. In most theorem proving applications, a proper treatment of equational theories or equality is mandatory. In this paper we show how to integrate a modern treatment of equality in the hyper tableau calculus. It is based on splitting of positive clauses and an adapted version of the superposition inference rule, where equations used for paramodulation are drawn (only) from a set of positive unit clauses, the candidate model. The calculus also features a generic, semantically justified simplification rule which covers many redundancy elimination techniques known from superposition theorem proving. Our main results are soundness and completeness, but we briefly describe the implementation, too.

1 Introduction

Tableau calculi play an important role in theorem proving, knowledge representation and in logic programming. Yet, for automated first-order theorem proving the influence of tableau calculi decreased in the last decade. The CASC competition [SS06] is dominated by saturation-based provers, and a tableau system like SETHEO, which was several times among CASC winners, is not even entering the competition any more. Among the reasons are the problems tableau calculi have with efficient handling of equality. Of course there are numerous papers on equality handling in tableau calculi. Various approaches have been discussed, for instance, in [Bec97]. It is not clear, however, whether they can be a basis for high performance theorem proving. This has to do with the usage of free variables in most semantic tableau calculi. The nature of these free variables, their rigidness, seems to be a major source for difficulties to define efficient proof procedures, even without equality. For instance, proof procedures often suffer from excessive backtracking and enumerate whole tableaux in an iterative-deepening fashion, typically based on the number of γ-rule applications in a tableau.

To avoid the problems of rigid variables for equality reasoning, in [DV96] the authors combine a superposition based equality reasoning system with a top down semantic tableau reasoner. Yet, certain substitutions still have to be applied globally to all variables in the tableau, which thus are still treated rigidly. As with most free-variable tableau calculi, the important property of *proof confluence* does not hold or is not known to hold.

Other free-variable tableau methods are based on solving (simultaneous) rigid E-unifiability problems [DV98] but still face the same problem of not exploiting proof confluence.

F. Pfenning (Ed.): CADE 2007, LNAI 4603, pp. 492–507, 2007.

A more recent stream of equality handling in free-variable tableaux has been initiated by Martin Giese. It is (also) motivated by addressing the excessive backtracking of the methods mentioned above. In [Gie02] the author gives a calculus for free variable tableaux with superposition-type inference and proves completeness by adapting the model generation technique for superposition [BG98,NR01]. One improvement, compared with [DV96] and other free-variable methods is that unification constraints leading to a closed tableau are now held locally together with tableau literals. This allows one to avoid backtracking over the tableaux generated in a derivation, but instead amounts to combining local substitutions in a compatible way for the purpose to witness a closed tableau (see [Gie01] for details). A drawback of this approach is its potentially high memory consumption, as, in essence, it does not admit a one-branch-at-a-time proof procedure.

In [Gie03], simplification rules and reasoning with universal variables[1] are added to the framework of [Gie02], *but without equality.* Equality aside, the most relevant contribution in [Gie03] from the viewpoint of this paper is the instantiation of the calculus there to a variant of the hyper tableau calculus [BFN96].[2] An important difference to [BFN96] is that [Gie03] uses rigid variables for variables that are shared between positive literals in clauses. For instance, a clause like $\forall x, y\ (p(x, y) \vee q(x))$ then is treated by β-expansion with the formulas $\forall y\ p(X, y)$ and $q(X)$, where X is a rigid variable shared between branches. In contrast, the hyper tableaux of [BFN96] would branch out on the formulas $\forall y\ p(t, y)$ and $q(t)$, where t is some "guessed" ground term of the input signature.[3]

In this paper we stick with the hyper tableau calculus and its "obviously inefficient" approach of guessing ground terms for shared variables, as opposed to using free variables. More precisely, we show how to incorporate efficient ordering-based equality inference rules and redundancy elimination techniques from the superposition calculus [BG98, NR01] into a tableau calculus. We believe the hyper tableau calculus [BFN96] is a good basis for doing that, for the following reasons.

- All variables in a hyper tableau are universally quantified in the branch literal they occur. This facilitates the adaption of the superposition framework and enables powerful redundancy criteria.
- As far as we know, none of the free-variable calculi mentioned above can be used as a non-trivial decision procedure for function-free clause logic. The same holds true for any known resolution refinement.
  On the other hand, our calculus is a non-trivial decision procedure for this fragment (with equality), which captures the complexity class NEXPTIME.

[1] Variables that are local to a clause or literal and that are universally quantified.

[2] Hyper tableaux is a tableau model generation method, which is applied to clauses and needs only one inference rule, which can be seen as a tableaux β-rule. It is applied in a "hyper-way", such that all negative literals are "resolved away" by positive literals in the branch. The remaining literals are positive and are split after that. This basic idea stems from SATCHMO [MB88], which is extended in hyper tableaux by making better use of universally quantified variables.

[3] Notice that Resolution- or Superposition calculi, also those with Splitting [Wei01], do not split $\forall x, y\ (p(x, y) \vee q(x))$.

Many practically relevant problems are NEXPTIME-complete, e.g. first-order model expansion (relevant for constraint solving).
- Advanced techniques are available to restrict the domain of the guessed ground terms (like t above). For instance, the preprocessing technique in [BS06] can readily be used in conjunction with our calculus without any change.[4]
- Specific to the theory of equality and in presence of simplification inference rules, that domain can even be further reduced. This occasionally shows unexpected (positive) effects, leading to termination of our system, where e.g. superposition based systems do not terminate. See Section 5 for details.
- The hyper tableau calculus is the basis of the KRHyper prover, which is used in various applications [FO06, **?**, BFGHS04, e.g.] from which we learned that an efficient handling of equality would increase its usability even more.

The closest approximation of the superposition calculus to E-hyper tableaux is obtained by using a selection function that selects all negative literals and using a prover that supports splitting, like SPASS [Wei01]. Even then, there remain differences. We discuss these issues in Section 5.

In [BT05], the model evolution calculus is extended by equality. Model evolution is a lifting of propositional DPLL to the first order case. The model construction method behind admits semantically justified redundancy elimination criteria. This calculus, as well as other instance-based methods (with equality, like [LS02]) are conceptually rather different to resolution- or tableau calculi as considered here.

This paper is organised as follows: we start with preliminaries in the following section. In Section 3 we present superposition inference rules for clauses together with a static completeness result. In Section 4 we introduce E-hyper tableaux and soundness and completeness properties. In Section 5 we consider improvements for splitting and discuss the relation with splitting in the SPASS prover. Section 6 describes the implementation of the E-KRHyper system. Detailed proofs of all results can be found in the long version of this paper.

2 Preliminaries

Most of the notions and notation we use in this paper are the standard ones in the field. We report here only notable differences and additions.

We will use an infinite set of variables X, and x and y denote elements of X. We fix a signature Σ throughout the paper. Unless otherwise specified, when we say term we will mean Σ-term. If t is a term we denote by $\mathcal{V}\mathrm{ar}(t)$ the set of t's variables. A term t is *ground* iff $\mathcal{V}\mathrm{ar}(t) = \emptyset$.

The notation $s[t]_p$ denotes the replacement of a subterm of s at position p with a term t, as usual. We leave away the subscript p if clear from the context. All of the above is extended from terms to literals in the obvious way.

[4] For example, the calculus described here does not admit a finite (fair) derivation from the clause set $\{\forall x\ p(x) \vee q(x), r(f(c))\}$, but in conjunction with the techniques in [BS06] it does.

In this paper we restrict ourselves to equational clause logic. Therefore, and essentially without loss of generality, we assume that the only predicate symbol in Σ is $\simeq$. Any atom A that is originally not an equation can be represented as the equation $A \simeq \mathbf{t}$, where $\mathbf{t}$ is some distinguished constant not appearing elsewhere. (But we continue to write, say, $P(a)$ instead of the official $P(a) \simeq \mathbf{t}$.) This move is harmless, in particular from an operational point of view.[5] An atom then is always an equation, and a literal then is always an equation or the negation of an equation. Literals of the latter kind, i.e., literals of the form $(s \simeq t)$ are also called *negative equations* and generally written $s \not\simeq t$ instead. We call a literal *trivial* if it is of the form $t \simeq t$ or $t \not\simeq t$.

We denote atoms by the letters A and B, literals by the letters K and L and by $\overline{L}$ the complement of a literal L.

A clause is a finite multiset of literals, written as a disjunction $A_1 \vee \cdots \vee A_m \vee \neg B_1 \vee \cdots \vee \neg B_n$ or an implication $A_1, \ldots, A_m \leftarrow B_1, \ldots, B_n$, where $m, n \geq 0$. Each atom A_i, for $i = 1, \ldots, m$, is called a *head atom*, and each atom B_j, for $j = 1, \ldots, n$, is called a *body atom*. We write $A, \mathcal{A} \leftarrow B, \mathcal{B}$ to denote a clause with head atoms $\{A\} \cup \mathcal{A}$ and body atoms $\{B\} \cup \mathcal{B}$, where $\mathcal{A}$ and $\mathcal{B}$ are multisets of atoms. As usual, clauses are implicitly universally quantified.

We suppose as given a reduction ordering $\succ$ that is total on ground Σ-terms.[6] The non-strict ordering induced by $\succ$ is denoted by $\succeq$, and $\prec$ and $\preceq$ denote the converse of $\succ$ and $\succeq$. The reduction ordering $\succ$ has to be extended to rewrite rules, equations and clauses. Following usual techniques [BG98, NR01, e.g.], to a given ground clause $\mathcal{A} \leftarrow \mathcal{B}$ we associate to each head atom $s \simeq t$ in $\mathcal{A}$ the multiset $\{s, t\}$ and to each body atom $u \simeq v$ in $\mathcal{B}$ the multiset $\{u, u, v, v\}$. Two atoms then (head or body) are compared by using the multiset extension of $\succ$, which is also denoted by $\succ$. This will have the effect of a lexicographic ordering, where, first, the bigger terms of two equations are compared, then the sign (body atoms are bigger) and at last the smaller sides of the equations. To compare clauses the two-fold multiset extension of $\succ$ is used, likewise denoted by $\succ$. When comparing ground rewrite rules they are treated as unit clauses.

A central notion for hyper tableaux is that of a *pure* clause [BFN96]: a clause $A_1, \ldots, A_m \leftarrow B_1, \ldots, B_n$ is called *pure* iff $\mathcal{V}\mathrm{ar}(A_i) \cap \mathcal{V}\mathrm{ar}(A_j) = \emptyset$, for all $1 \leq i, j \leq m$ with $i \neq j$. That is, in a pure clause variables are not shared among head literals. (In the rest of this paper we will need this concept for positive clauses only.) Any substitution that turns a clause C into a pure instance $C\pi$ is called a *purifying substitution (for C)*.

A *(Herbrand) interpretation* I is a set of ground Σ-equations—those that are true in the interpretation. Satisfiability/validity of ground Σ-literals, Σ-clauses,

[5] Strictly speaking, one has to move to a two-sorted signature with different signatures for function symbols and predicate symbols, and all variables are of the sort of terms. We ignore this aspect throughout the paper because it does not cause any complications.

[6] A *reduction ordering* is a strict partial ordering that is well-founded and is closed unter context i.e., $s \succ s'$ implies $t[s] \succ t[s']$ for all terms t, and liftable, i.e., $s \succ t$ implies $s\delta \succ t\delta$ for every term s and t and substitution δ.

and clause sets in a Herbrand interpretation is defined as usual. We write $I \models F$ to denote that I satisfies F, where F is a ground Σ-literal or a Σ-clause (set).

Since every interpretation defines in effect a binary relation on ground Σ-terms, and every binary relation on such terms defines an interpretation, we will identify the two notions in the sequel.

An *E-interpretation* is an interpretation that is also a congruence relation on the Σ-terms. If I is an interpretation, we denote by I^{E} the smallest congruence relation on the Σ-terms that includes I. We say that I *E-satisfies* F iff $I^{\mathrm{E}} \models F$. Instead of $I^{\mathrm{E}} \models F$ we generally write $I \models_{\mathrm{E}} F$. We say that F *E-entails* F', written $F \models_{\mathrm{E}} F'$, iff every E-interpretation that satisfies F also satisfies F'. We say that F and F' are *E-equivalent* iff $F \models_{\mathrm{E}} F'$ and $F' \models_{\mathrm{E}} F$.

Redundant Clauses. Intuitively, a clause is redundant iff it follows from a set of smaller clauses. We will formalize this now, following [BG98]. There is a related notion of "redundant inference" which will be introduced in Section 3.1 below.

If D is a ground clause and $\mathcal{C}$ is a set of ground clauses then let $\mathcal{C}_D = \{C \in \mathcal{C} \mid D \succ C\}$. When $\mathcal{C}$ is a set of non-ground clauses and when writing $\mathcal{C}_D$ we identify $\mathcal{C}$ with the set of all ground instances of all its clauses.

Now, a ground clause D is *redundant wrt. a set of clauses* $\mathcal{C}$ iff $\mathcal{C}_D \models_{\mathrm{E}} D$. That is, D is redundant wrt. $\mathcal{C}$ iff D follows from smaller clauses taken from $\mathcal{C}$.[7] When D is a non-ground clause we say that D is redundant wrt. $\mathcal{C}$ iff every ground instance of D is redundant wrt. $\mathcal{C}$. For instance, using any simplification ordering, $P(f(a)) \leftarrow$ is redundant wrt. $\{P(a) \leftarrow, f(x) \simeq x \leftarrow\}$, because $\{P(a) \leftarrow, f(a) \simeq a \leftarrow\} \models_{\mathrm{E}} P(f(a)) \leftarrow$ and each clause in the premise is smaller than $P(f(a)) \leftarrow$.

3 Inference Rules on Clauses

The following three inference rules are taken from the superposition calculus [BG98] and adapted to our needs. We need in addition a splitting rule that will be defined afterwards. All rules will later be embedded into the hyper tableau derivation rules.

An equation $l \simeq r$ always also denotes its symmetric version $r \simeq l$.

The sup-left rule (*superposition left*[8]) applies a superposition step to a body literal:

$$\mathsf{sup\text{-}left}(\sigma)\ \frac{\mathcal{A} \leftarrow s[l'] \simeq t, \mathcal{B} \qquad l \simeq r \leftarrow}{(\mathcal{A} \leftarrow s[r] \simeq t, \mathcal{B})\sigma} \quad \text{if} \begin{cases} l' \text{ is not a variable,} \\ \sigma \text{ is a mgu of } l \text{ and } l', \\ l\sigma \not\preceq r\sigma, \text{ and} \\ s\sigma \not\preceq t\sigma \end{cases}$$

[7] By compactness, even from a *finite* set of clauses.

[8] With our notation for clauses, the name superposition *left* is actually counterintuitive, but we keep it for compatibility with corresponding rules in the superposition calculus.

The last condition can be dropped, and the resulting inference rule is then called *ordered paramodulation left*.

The unit-sup-right rule (*unit superposition right*) applies a superposition step to a positive *unit* clause:

$$\mathsf{unit\text{-}sup\text{-}right}(\sigma)\ \frac{s[l'] \simeq t \leftarrow \qquad l \simeq r \leftarrow}{(s[r] \simeq t \leftarrow)\sigma} \quad \text{if} \begin{cases} l' \text{ is not a variable,} \\ \sigma \text{ is a mgu of } l \text{ and } l', \\ (s \simeq t)\sigma \not\preceq (l \simeq r)\sigma, \\ l\sigma \not\preceq r\sigma, \text{ and} \\ s\sigma \not\preceq t\sigma \end{cases}$$

The last condition can be dropped, and the resulting inference rule is then called *ordered unit paramodulation right*.

The general superposition right inference rule of [BG98] between non-unit clauses is not needed, essentially due to the presence of the splitting rule below.

The ref rule (*reflexivity*) eliminates a body literal on the grounds of being trivially true (after applying a substitution).

$$\mathsf{ref}(\sigma)\ \frac{\mathcal{A} \leftarrow s \simeq t, \mathcal{B}}{(\mathcal{A} \leftarrow \mathcal{B})\sigma} \quad \text{if } \sigma \text{ is a mgu of } s \text{ and } t$$

Finally, the announced splitting rule. It takes a disjunctive fact, applies a purifying substitution π to it and returns the instantiated head atoms, one conclusion per head atom.

$$\mathsf{split}(\pi)\ \frac{A_1, \ldots, A_m \leftarrow}{A_1\pi \leftarrow \quad \cdots \quad A_m\pi \leftarrow} \quad \text{if} \begin{cases} m \geq 2, \text{ and} \\ \pi \text{ is a purifying substitution for } A_1, \ldots, A_m \leftarrow \end{cases}$$

3.1 Redundant Inferences and Saturation

We write $C, D \Rightarrow_{\mathsf{sup\text{-}left}(\sigma)} E$ to denote a sup-left inference, i.e., an instance of the sup-left inference rule with left premise C, right premise D, conclusion E and substitution σ that satisfies the rule's side condition. We use analogous notation for an application of the sup-right inference rule, and for an application of ref we write, similarly, $C \Rightarrow_{\mathsf{ref}(\sigma)} E$. Likewise, $C \Rightarrow_{\mathsf{split}(\pi)} A_1 \leftarrow, \ldots, A_m \leftarrow$ denotes a split inference with premise C, purifying substitution π and conclusions $A_1 \leftarrow, \ldots, A_m \leftarrow$.

An R-inference, with $R \in \{\mathsf{sup\text{-}left}, \mathsf{unit\text{-}sup\text{-}right}, \mathsf{ref}\}$ is *ground* iff its constituent clauses C, D and E are ground. The substitution σ in a ground inference is irrelevant and may be assumed, without loss of generality, to be the empty substitution ϵ.

If $C, D \Rightarrow_{R(\sigma)} E$ is an R-inference (with D absent in the case of ref) and γ is a substitution such that $C\sigma\gamma, D\sigma\gamma \Rightarrow_{R(\epsilon)} E\gamma$ is a ground inference, then the latter inference is called a *ground instance* of the inference $C, D \Rightarrow_{R(\sigma)} E$.

For instance, by taking $\gamma = \{x \mapsto a\}$ one sees that the ground inference

$$(P(f(a)) \leftarrow), (f(a) \simeq a \leftarrow) \Rightarrow_{\mathsf{sup\text{-}right}(\epsilon)} P(a) \leftarrow$$

is a ground instance of the inference

$$(P(f(x)) \leftarrow), (f(y) \simeq y \leftarrow) \Rightarrow_{\mathsf{sup\text{-}right}(\{y \mapsto x\})} P(x) \leftarrow \quad .$$

In contrast,

$$(P(f(f(a))) \leftarrow), (f(a) \simeq a \leftarrow) \Rightarrow_{\mathsf{sup\text{-}right}(\epsilon)} P(f(a)) \leftarrow$$

is not a ground instance of the inference above, for any substitution γ. Intuitively, only such ground inferences can be ground instances of inferences where paramodulation takes place at positions that exist also at the non-ground level. This excludes ground inferences that are not liftable because they would require paramodulation into or below variables. We can define these notions for the split rule analogously: a split inference is *ground* if the premise is ground (and hence all its conclusions are ground). Similarly as above for the other rules, the purifying substitution π can always be assumed to be the empty substitution then.

If $C \Rightarrow_{\mathsf{split}(\pi)} A_1 \leftarrow , \ldots , A_m \leftarrow$ is a split inference and γ is a substitution such that $C\pi\gamma \Rightarrow_{\mathsf{split}(\epsilon)} A_1\gamma \leftarrow , \ldots , A_m\gamma \leftarrow$ is a ground split inference, then the latter inference is called a *ground instance* of the former inference.

Let $\mathcal{D}$ be a set of (possibly non-ground) clauses. A ground inference $C, D \Rightarrow_{\mathsf{sup\text{-}left}(\epsilon)} E$ or $C, D \Rightarrow_{\mathsf{sup\text{-}right}(\epsilon)} E$ is *redundant wrt.* $\mathcal{D}$ iff E is redundant wrt. $\mathcal{D}_C \cup \{D\}$. A ground inference $C \Rightarrow_{\mathsf{ref}(\epsilon)} E$ is *redundant wrt.* $\mathcal{D}$ iff E is redundant wrt. $\mathcal{D}_C$. And a ground inference $C \Rightarrow_{\mathsf{split}(\epsilon)} A_1 \leftarrow , \ldots , A_m \leftarrow$ is *redundant wrt.* $\mathcal{D}$ iff there is an i with $1 \leq i \leq m$ such that $A_i \leftarrow$ is redundant wrt. $\mathcal{D}_C$.

For all inference rules sup-left, unit-sup-right, ref and split, a (possibly non-ground) inference is *redundant wrt.* $\mathcal{D}$ iff each of its ground instances is redundant wrt. $\mathcal{D}$.

Intuitively a ground inference is redundant wrt. $\mathcal{D}$ iff its conclusion follows from a set of smaller clauses than the left premise, while fixing the right premise. Because all (ground) inferences work in a strictly order-decreasing way, adding the conclusion of an inference to the clause set the premises are taken from renders the inference redundant wrt. that set.[9] For instance, adding $P(a) \leftarrow$ to the set $\{(P(f(a)) \leftarrow), (f(a) \simeq a \leftarrow)\}$ renders the obvious sup-right inference redundant wrt. the resulting set.

It is not only redundant inferences that can be neglected. Also inferences where one or both parent clauses are redundant can be neglected. This is captured by the following definition.

Definition 3.1 (Saturation up to redundancy). *A clause set $\mathcal{C}$ is* saturated up to redundancy *iff for all clauses $C \in \mathcal{C}$ such that C is not redundant wrt. $\mathcal{C}$ all of the following hold:*

1. *Every inference $C \Rightarrow_{\mathsf{split}(\pi)} A_1 \leftarrow , \ldots , A_m \leftarrow$ such that $C\pi$ is not redundant wrt. $\mathcal{C}$, is redundant wrt. $\mathcal{C}$.*
2. *Every inference $C, D \Rightarrow_{R(\sigma)} E$, where $R \in \{\mathsf{sup\text{-}left}, \mathsf{unit\text{-}sup\text{-}right}\}$ and D is a fresh variant of a positive unit clause from $\mathcal{C}$, such that neither $C\sigma$ nor $D\sigma$ is redundant wrt. $\mathcal{C}$, is redundant wrt. $\mathcal{C}$.*

[9] This property makes it obvious that fair derivations, as defined later, exist.

3. *Every inference* $C \Rightarrow_{\mathsf{ref}(\sigma)} E$ *such that* $C\sigma$ *is not redundant wrt.* $\mathcal{C}$*, is redundant wrt.* $\mathcal{C}$*.*

For instance, the (satisfiable) propositional clause set $\mathcal{C} = \{(A, B \leftarrow), (\leftarrow A)\}$ is *not* saturated up to redundancy. By an application of the split rule to $A, B \leftarrow$ one can infer $A \leftarrow$ and $B \leftarrow$, and adding, say, $B \leftarrow$ to $\mathcal{C}$ renders the clause $A, B \leftarrow$ redundant.

As an example for a non-ground split inference consider a clause $P(x), Q(x) \leftarrow$ from some clause set. One may want to avoid applying all purifying substitutions to it. Fortunately, Definition 3.1-1 does not prescribe that at all. For instance, when the clause set includes an equation $a \simeq b \leftarrow$ (where $a \succ b$), then purifying $P(x), Q(x) \leftarrow$ by $\pi = \{x/b\}$, yielding $P(b), Q(b) \leftarrow$, and adding $P(b) \leftarrow$ to the clause set is sufficient to render the split inference with purifying substitution $\{x/a\}$ redundant, as the clause $P(a) \leftarrow$ follows from $P(b) \leftarrow$ and $a \simeq b \leftarrow$, both of which are smaller than $P(a), Q(a) \leftarrow$.

Theorem 3.2 (Static Completeness). *Let* $\mathcal{C}$ *be a clause set saturated up to redundancy. If* $\square \notin \mathcal{C}$ *then* $\mathcal{C}$ *is* E*-satisfiable.*

The proof employs the model-construction technique originally developed for the superposition calculus, but adapted to our needs. The difference come from the facts that in our case all side premises are unit clauses, and so there is no equality factoring (or merging paramodulation) inference rule, and that we need a splitting rule.

Notice that Theorem 3.2 applies to a *statically* given clause set $\mathcal{C}$. The connection to the *dynamic* derivation process of the E-hyper tableau calculus will be given later, and Theorem 3.2 will be essential in proving the completeness of the E-hyper tableau calculus.

4 E-Hyper Tableaux

In [BFN96], based on [LMG94], hyper tableau have been introduced as labeled trees over *literals* (which are universally quantified, and hence can be seen as unit clauses). For our purposes, however, a generalization towards trees over *clauses* is better suited. This is, because *new* clauses can now be derived as the derivation proceeds, and these clauses are context dependant (branch local), and tableaux are an obvious data structure to deal with this context dependency.

A *labeled tree over a set* M is a pair $(\mathcal{T}, \lambda)$ consisting of a finite, ordered tree $\mathcal{T}$ and a labeling function λ that maps each node of $\mathcal{T}$ to some element from M. A *(clausal) tableau over a signature* Σ is a labeled tree over the set of Σ-clauses.

We use the letter $\mathbf{T}$ to denote tableaux.

Let $\mathbf{B}$ be a branch of a tableau $\mathbf{T}$ of length n, i.e., a sequence of nodes $(\mathbf{N}_1, \ldots, \mathbf{N}_n)$, for some $n \geq 0$, where $\mathbf{N}_1$ is the root and $\mathbf{N}_n$ is the leaf of $\mathbf{B}$. Each of the clauses $\lambda(\mathbf{N}_i)$, for $i = 1, \ldots, n$, is called a *(tableau) clause of* $\mathbf{B}$.

Occasionally it is convenient to read a branch $\mathbf{B}$ as the multiset of its tableau clauses $\lambda(\mathbf{B}) := \{D \mid D \text{ is a tableau clause of } \mathbf{B}\}$. This allows us to write, for

instance, $C \in \mathbf{B}$ instead of $C \in \lambda(\mathbf{B})$. Furthermore, if $\mathbf{B}$ is a branch of a tableau $\mathbf{T}$ we write $\mathbf{B} \cdot C$ and mean the tableau obtained from $\mathbf{T}$ by adding an edge from the leaf of $\mathbf{B}$ to a fresh node labeled with C. Furthermore, we write $\mathbf{B} \cdot \mathbf{B}'$ to denote the branch obtained by concatenating the branch $\mathbf{B}$ and the node sequence $\mathbf{B}'$.

4.1 Extension Rules

We define two derivation rules for extending branches in a given tableau.

The Split rule branches out on an instance of a positive clause; its conclusions are labeled as "decision clauses", as indicated by the annotation $^{\mathrm{d}}$. The role of this labeling will become clear below in Section 4.2.

$$\mathsf{Split}\ \frac{\mathbf{B}}{\mathbf{B} \cdot A_1 \leftarrow^{\mathrm{d}} \quad \cdots \quad \mathbf{B} \cdot A_m \leftarrow^{\mathrm{d}}}\ \text{if}\ \begin{cases} \text{there is a clause } C \in \mathbf{B} \text{ and} \\ \text{a substitution } \pi \text{ such that} \\ \quad C \Rightarrow_{\mathsf{split}(\pi)} A_1 \leftarrow, \ldots, A_m \leftarrow \text{ and} \\ \quad \mathbf{B} \text{ contains no variant of } A_i \leftarrow, \\ \quad \text{for each } i = 1, \ldots, m \end{cases}$$

The clause C is called the *selected clause (of a* Split *inference)*.

The Equality rule applies an inference rule for equality reasoning from Section 3 to a body literal.

$$\mathsf{Equality}\ \frac{\mathbf{B}}{\mathbf{B} \cdot E}\ \text{if}\ \begin{cases} \text{there is a clause } C \in \mathbf{B}, \\ \text{a fresh variant } D \text{ of a positive unit clause in } \mathbf{B}, \text{ and} \\ \text{a substitution } \sigma \text{ such that} \\ \quad C, D \Rightarrow_{R(\sigma)} E \text{ with } R \in \{\mathsf{sup\text{-}left}, \mathsf{unit\text{-}sup\text{-}right}\} \text{ or} \\ \quad C \Rightarrow_{\mathsf{ref}(\sigma)} E, \text{ and} \\ \mathbf{B} \text{ contains no variant of } E \end{cases}$$

In both rules, the test for the conclusion(s) being not contained in $\mathbf{B}$ is needed in interplay with deletion of clauses based on non-proper subsumption (see the Del below).

For later use, we say that an application of a Split, Sup-left, Unit-sup-right or Ref derivation rule to a branch $\mathbf{B}$ is *redundant* iff its conclusion (at least one of its conclusions, in the case of Split) is redundant wrt. $\mathbf{B}$.

4.2 Deletion and Simplification Rules

From a practical point of view, deletion of redundant clauses and simplification operations on clauses are crucial. We will introduce these now. Adding such rules is a major addition to the hyper tableau calculus and involves a more sophisticted technical treatment than that in [BFN96]. This is, because hyper tableau as defined in [BFN96] are *non-destructive*, in the sense that extending a branch goes along with increasing the set of its corresponding labels (unit clauses). This is no longer the case in presence of, for instance, the Del rule (*deletion*) below,

which removes a clause that is redundant in a branch or subsumed by another clause in the branch.

Also, to preserve the calculus' soundness, arbitrary deletion of redundant clauses is not possible. A clause can be deleted only on the condition that none of the clauses which make the clause redundant is a clause which has been introduced at a later "decision level" (i.e. one that occurs further down in the tree below a more leafwards decision clause). This is formalized next.

$$\mathsf{Del}\ \frac{\mathbf{B}\cdot C^{(\mathrm{d})}\cdot\mathbf{B}_1\cdot\mathbf{B}_2}{\mathbf{B}\cdot\mathbf{t}\simeq\mathbf{t}\leftarrow^{(\mathrm{d})}\cdot\mathbf{B}_1\cdot\mathbf{B}_2}\quad \text{if}\ \begin{cases}(1)\ C \text{ is redundant wrt. } \mathbf{B}\cdot\mathbf{B}_1, \text{ or some}\\ \text{clause in } \mathbf{B}\cdot\mathbf{B}_1 \text{ non-properly subsumes } C, \text{ and}\\ (2)\ \mathbf{B}_1 \text{ does not contain a decision clause}\end{cases}$$

The notation $^{(\mathrm{d})}$ is meant to say that if there is a label $^{\mathrm{d}}$, it is preserved when replacing C by $\mathbf{t}\simeq\mathbf{t}\leftarrow$.

Observe that our redundancy notion does not cover non-proper subsumption.[10] For instance, the clause $P(a)\leftarrow$ is *not* redundant wrt. $\{P(x)\leftarrow\}$ (and neither is the clause $P(y)\leftarrow$). Therefore, deletion of non-properly subsumed clauses has been taken care of explicitly.

The next rule, Simp (simplification), replaces a clause by another one that is smaller in the ordering:

$$\mathsf{Simp}\ \frac{\mathbf{B}\cdot C^{(\mathrm{d})}\cdot\mathbf{B}_1\cdot\mathbf{B}_2}{\mathbf{B}\cdot D^{(\mathrm{d})}\cdot\mathbf{B}_1\cdot\mathbf{B}_2}\quad \text{if}\ \begin{cases}(1)\ \mathbf{B}\cdot C\cdot\mathbf{B}_1\models_{\mathrm{E}} D,\\ (2)\ C \text{ is redundant wrt. } \mathbf{B}\cdot D\cdot\mathbf{B}_1, \text{ and}\\ (3)\ \mathbf{B}_1 \text{ does not contain a decision clause}\end{cases}$$

The Simp rule covers, for instance, standard rewriting by unit clauses.

The condition (2) in Del is needed for completeness reasons, and the condition (3) in Simp is needed for both completeness and soundness reasons. They make sure that no deletion or simplification step is justified by a clause from a decision level further down in the tableau. Such a step would in general be justified only in the branch containing the used clauses, but not in the other branches. For illustration consider the following clause set.

$$P(a)\leftarrow \qquad (1) \qquad\qquad \leftarrow P(b) \qquad (2) \qquad\qquad a\simeq b,\ Q\leftarrow \qquad (3)$$

After a Split with clause (3) a branch containing the decision clause $a\simeq b\leftarrow$ comes up. If condition (3) in Simp were dropped (and $a\succ b$), then clause (1) could be simplified to $P(b)\leftarrow$, leading to a refutation. This would be unsound because the simplification is not justified in the branch containing $Q\leftarrow$ although it would contain the simplified literal. But with the restrictions in place we arrive at the following lemma.

Lemma 4.1. *For each of the derivation rules* Split, Equality, Del *and* Simp, *if the premise of the rule is* E*-satisfiable, then one of its conclusions is* E*-satisfiable as well.*

[10] A clause C non-properly subsumes a clause D iff $C\sigma = D$ for some substitution σ.

For similar reasons as for Simp, the Del rule cannot just delete the clause C^{d} mentioned in the premise, as the deletion would remove the separation of $\mathbf{B}$ and $\mathbf{B}_1$ by a decision clause (while the replacement by $\mathbf{t} \simeq \mathbf{t} \leftarrow {}^{\mathrm{d}}$ preserves the separation).

A different approach to deletion and simplification is implemented in the SPASS prover [Wei01]. The corresponding rules in SPASS are even more general than ours as they allow to ignore the decision levels. But then, in general, a deleted or simplified clause must be reinserted on backtracking to an earlier decision level. This is never necessary in our case, essentially because of disallowing "backward" deletion and simplification steps across decision levels, as just discussed in the previous example.

4.3 Derivations

We say that a branch of a tableau is *closed* iff it contains the empty clause $\square$.[11] A branch that is not closed is also called *open*. A tableau is *closed* iff each of its branches is closed, and it is *open* iff it is not closed (i.e., if it has an open branch).

An *(*E*-hyper tableau) derivation* from a set $\{C_1, \ldots, C_n\}$ of Σ-clauses is a possibly infinite sequence of tableaux $\mathbf{D} = (\mathbf{T}_i)_{0 \le i < \kappa}$ such that

1. $\mathbf{T}_0$ is the clausal tableau over Σ that consists of a single branch of length n with tableau clauses $C_1, \ldots, C_n$.[12], and
2. for all $i > 0$, $\mathbf{T}_i$ is obtained from $\mathbf{T}_{i-1}$ by a single application of one of the derivation rules in Sections 4.1 and 4.2 to some open branch of $\mathbf{T}_{i-1}$, called the *selected branch*.

Recall that a tableau $\mathbf{T}$ is of the form $(\mathcal{T}, \lambda)$, where $\mathcal{T}$ is a tree, i.e., a pair $(\mathcal{N}, \mathcal{E})$ where $\mathcal{N}$ is the set of the nodes of $\mathcal{T}$ and $\mathcal{E}$ is the set of the edges of $\mathcal{T}$.

A derivation $\mathbf{D} = ((\mathcal{N}_i, \mathcal{E}_i), \lambda_i)_{i<\kappa}$ determines a *limit tree* $((\bigcup_{i<\kappa} \mathcal{N}_i, \bigcup_{i<\kappa} \mathcal{E}_i)$. It is easy to show that a limit tree of a derivation $\mathbf{D}$ is indeed a (possibly infinite) tree.

Now let $\mathbf{T}$ be the limit tree of some derivation, let $\mathbf{B} = (\mathbf{N}_i)_{i<\kappa}$ be a (possibly infinite) branch in $\mathbf{T}$ with κ nodes, and let $\mathbf{B}_i = (\mathbf{N}_1, \ldots, \mathbf{N}_i)$ be the initial segment of $\mathbf{B}$ with i nodes, for all $i < \kappa$. Define $\mathbf{B}_\infty = \bigcup_{i<\kappa} \bigcap_{i \le j < \kappa} \lambda_j(\mathbf{B}_j)$, the set of *persistent clauses (of* $\mathbf{B}$*)*.

Definition 4.2 (Exhausted Branch). *Let* $\mathbf{T}$ *be a limit tree, and let* $\mathbf{B} = (\mathbf{N}_i)_{i<\kappa}$ *be a branch in* $\mathbf{T}$ *with* κ *nodes. The branch* $\mathbf{B}$ *is* exhausted *iff it does not contain the empty clause, and for every clause* $C \in \mathbf{B}_\infty$ *and every fresh variant* D *of every positive unit clause in* $\mathbf{B}_\infty$ *such that neither* C *nor* D *is redundant wrt.* $\mathbf{B}_\infty$ *all of the following hold, for all* $i < \kappa$ *such that* $C \in \mathbf{B}_i$ *and* D *is a variant of a clause in* $\mathbf{B}_i$*:*

[11] We write $\square$ instead of " $\leftarrow$ ".

[12] The order does not matter, as the collection of tableau clauses of a branch will be seen as sets. For technical reasons we assume that no clause C_i is a variant of a clause C_j, for all $1 \le i < j \le n$, but this is obviously not an essential restriction.

1. *if* Split *is applicable to* $\mathbf{B}_i$ *with underlying inference* $C \Rightarrow_{\mathsf{split}(\pi)} A_1 \leftarrow, \ldots, A_m \leftarrow$ *and* $C\pi$ *is not redundant wrt.* $\mathbf{B}_i$*, then there is a* $j < \kappa$ *such that the inference* $C \Rightarrow_{\mathsf{split}(\pi)} A_1 \leftarrow, \ldots, A_m \leftarrow$ *is redundant wrt.* $\mathbf{B}_j$*.*
2. *if* Equality *is applicable to* $\mathbf{B}_i$ *with underlying inference* $C, D \Rightarrow_{R(\sigma)} E$*, for some* $R \in \{\mathsf{sup\text{-}left}, \mathsf{unit\text{-}sup\text{-}right}\}$*, and neither* $C\sigma$ *nor* $D\sigma$ *is redundant wrt.* $\mathbf{B}_i$*, then there is a* $j < \kappa$ *such that the inference* $C, D \Rightarrow_{R(\sigma)} E$ *is redundant wrt.* $\mathbf{B}_j$*.*
3. *if* Equality *is applicable to* $\mathbf{B}_i$ *with underlying inference* $C \Rightarrow_{\mathsf{ref}(\sigma)} E$ *and* $C\sigma$ *is not redundant wrt.* $\mathbf{B}_i$*, then there is a* $j < \kappa$ *such that the inference* $C \Rightarrow_{\mathsf{ref}(\sigma)} E$ *is redundant wrt.* $\mathbf{B}_j$*.*

A *refutation of a clause set* $\mathcal{C}$ is a finite derivation of $\mathcal{C}$ that ends in a closed tableau.

A derivation is *fair* iff it is a refutation or its limit tree has an exhausted branch.

In the preceeding definition, actually carrying out a Split inference with a clause C and (irreducible) purifying substitution π, when applicable, will achieve the conclusion, i.e. make $C\pi$ redundant wrt. $\mathbf{B}_j$. The analogous holds for the Equality inferences in items 2 and 3. This observation indicates that proof procedures implementing fair derivations indeed can be given.

Theorem 4.3 (Soundness of E**-Hyper Tableaux).** *Let* $\mathcal{C}$ *be a clause set that has a refutation. Then* $\mathcal{C}$ *is* E*-unsatisfiable.*

For the completeness direction we need the following result:

Proposition 4.4 (Exhausted branches are saturated up to redundancy). *If* $\mathbf{B}$ *is an exhausted branch of a limit tree of some fair derivation then* $\mathbf{B}_\infty$ *is saturated up to redundancy.*

Proposition 4.4 and Theorem 3.2 entails our main result:

Theorem 4.5 (Completeness of E**-Hyper Tableaux).** *Let* $\mathcal{C}$ *be a clause set and* $\mathbf{T}$ *be the limit tree of a fair derivation* $\mathbf{D}$ *of* $\mathcal{C}$*. If* $\mathbf{D}$ *is not a refutation then* $\mathcal{C}$ *is satisfiable.*

Because the proof of this theorem refers to the proof of Theorem 3.2, the model constructed in the proof of Theorem 3.2 provides a strengthening of Theorem 4.5 by being more specific.

Corollary 4.6 (Bernays-Schönfinkel Class with Equality). *The* E*-hyper tableau calculus can be used as a decision procedure for the Bernays-Schönfinkel class with equality, i.e., for function free formulae with the quantifier prefix* $\exists^*\forall^*$*.*

The proof of Corollary 4.6 follows from the soundness and completeness results, and the facts that the calculus cannot derive clauses that grow in length, or that grow in term depth (using the assumption that no non-nullary function symbols are present) or that are variants of clauses already contained in the

branch. Therefore any (exhausted) branch derivable must be finite.[13] Because of the finite branching of hyper tableaux and by Koenig's Lemma it follows that any (limit) derivation must be finite.

5 Restricting Split and the Relation to Splitting in SPASS

For performance reasons it is mandatory to restrict the search space induced by having to apply purifying substitutions in Split rule applications. The fairness criteria in Definition 4.2 already support that. For instance, one can take advantage of avoiding purifying substitutions that are *reducible*, as they lead to redundant inferences.

Definition 5.1 (Reducible substitution). *Let $\mathcal{C}$ be a clause set and σ a substitution. We say that σ is* reducible wrt. *$\mathcal{C}$ iff there is a term $t \in \mathcal{R}\mathrm{an}(\sigma)$[14], a unit clause $l \simeq r \leftarrow \in \mathcal{C}$ and a (matching) substitution μ such that $l\mu$ occurs in t and $l\mu \succ r\mu$.*

We say that σ is *irreducible wrt.* $\mathcal{C}$ if σ is not reducible wrt. $\mathcal{C}$.

Obviously, for each (positive) clause $C = A_1, \ldots, A_m \leftarrow$ in a branch $\mathbf{B}$ and each purifying substitution π_0 for C there is a maximal chain $C\pi_0 \succ C\pi_1 \succ \cdots \succ C\pi_n$, for some $n \geq 0$, where π_i is obtained from π_{i-1} by one-step rewriting a term of its range with a positive unit clause from $\mathbf{B}$ and such that π_n is irreducible wrt. $\mathbf{B}$. It is not difficult to see that, by equality, applying Split with $C\pi_n$ renders the Split inferences with $C\pi_0, \ldots, C\pi_{n-1}$ redundant (wrt. all branches obtained by splitting $C\pi_n$). No reducible purifying substitution need therefore ever be considered in Split inferences to obtain an exhausted branch.

An example of such a situation is $C = P(x), Q(x) \leftarrow$, $a \simeq b \leftarrow \in \mathbf{B}$, $a \succ b$, $\pi_0 = \{x/a\}$ and $\pi_1 = \{x/b\}$. Split with $P(b), Q(b) \leftarrow$ alone to extend $\mathbf{B}$ is sufficient.

A significantly different split rule is implemented in the SPASS prover [Wei01]. It does not apply a purifying substitution to force partitioning a clause into variable disjoint parts. Instead, it can split on clauses only that are already partitioned.

We do not claim that our approach is always preferrable in practice. Yet, there are situations where indeed it is. By way of example, consider the following clauses

$$f(a) \simeq a \leftarrow \quad (1) \qquad\qquad f(g(x)) \simeq g(f(x)) \leftarrow \quad (3)$$
$$g(a) \simeq a \leftarrow \quad (2) \qquad\qquad p(f(x)), p(g(x)) \leftarrow \quad (4)$$

Suppose a precedence $f \succ g \succ a$ (or $g \succ f \succ a$, as the problem is symmetric in f and g), lifted to any simplification ordering. All superposition inferences among the clauses 1-3 are redundant, and a prover like SPASS will detect that.

[13] The situation is slightly more complicated due to the Simp and Del rules.
[14] As usual, the *range* of a substitution σ is $\mathcal{R}\mathrm{an}(\sigma) = \{x\sigma \mid x\sigma \neq x\}$.

Among others, there is a superposition inference between clause 4 and 3, which yields the clause

$$p(g(f(x))), p(g(g(x))) \leftarrow \quad . \tag{5}$$

In fact this inference is redundant, too. To see this, consider any ground substitution γ. It must map x to some term comprised of a combination of fs, gs and (one) a, e.g. $\gamma = \{x/f(f(g(f(a))))\}$. Now, any ground instance obtained from clause 5 in this way can be reduced by the unit clauses 1-3 in one or more steps to the clause $p(f(a)), p(g(a)) \leftarrow$ (they can be reduced even further), which is a ground instance of clause 4 and which is smaller in the ordering than the ground instance of clause 5 we started with. By this argument the superposition inference leading to clause 5 is redundant (and need not be carried out).

Notice that this argumentation takes the clause set's signature into account. However, the commonly implemented redundancy criteria do not do that. In particular, for instance, SPASS does not find a finite saturation of the clause set above. In contrast, E-hyper tableaux are aware of the input signature and the redundancy criteria based on irreducible purifying substitutions, as mentioned above, are strong enough to achieve termination.[15] To see this, it is enough to observe that every purifying substitution, like $\pi = \{x/f(f(g(f(a))))\}$, is reducible (to $\pi = \{x/a\}$) wrt. every branch containing clauses 1 and 2. Thus, the *only* instance of clause 4 to be considered for splitting (in presence of 1-3) is $p(f(a)), p(g(a)) \leftarrow$ (which can be simplified further). Moreover, this can easily be achieved by adding the following "logic program"

$$ran(a) \leftarrow \quad (6) \qquad ran(f(x)) \leftarrow ran(x) \quad (7) \qquad ran(g(x)) \leftarrow ran(x) \quad (8)$$

which, in combination with rewriting by unit clauses will enumerate in its *ran* predicate the ground terms of the input signature that are irreducible wrt. the orientable current positive unit clauses. In presence of clauses 1 and 2 this is the singleton $\{a\}$. The general form of the "logic program" has, of course, already been used within SATCHMO [MB88] and some descendants. To our knowledge, though, it was never observed before that equational reasoning can help to confine the *ran*-predicate.

6 Implementation

We have implemented the E-hyper tableau calculus by extending our existing KRHyper system. KRHyper is a hyper tableaux theorem prover, and as such it lacked equality handling in the original version. The modified system, called E-KRHyper, adapts the methods of its precursor to accommodate the new inferences, while at the same time retaining the original functionality.

The derivation proceeds in a bottom-up manner. Internally, clauses are divided into three sets, one containing the positive non-equational units (*facts*),

[15] More precisely, there is a finite derivation in the E-hyper tableau calculus, and any reasonable implementation, like our E-KRHyper system, will find it.

the other consisting of the positive non-unit clauses (*disjunctions*), and the third including both the unit equations and the clauses with negative literals (*rules*). The hyper extension inference of KRHyper is equivalent to a series of Sup-left, Ref and Split applications, and therefore it is kept in place in E-KRHyper as a shortcut inference for the resolution of non-equational atoms. The E-hyper tableau is generated depth first, with the current state of the three clause sets always representing a single branch. The Split on a disjunction is only executed when the other inference possibilities have been exhausted. An iterative deepening strategy with a limit on the maximum term weight of generated clauses is employed. This ensures the refutational completeness and a fair search control, as it prevents splitting from being delayed indefinitely by other inferences.

Clauses are derived by a loop iterating over the rules, with each rule in turn accessing indexes in the search for inference partners. The inferred clauses are added to their respective sets after having passed the weight and subsumption tests. The dynamic nature of the rule set represents a major change compared to the previous system version. As the hyper tableaux calculus has no inferences that generate new rule clauses, this set remained fixed throughout the derivation of KRHyper, and many optimizations on the input could be delegated to preprocessing. Operations like the clause subsumption test are necessary for the new calculus, and they are now employed to optimize the input clauses as well.

The superposition inferences utilize a discrimination-tree based index [McC92] over the subterms of clauses, and terms are ordered according to the recursive path ordering (RPO). As an option, the backtracking mechanism allows the removal of redundant clauses from the entire current branch, beyond the limits set in Section 4.2. More details about the system can be found in [PW07]; it is available under the GNU Public License from the E-KRHyper website at `http://www.uni-koblenz.de/~bpelzer/ekrhyper`.

7 Conclusion

We have presented a tableau calculus with equality, by integrating superposition based inference rules into the hyper tableau calculus rules. Our main result is its soundness and completeness, the latter in combination with redundancy criteria. The calculus is implemented in the E-KRHyper system, an extension of our existing KRHyper prover.

Acknowledgements. We thank the anonymous reviewers for their useful comments on improving the paper's presentation.

References

[Bec97] Beckert, B.: Semantic Tableaux With Equality. Journal of Logic and Computation 7(1), 39–58 (1997)

[BFGHS04] Baumgartner, P., Furbach, U., Gross-Hardt, M., Sinner, A.: Living Book – Deduction, Slicing, and Interaction. J. of Aut. Reasoning 32(3) (2004)

[BFN96] Baumgartner, P., Furbach, U., Niemelä, I.: Hyper Tableaux. In: Orłowska, E., Alferes, J.J., Moniz Pereira, L. (eds.) JELIA 1996. LNCS, vol. 1126, Springer, Heidelberg (1996)

[BG98] Bachmair, L., Ganzinger, H.: Chapter 11: Equational Reasoning in Saturation-Based Theorem Proving. In: Bibel, W., Schmitt, P.H. (eds.) Automated Deduction. A Basis for Applications, vol. 1, Kluwer, Dordrecht (1998)

[BS06] Baumgartner, P., Schmidt, R.: Blocking and Other Enhancements for Bottom-up Model Generation Methods. In: Furbach, U., Shankar, N. (eds.) IJCAR 2006. LNCS (LNAI), vol. 4130, Springer, Heidelberg (2006)

[BT05] Baumgartner, P., Tinelli, C.: The Model Evolution Calculus with Equality. In: Nieuwenhuis, R. (ed.) Automated Deduction – CADE-20. LNCS (LNAI), vol. 3632, Springer, Heidelberg (2005)

[DV96] Degtyarev, A., Voronkov, A.: Equality Elimination for the Tableau Method. In: Limongelli, C., Calmet, J. (eds.) DISCO 1996. LNCS, vol. 1128, Springer, Heidelberg (1996)

[DV98] Degtyarev, A., Voronkov, A.: What you Always Wanted to Know About Rigid E-Unification. Journal of Automated Reasoning 20(1), 47–80 (1998)

[FO06] Furbach, U., Obermaier, C.: Applications of Automated Reasoning. In: Freksa, C., Kohlhase, M., Schill, K. (eds.) KI 2006. LNCS (LNAI), vol. 4314, Springer, Heidelberg (2007)

[Gie01] Giese, M.: Incremental Closure of Free Variable Tableaux. In: Goré, R.P., Leitsch, A., Nipkow, T. (eds.) IJCAR 2001. LNCS (LNAI), vol. 2083, Springer, Heidelberg (2001)

[Gie02] Giese, M.: A Model Generation Style Completeness Proof For Constraint Tableaux With Superposition. In: Egly, U., Fermüller, C. (eds.) TABLEAUX 2002. LNCS (LNAI), vol. 2381, Springer, Heidelberg (2002)

[Gie03] Giese, M.: Simplification Rules for Constrained Formula Tableaux. In: Mayer, M.C., Pirri, F. (eds.) TABLEAUX 2003. LNCS, vol. 2796, Springer, Heidelberg (2003)

[LMG94] Letz, R., Mayr, K., Goller, C.: Controlled Integrations of the Cut Rule into Connection Tableau Calculi. J. of Aut. Reasoning 13 (1994)

[LS02] Letz, R., Stenz, G.: Integration of Equality Reasoning into the Disconnection Calculus. In: Egly, U., Fermüller, C. (eds.) TABLEAUX 2002. LNCS (LNAI), vol. 2381, Springer, Heidelberg (2002)

[MB88] Manthey, R., Bry, F.: SATCHMO: a Theorem Prover Implemented in Prolog. In: Lusk, E.R., Overbeek, R. (eds.) 9th International Conference on Automated Deduction. LNCS, vol. 310, Springer, Heidelberg (1988)

[McC92] McCune, W.: Experiments with Discrimination-Tree Indexing and Path Indexing for Term Retrieval. J. of Aut. Reasoning 9(2), 147–167 (1992)

[NR01] Nieuwenhuis, R., Rubio, A.: Paramodulation-based Theorem Proving. In: Robinson, J.A., Voronkov, A. (eds.) Handbook of Automated Reasoning, Elsevier and MIT Press (2001)

[PW07] Pelzer, B., Wernhard, C.: System Description: E-KRHyper. In: Pfenning, F. (ed.) CADE-21. LNCS, Springer, Heidelberg (2007)

[SS06] Sutcliffe, G., Suttner, C.: The State of CASC. AI Communications 19(1), 35–48 (2006)

[Wei01] Weidenbach, C.: Combining Superposition, Sorts and Splitting. In: Robinson, A., Voronkov, A. (eds.) Handbook of Automated Reasoning. North Holland (2001)