The Hyper Tableaux Calculus with Equality and an Application to Finite Model Computation

Peter Baumgartner
NICTA, Canberra
Australia
Peter.Baumgartner@nicta.com.au

Björn Pelzer
Universität Koblenz-Landau, Koblenz
Germany
bpelzer@uni-koblenz.de

Ulrich Furbach
Universität Koblenz-Landau, Koblenz
Germany
uli@uni-koblenz.de

September 5, 2008

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 superposition are drawn (only) from a set of positive unit clauses, and superposition inferences into positive literals is restricted into (positive) unit clauses only. 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 of the calculus, but we also show how to apply the calculus for finite model computation, and we briefly describe the implementation.

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.

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

[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 $\forall x, y\ (p(x, y) \vee q(x))$.

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. 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 6 for details and Section 7 for exploiting this idea for finite model computation.

Also, the hyper tableau calculus is the basis of the KRHyper prover, which is used in various applications [FO06, BF03, 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 in a clause and using a prover that supports splitting (of variable-disjoint subclauses) like SPASS [Wei01]. Even then, there remain differences. We discuss these issues in Section 6.

The article [LS02] discusses various ways of integrating equality reasoning in disconnection tableaux. It includes a variant based on ordered paramodulation, where paramodulation inferences are determined by inspecting connections between literals of two clauses. Only comparably weak redundancy criteria are available.

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.

Both caluli belong to the family of instance-based methods, which are conceptually rather different to resolution- or tableau calculi as considered here.

[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.

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 6 we consider improvements for splitting and discuss the relation with splitting in the SPASS prover. Section 7 shows how to apply the calculus for finite model computation. Section 8 describes the implementation of the E-KRHyper system. A number of the results in this paper have first been desribed in [BFP07]. Apart from updating the presentation and including the new Section 7 we have extended the paper with a full complement of proofs in Section A.

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 $\mathcal{V}\mathrm{ar}(t) = \emptyset$.

A substitution σ is a mapping from X to Σ, with a finite domain $dom(\sigma) = \{x \mid x\sigma \neq x\}$ and a finite range $ran(\sigma) = \{x\sigma \mid x\sigma \neq x\}$, $x \in X$. A ground substitution γ is a substitution with $vars(ran(\gamma)) = \emptyset$. A renaming ρ is a substitution which is a bijection of X onto itself. Given two terms s and t, a substitution σ is a unifier for s and t if $s\sigma = t\sigma$. σ is a most general unifier (mgu), if for any other unifier τ for s and t there is a substitution λ with $\sigma\lambda = \tau$.

A term s is an instance of a term t if there is a substitution σ such that $s\sigma = t$. s is a variant of t if there is a renaming ρ such that $s\rho = t$. A variant is fresh as long as it shares no variables with any other term. All of the above is extended from terms to literals and clauses in the obvious way.

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. The notion of positions is extended from terms to literals in the obvious way.

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 $\neg(s \simeq t)$ 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 $t \not\simeq t$.

[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.

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 $\succ$. Given two clauses C and D, $C \succ D$ holds iff C and D are not variants and for each literal L exclusive to D there exists a literal K exclusive to C with $K \succ L$. 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, 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, which is an E-interpretation. 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 $F' \models_{\mathrm{E}} F$.

[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.

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:

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

If the last condition is dropped, then the resulting inference rule is called *ordered paramodulation left*. This rule will not be used in our calculus.

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

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

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.

[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 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. $\mathcal{C}$.

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 $\pi =$

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

$\{x/b\}$, 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 $\Box \notin \mathcal{C}$ 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 *tree* is 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 *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.

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

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.

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

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).

Without this test, it is conceivable the calculus derives the following sequence of branches:

$$\begin{array}{lllll} \ldots, & (P(x) \leftarrow) & & & \\ \ldots, & (P(x) \leftarrow), & (P(y) \leftarrow) & & \\ \ldots, & (\mathbf{t} \simeq \mathbf{t} \leftarrow), & (P(y) \leftarrow) & & \\ \ldots, & (\mathbf{t} \simeq \mathbf{t} \leftarrow), & (P(y) \leftarrow), & (P(x) \leftarrow) & \\ \ldots, & (\mathbf{t} \simeq \mathbf{t} \leftarrow), & (\mathbf{t} \simeq \mathbf{t} \leftarrow), & (P(x) \leftarrow) & \\ \ldots, & (\mathbf{t} \simeq \mathbf{t} \leftarrow), & (\mathbf{t} \simeq \mathbf{t} \leftarrow), & (P(x) \leftarrow), & (P(y) \leftarrow) \\ \ldots, & (\mathbf{t} \simeq \mathbf{t} \leftarrow), & (\mathbf{t} \simeq \mathbf{t} \leftarrow), & (\mathbf{t} \simeq \mathbf{t} \leftarrow), & (P(y) \leftarrow) \\ & \ldots & & & \end{array}$$

The calculus alternately adds and deletes unit clauses. For the additions the calculus alternates between $P(x) \leftarrow$ and $P(y) \leftarrow$. The Del applications delete the next-youngest unit respectively. As the inference conclusions are not checked for already having variants in the branch, there always exist one or two variants of $P(x) \leftarrow$, even though no particular instance ultimately persists. In spite of no new clauses being added, the branch grows indefinitely. The problem with such situations is there is no "well-founded" way to argue in the completeness proof that $P(x) \leftarrow$ will be satisfied by the candidate model.

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.

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

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:

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

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.

$$P(a)\leftarrow \tag{1}$$
$$\leftarrow P(b) \tag{2}$$
$$a\simeq b,\ Q\leftarrow \tag{3}$$

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

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.*

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

In the following, the letter κ will denote an ordinal smaller than or equal to the first infinite ordinal.

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 (i.e. of length κ) 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.

[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.

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 multiset of *persistent clauses (of* $\mathbf{B}$*)*.

Recall that tableaux clauses can be labeled as decision clauses. These labels are preserved when building the limit tree, i.e., the tableaux clauses in a limit tree are possibly also labeled as decision clauses. However, the labels are ignored when building the persistent clauses of a branch. If two clauses differ only in their label, they count as equal then.

Intuitively, the central property of a limit branch is a "static" one, saturation up to redundancy, for which the labels are not relevant. However, the derivation of a limit branch needs to take the labels into account.

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 $\mathbf{B}_i$:

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 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* E*-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 Derivation Examples

Figure 1 shows an E-hyper tableau that has been derived from the given clauses (1) - (6). The right branch of the tableau is open, and no further extension steps can be applied. Clause (14) cannot be split, as the resulting decision clause $R(b) \simeq t \leftarrow$ is already an element of the branch. Del steps can be used to further overwrite clauses (7) (redundant wrt. clause (8)) and (13) (redundant wrt. clause (14)).

Figure 2 illustrates the usage of the Del and Simp rules. Equality extensions are used to construct the tableau consisting of clauses (1) - (14). Clause (10) is non-properly subsumed by clause (14) and can thus be deleted, replacing it with (10') - $t \simeq t \leftarrow$. Clause (9) is redundant wrt. clause (12) and the simpler clause (3). The Simp rule replaces clause (9) with (9') - $Q(a) \simeq t \leftarrow$. As (9') itself is again non-properly subsumed by (3), it can be deleted in a further step. Note that clause (1) cannot be deleted by clause (14), because there is a decision clause between the two clauses in the branch.

[13]The situation is slightly more complicated due to the Simp and Del rules.

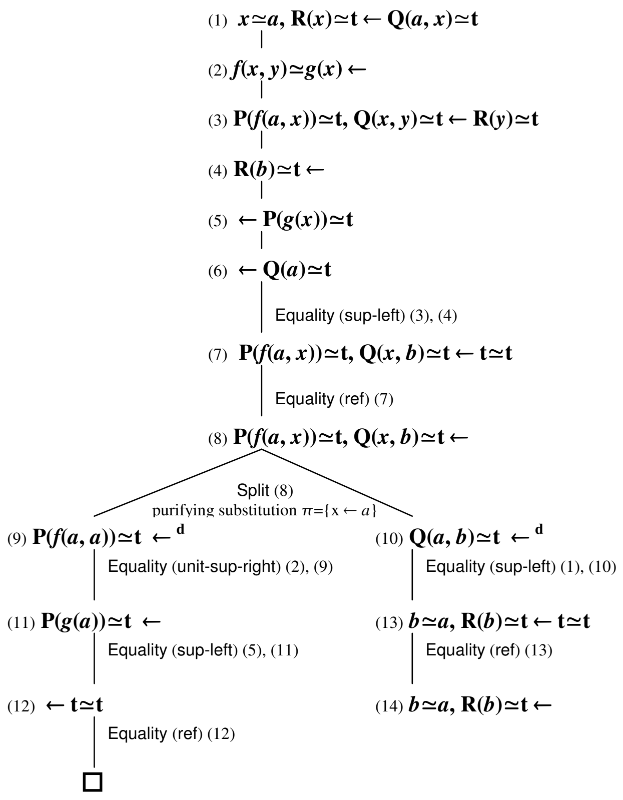
(1) x≃a, R(x)≃t ← Q(a, x)≃t
(2) f(x, y)≃g(x) ←
(3) P(f(a, x))≃t, Q(x, y)≃t ← R(y)≃t
(4) R(b)≃t ←
(5) ← P(g(x))≃t
(6) ← Q(a)≃t
Equality (sup-left) (3), (4)
(7) P(f(a, x))≃t, Q(x, b)≃t ← t≃t
Equality (ref) (7)
(8) P(f(a, x))≃t, Q(x, b)≃t ←
Split (8)
purifying substitution π={x ← a}
(9) P(f(a, a))≃t ← d
Equality (unit-sup-right) (2), (9)
(11) P(g(a))≃t ←
Equality (sup-left) (5), (11)
(12) ← t≃t
Equality (ref) (12)
□
(10) Q(a, b)≃t ← d
Equality (sup-left) (1), (10)
(13) b≃a, R(b)≃t ← t≃t
Equality (ref) (13)
(14) b≃a, R(b)≃t ←

Figure 1: **Example:** E-hyper tableau for given clauses (1) - (6)

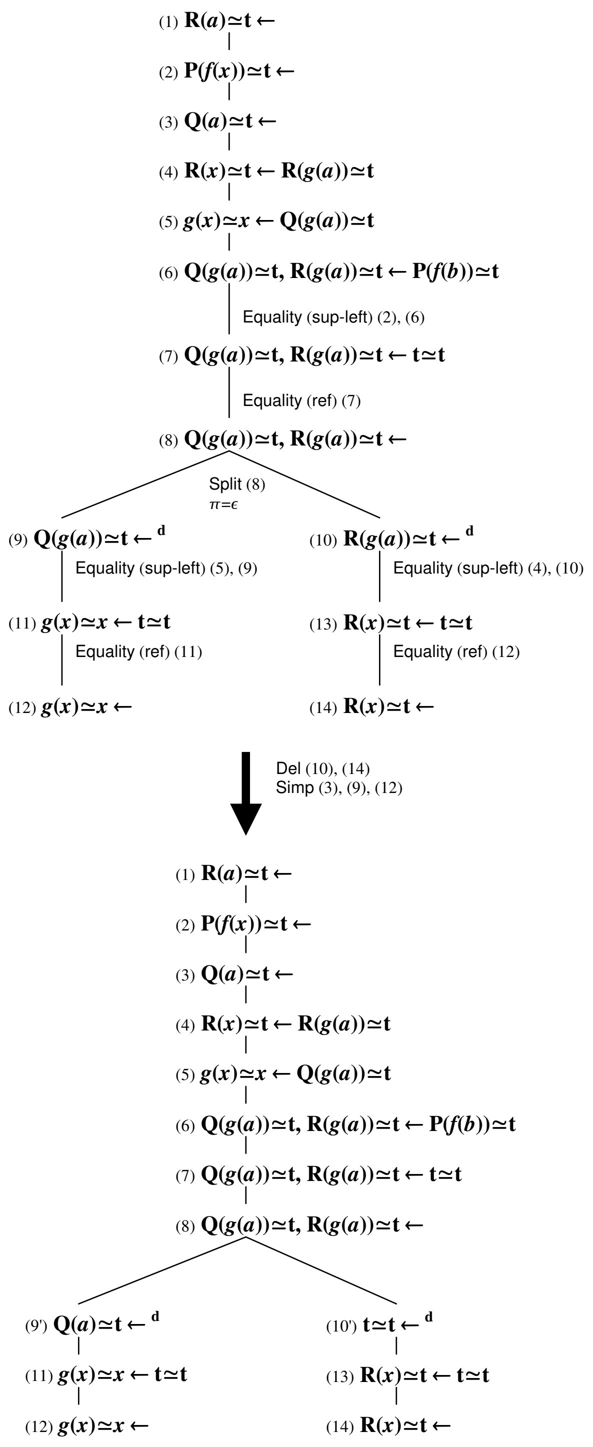

Figure 2: **Example:** E-hyper tableau for given clauses (1) - (6)

6 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 6.1 (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 $t \in \mathfrak{Ran}(\sigma)$[14], a unit clause $l \simeq r \leftarrow\ \in \mathcal{C}$ and a (matching) substitution μ such that $l\mu$ occurs in t and $l\mu \succ r\mu$. $\square$

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, and this is what is implemented in our system.

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 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 splits on clauses only that are *already* partitioned.

It is not clear a priori which of these approaches to splitting is preferrable in practice. An example where our approach is preferrable is as follows. Consider the 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. 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 f's, g's and (one) a, e.g. $\gamma = \{x/f(f(g(f(a))))\}$. Now, any ground instance of clause 5, for instance,

$$p(g(f(f(f(g(f(a))))))), p(g(g(f(f(g(f(a))))))) \leftarrow$$

[14]As usual, the *range* of a substitution σ is $\mathfrak{Ran}(\sigma) = \{x\sigma \mid x\sigma \neq x\}$.

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"

$$\mathsf{dom}(a) \leftarrow \tag{6}$$

$$\mathsf{dom}(f(x)) \leftarrow \mathsf{dom}(x) \tag{7}$$

$$\mathsf{dom}(g(x)) \leftarrow \mathsf{dom}(x) \tag{8}$$

which, in combination with rewriting by unit clauses will enumerate in its dom 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 dom-predicate.

In the following Section 7 we will build on the informal observations made here and devise a sound and complete calculus for *finite model computation* based on E-hyper taleaux.

7 Finite Model Computation

Finite model computation is the problem of computing an (E-)model of a given clause set with a finite domain (or detecting there is none).

Notice that as soon as the clause set contains one single non-zero arity function symbol, any finite model is necessarily not a Herbrand model. This indicates that standard theorem proving paradigms, which are ultimately based on Herbrand interpretations, cannot be used directly for finite model computation then. Indeed, methods for finite model computation can be classified as those that directly search for a finite model, like the extended PUHR tableau method [BT98], the methods in [Bez05, dNM06] and the methods in the SEM-family [Sla92, ZZ95, McC03], and those that are based on

[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.

transformations into (clausal) logic and which rely on readily available theorem provers or SAT solvers.

The latter approach includes the family of MACE-style model builders [McC03]. These systems search for finite models essentially by constructing a sequence of translations corresponding to interpretations with domain sizes $1, 2, \ldots$, in increasing order, until a model has been found. The model builder from this class with the best performance today is probably Paradox [CS03]. It is based on translation into propositional logic, and it uses a (very efficient) SAT solver to decide if there is a model of the current domain size.

However, there are several intrinsic problems with this and other approaches that are based on translation into propositional logic. One of them is lack of space efficiency, because the number of the ground instances of a clause grows exponentially in the number of variables in the clause [CS03]. (See [BFdNT07] for a more detailed discussion of this issue and examples of problematic clause sets.)

To address the space efficiency problem of MACE-style model computation, [BFdNT07] proposes to employ a first-order theorem prover instead of a propositional SAT solver. The target logic for the corresponding translation then is function-free clause logic (the theorem prover used there, the Darwin system [BFT06], is a decision procedure for that fragment). This allows to avoid the exponential growth of the clause set as the domain size increases, which is crucial for problems that have models of a relatively large size.

Yet, there remain other problems with the MACE approach, for instance lack of decent equality handling. Because equality reasoning is not supported natively by the underlying systems (naturally, equality reasoning does not apply to propositional SAT solving), equality is translated away. This involves removing the whole term structure by flattening all deep terms at the cost of introducing long clauses and turning each n-ary function symbol into a $n+1$-are predicate symbol. This way, all equations are translated away, too. This makes any built-in equational reasoning impossible then, such as simplification by rewriting, which is one of the crucial techniques for efficient equality reasoning. Using E-hyper tableau (or any other system supporting equational reasoning) is pointless then.

Notice that equality comes in even if the original problem is not equational. If the current domain size is n, then the translations into the target logic have to treat disjunctions of the form

$$x \simeq 1 \vee \cdots \vee x \simeq n \ .$$

This leads to the motivation for our approach presented below: we propose yet another MACE-style model finder, but this time using a theorem prover based on E-hyper tableau. This way, the exponential space requirements of the translation into propositional clause logic can be avoided while at the same time enabling more efficient equality treatment.

We present here only the theoretical core of of our approach. We introduce a transformation on clause sets related to the transformation into range-restricted form described at the end of Section 6, but that is adapted to enable finite model computation. More precisely, our transformation is sound and complete (cf. Proposition 7.2 below) and E-hyper tableau (or related calculi, like hyper-resolution with equality and splitting) can then be used to decide whether there is a model of a given size for a given clause set. This is our main result in this section.

In [BS06] somewhat related techniques are discussed. However, the method there is known to be *incomplete* for finite model computation. That is, it may fail to compute a finite model for a given clause set although there is one.

In [HW07] a streamlined version of the superposition calculus for finite model computation is described. Indeed, the ideas in both approaches are quite similar. Our approach can be seen as a simplified implementation of their approach. It is simpler in the sense that it is based on preprocessing and exploiting the properties of the calculus and standard redundancy criteria. Unlike [HW07] it does not require modifications to the prover itself. For fairness it has to be added, though, that some redundancy criteria that have been specifically designed in [HW07] will possibly be missed by our transformation.

7.1 Transformation

We define a transformation FD_d on clause sets. It is parametrized by a positive integer d, the *current domain size*. The transformation can be defined by a procedure carrying out the following steps.

(0) Initialization. Initially, let $\mathsf{FD}_d(\mathcal{C}) := \mathcal{C}$.

(1) Finite domain elements. Add to $\mathsf{FD}_d(\mathcal{C})$ the clauses

$$\begin{aligned} \mathsf{dom}(1) &\leftarrow \\ &\vdots \\ \mathsf{dom}(d) &\leftarrow \end{aligned} \tag{1}$$

$$\leftarrow i \simeq j \qquad \text{for all } i, j = 1, \ldots, d \text{ with } i < j \tag{2}$$

where $1, \ldots, d$ are fresh constants.

(2) Finite domain constraints. For each n-ary function symbol $f \in \mathcal{C}$, where $n \geq 0$, add to $\mathsf{FD}_d(\mathcal{C})$ the clause

$$f(x_1, \ldots, x_n) \simeq 1, \ldots, f(x_1, \ldots, x_n) \simeq d \leftarrow \quad . \tag{3}$$

(3) Range restriction. For each clause $\mathcal{A} \leftarrow \mathcal{B}$ in $\mathsf{FD}_d(\mathcal{C})$, let $\{x_1, \ldots, x_k\}$ be the set of variables occurring in $\mathcal{A}$ but not in $\mathcal{B}$. Replace $\mathcal{A} \leftarrow \mathcal{B}$ by the clause

$$\mathcal{A} \leftarrow \mathcal{B}, \mathsf{dom}(x_1), \ldots, \mathsf{dom}(x_k) \quad . \tag{4}$$

Example 7.1 Consider the following clause set $\mathcal{C}$:

$$P(a) \leftarrow \tag{C_1}$$

$$P(f(x)) \leftarrow P(x) \tag{C_2}$$

Of course it has finite models, even of size 1, but E-Hyper tableaux, like any "bottom-up" model generation method will fail to find any of them.

For the sake of illustration of our transformation let $d = 3$. Then, $\mathsf{FD}_3(\mathcal{C})$ consists of the clauses:

$$\mathsf{dom}(1) \leftarrow \qquad \mathsf{dom}(2) \leftarrow \qquad \mathsf{dom}(3) \leftarrow \tag{1}$$

$$\leftarrow 1 \simeq 2 \qquad \leftarrow 1 \simeq 3 \qquad \leftarrow 2 \simeq 3 \tag{2}$$

$$a \simeq 1, a \simeq 2, a \simeq 3 \leftarrow \tag{3-a}$$

$$f(x) \simeq 1, f(x) \simeq 2, f(x) \simeq 3 \leftarrow \mathsf{dom}(x) \tag{3-f}$$

$$P(a) \leftarrow \tag{4-C_1}$$

$$P(f(x)) \leftarrow P(x) \tag{4-C_2}$$

With appropriate simplification rules (see below), E-Hyper tableau derives a (finite) exhausted branch consisting of the unit clauses

$$\begin{array}{rr} \mathsf{dom}(1) \leftarrow & a \simeq 1 \leftarrow \\ \mathsf{dom}(2) \leftarrow & f(1) \simeq 1 \leftarrow \\ \mathsf{dom}(3) \leftarrow & f(2) \simeq 1 \leftarrow \\ p(1) \leftarrow & f(3) \simeq 1 \leftarrow \end{array}$$

□

Notice that our transformation does *not* start from a clause

$$x \simeq 1 \vee \cdots \vee x \simeq n$$

as mentioned above. Instead of our clauses (3), such a clause could be used in conjunction with the clauses

$$\mathsf{dom}(f(x_1, \ldots, x_n)) \leftarrow \mathsf{dom}(x_1), \ldots, \mathsf{dom}(x_n) \enspace ,$$

for each n-ary function symbol $f \in \mathcal{C}$.[16] But observe that E-Hyper tableau can then derive dom-literals with nested terms, such as $\mathsf{dom}(f(f(1)))$. These can always be simplified into non-nested ones (in the example, any branch containing $\mathsf{dom}(f(f(1)))$ must also contain $f(1) = j$, for some $1 \leq j \leq d$). However, it is preferable not to derive them in the first place. Indeed, the transformation as defined makes it *impossible* to derive dom-literals with nested terms.

[16]Such a translation then would be quite similar to the "classical" transformation into range-restricted form [MB88, BY00].

7.2 Correctness

The transformation FD_d is sound and complete wrt. finite model of size d in the following sense:

Proposition 7.2 (Correctness of FD_d)
Let $\mathcal{C}$ be a clause set and d a non-negative integer. Then $\mathcal{C}$ has a finite model with d domain elements if and only if $\mathsf{FD}_d(\mathcal{C})$ is E-satisfiable.

Notice that Proposition 7.2 alone is not sufficient for practical purposes. We still need to know that E-Hyper tableau provide a procedure to decide satisfiability of $\mathsf{FD}_d(\mathcal{C})$, for any clause set $\mathcal{C}$ and non-negative integer d. To this end, one can define a concrete fair strategy for derivations and prove that any derivation from $\mathsf{FD}_d(\mathcal{C})$ is finite. To obtain this result, a certain simplification technique has to be used. We refrain from laying out the details here, in particular as only standard techniques are needed, and instead we provide only a brief account.

Any fair derivation strategy will do, and the only redundancy elimination technique needed is "eager rewriting by positive branch equations". This will suffice to make infinite extension of branches impossible.

More precisely, observe that according to the scheme (4) in the definition of FD_d all clauses are range restricted. In particular, thus, initially all positive unit clauses are ground. Moreover, any new positive unit clause in any branch must be ground, too. This follows inductively together with the design of the inference rule: in essence, any Equality application then can only remove occurrences of variables in clauses, but never add new ones. In consequence then and together with range restriction, if all negative literals from a clause have been removed by enough Equality applications, any resulting positive clause must be ground. Either it is a positive (ground) unit clause already or Split turns it into some positive (ground) unit clauses, which are the only ways to obtain new positive clauses.

From the just obtained fact that all positive unit clauses in branches are ground it follows that there is no infinite sequence of applications of Equality inference rules to clause bodies. This follows because, as said, any Equality application (ref or sup-left) removes at least one occurrence of a variable, or otherwise it replaces a ground term by a smaller ground term. Thus, the only way an infinite branch could be constructed is by infinitely many Equality (unit-sup-right) or Split applications.

However, in both cases the selected clauses can be supposed to be maximally simplified by rewriting with the positive equations in the branch. By construction of FD_d, for every n-ary function symbol f in $\mathcal{C}$ and any n integers $i_1, \ldots, i_n$ with $1 \leq i_1, \ldots, i_n \leq d$, there is a k with $1 \leq k \leq d$ such that $f(i_1, \ldots, i_n) \simeq k \leftarrow$ is contained in the branch (cf. also the proof of proposition 7.2 above). Together, this suffices to rewrite any functional term into one of the constants $1, \ldots, d$. If such rewriting is applied exhaustively prior to adding new positive unit clauses it is clear that only finitely many different such clauses exist. No branch need thus be extended infinitely. In conclusion, E-Hyper tableaux (proof procedures) can be used as a decision procedure for finite satisfiability.

8 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*), 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: the tableau is only extended by such inference results which fall within the limit, and exceeding results are stored separately. If the tableau is exhausted and non-redundant exceeding results have been found, then the limit is raised and E-KRHyper backtracks to the point of the first weight transgression. This strategy 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.

E-KRHyper supports input clauses in a Prolog-like syntax. The system also accepts input in the common TPTP-syntax, both in clause normal form (CNF) and as first-order formulas (FOF). As an option, E-KRHyper can apply the finite model transformation (see Section 7). E-KRHyper is intended for embedding in knowledge-representation applications and has a number of features for this purpose, including logic extensions like stratified negation as failure, proof output for models and refutations as well as for partial results, as well as rapid switching and retraction of input clause sets for an efficient usage as a reasoning server. More details about the system can be found in

Table 1: Results for E-KRHyper on CNF Problems from the TPTP

status	*Horn*	*range restricted*	*No. of problems*	*solved*
satisfiable	yes	yes	26	20 (77%)
satisfiable	yes	no	239	55 (23%)
satisfiable	no	yes	89	66 (74%)
satisfiable	no	no	470	140 (30%)
unsatisfiable	yes	yes	122	122 (100%)
unsatisfiable	yes	no	2016	924 (46%)
unsatisfiable	no	yes	335	232 (69%)
unsatisfiable	no	no	2027	666 (33%)
open/unknown	both	both	953	0 (0%)
overall	both	both	6250	2225 (36%)

[PW07]; it is available under the GNU Public License from the E-KRHyper website at `http://www.uni-koblenz.de/~bpelzer/ekrhyper`.

We have tested E-KRHyper on the CNF-problems of the current TPTP version 3.3.0 [GS98], using the 360 seconds timeout limit of the most recent CASC system competition in 2007. No special transformations were applied to the problems. Table 1 summarizes the results for various subsets of problems. The status *open/unknown* indicates those problems that have not yet been solved by any theorem proving system. The most difficult problems solved by E-KRHyper are `SYN761-1.p`, `SYN788-1.p`, `SYN789-1.p` and `SYN792-1.p` with a TPTP-rating of 0.83. The average time for a successful proof is 10.4 seconds, and 77% of the proofs were found in less than one second.

The use of purifying substitutions may be necessary for those problems which occur in the subsets containing both non-Horn clauses and clauses which are not range restricted. Recognizing which problems need purification is non-trivial. This can be demonstrated with an example using the clause $C = p(x) \vee q(y) \leftarrow r(x, y)$, which is range-restricted and has no shared variables among its positive literals. Nevertheless, if at any point during a derivation a branch contains a unit $r(t, t)$ with t being a non-ground term, then an inference resulting in the positive disjunction $p(t) \vee q(t) \leftarrow$ is possible. This disjunction will require purification upon splitting. If on the other hand no such non-ground unit is ever derived, then the presence of C in a set of clauses will never make purification necessary. As the need for purification cannot be determined beforehand in an efficient manner, the domain required for purification must be enumerated for all problems where purification might become necessary.

Further optimization of E-KRHyper is necessary, in particular regarding purely equational problems. Experiments with preprocessing steps like the finite model transformation may also yield improvements. We consider this version of E-KRHyper a first step towards an efficiently applicable tableau prover with equality.

9 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. These are exploited to obtain a sound and complete procedure for finite model generation. To our knowledge, this is the first MACE-style approach to finite model computation that supports equality reasoning natively instead of translating it away. The calculus is implemented in the E-KRHyper system, an extension of our existing KRHyper prover.

References

[Bec97] Bernhard Beckert. Semantic tableaux with equality. *Journal of Logic and Computation*, 7(1):39–58, 1997.

[Bez05] M. Bezem. Disproving distributivity in lattices using geometry logic. In *Proc. CADE-20 Workshop on Disproving*, 2005.

[BF03] Peter Baumgartner and Ulrich Furbach. Automated Deduction Techniques for the Management of Personalized Documents. *Annals of Mathematics and Artificial Intelligence – Special Issue on Mathematical Knowledge Management*, 38(1), 2003.

[BFdNT07] Peter Baumgartner, Alexander Fuchs, Hans de Nivelle, and Cesare Tinelli. Computing finite models by reduction to function-free clause logic. *Journal of Applied Logic*, 2007. In Press.

[BFGHS04] Peter Baumgartner, Ulrich Furbach, Margret Gross-Hardt, and Alex Sinner. Living Book – Deduction, Slicing, and Interaction. *Journal of Automated Reasoning*, 32(3):259–286, 2004.

[BFN96] Peter Baumgartner, Ulrich Furbach, and Ilkka Niemelä. Hyper Tableaux. In *Proc. JELIA 96*, number 1126 in Lecture Notes in Artificial Intelligence. European Workshop on Logic in AI, Springer, 1996.

[BFP07] Peter Baumgartner, Ulrich Furbach, and Björn Pelzer. Hyper tableaux with equality. In Frank Pfenning, editor, *CADE-21 – The 21st International Conference on Automated Deduction*, volume 4603 of *Lecture Notes in Artificial Intelligence*, pages 492–507. Springer, 2007.

[BFT06] Peter Baumgartner, Alexander Fuchs, and Cesare Tinelli. Implementing the model evolution calculus. *International Journal of Artificial Intelligence Tools*, 15(1):21–52, 2006.

[BG98] Leo Bachmair and Harald Ganzinger. Chapter 11: Equational Reasoning in Saturation-Based Theorem Proving. In Wolfgang Bibel and Peter H. Schmitt, editors, *Automated Deduction. A Basis for Applications*, volume

I: Foundations. Calculi and Refinements, pages 353–398. Kluwer Academic Publishers, 1998.

[BS06] Peter Baumgartner and Renate Schmidt. Blocking and other enhancements for bottom-up model generation methods. In U. Furbach and N. Shankar, editors, *Automated Reasoning – Third International Joint Conference on Automated Reasoning (IJCAR)*, volume 4130 of *LNAI*. Springer, 2006.

[BT98] François Bry and Sunna Torge. A Deduction Method Complete for Refutation and Finite Satisfiability. In *Proc. 6th European Workshop on Logics in AI (JELIA)*, LNAI. Springer, 1998.

[BT05] Peter Baumgartner and Cesare Tinelli. The model evolution calculus with equality. In Robert Nieuwenhuis, editor, *CADE-20 – The 20th International Conference on Automated Deduction*, volume 3632 of *Lecture Notes in Artificial Intelligence*, pages 392–408. Springer, 2005.

[BY00] F. Bry and A. Yahya. Positive unit hyperresolution tableaux for minimal model generation. *J. Automated Reasoning*, 25(1):35–82, 2000.

[CS03] Koen Claessen and Niklas Sörensson. New techniques that improve MACE-style finite model building. In Peter Baumgartner and Christian G. Fermüller, editors, *CADE-19 Workshop: Model Computation – Principles, Algorithms, Applications*, 2003.

[dNM06] Hans de Nivelle and Jia Meng. Geometric resolution: A proof procedure based on finite model search. In U. Furbach and N. Shankar, editors, *Proc. International Joint Conference on Automated Reasoning (IJCAR)*, volume 4130 of *LNAI*. Springer, 2006.

[DV96] Anatoli Degtyarev and Andrei Voronkov. Equality elimination for the tableau method. In *Proceedings of the International Symposium on the Design and Implementation of Symbolic Computation Systems (DISCO-96)*, volume 1128 of *Lecture Notes in Artificial Intelligence*. Springer Verlag, Berlin, Heidelberg, New-York, 1996.

[DV98] Anatoli Degtyarev and Andrei Voronkov. What you always wanted to know about rigid E-unification. *J. Autom. Reasoning*, 20(1):47–80, 1998.

[FO06] Ulrich Furbach and Claudia Obermaier. Applications of automated reasoning. In *Proc. of the 29th Gertman Conference on AI*, LNAI 4314. Springer-Verlag, 2006.

[Gie01] Martin Giese. Incremental closure of free variable tableaux. In Rajeev Goré, Alexander Leitsch, and Tobias Nipkow, editors, *Proc. Intl. Joint Conf. on Automated Reasoning IJCAR, Siena, Italy*, volume 2083 of *LNCS*, pages 545–560. Springer-Verlag, 2001.

[Gie02] Martin Giese. A model generation style completeness proof for constraint tableaux with superposition. In Uwe Egly and Christian G. Fermüller, editors, *Proc. Intl. Conf. on Automated Reasoning with Analytic Tableaux and Related Methods, Copenhagen, Denmark*, volume 2381 of *LNCS*. Springer-Verlag, 2002.

[Gie03] Martin Giese. Simplification rules for constrained formula tableaux. In Marta Cialdea Mayer and Fiora Pirri, editors, *utomated Reasoning with Analytic Tableaux and Related Methods, International Conference, TABLEAUX 2003, Rome, Italy, September 9-12, 2003. Proceedings*, volume 2796 of *Lecture Notes in Computer Science*, pages 65–80. Springer-Verlag, 2003.

[GS98] Christian Suttner Geoff Sutcliffe. The TPTP problem library: CNF release v1.2.1. *Journal of Automated Reasoning*, 21(2):177–203, 1998.

[HW07] Thomas Hillenbrand and Christoph Weidenbach. Superposition for finite domains. Research Report MPI-I-2007-RG1-002, Max-Planck Institute for Informatics, Saarbruecken, Germany, April 2007.

[LMG94] R. Letz, K. Mayr, and C. Goller. Controlled Integrations of the Cut Rule into Connection Tableau Calculi. *Journal of Automated Reasoning*, 13, 1994.

[LS02] Reinhold Letz and Gernot Stenz. Integration of Equality Reasoning into the Disconnection Calculus. In Uwe Egly and Christian G. Fermüller, editors, *TABLEAUX*, volume 2381 of *Lecture Notes in Computer Science*, pages 176–190. Springer, 2002.

[MB88] Rainer Manthey and François Bry. SATCHMO: a theorem prover implemented in Prolog. In Ewing Lusk and Ross Overbeek, editors, *Proceedings of the 9th Conference on Automated Deduction, Argonne, Illinois, May 1988*, volume 310 of *Lecture Notes in Computer Science*, pages 415–434. Springer, 1988.

[McC92] William McCune. Experiments with discrimination-tree indexing and path indexing for term retrieval. *Journal of Automated Reasoning*, 9(2):147–167, October 1992.

[McC03] W. McCune. Mace4 reference manual and guide. Technical Report ANL/MCS-TM-264, Argonne National Laboratory, 2003.

[NR01] Robert Nieuwenhuis and Albert Rubio. Paramodulation-based theorem proving. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, pages 371–443. Elsevier and MIT Press, 2001.

[PW07] Björn Pelzer and Christoph Wernhard. System description: E-KRHyper. In Frank Pfenning, editor, *Automated Deduction - CADE-21, 21st International Conference on Automated Deduction, Bremen, Germany, July 17-20, 2007, Proceedings*, volume 4603 of *Lecture Notes in Computer Science*, pages 508–513. Springer, 2007.

[Sla92] John Slaney. Finder (finite domain enumerator): Notes and guide. Technical Report TR-ARP-1/92, Australian National University, Automated Reasoning Project, Canberra, 1992.

[SS06] Geoff Sutcliffe and Christian Suttner. The State of CASC. *AI Communications*, 19(1):35–48, 2006.

[Wei01] Christoph Weidenbach. Combining Superposition, Sorts and Splitting. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*. North Holland, 2001.

[ZZ95] Jian Zhang and Hantao Zhang. Sem: a system for enumerating models. In *IJCAI-95 — Proceedings of the 14th International Joint Conference on Artificial Intelligence, Montreal*, pages 298–303, 1995.

A Proofs

The general technique to prove the E-hyper tableau calculus complete is taken from the completeness proof of the superposition calculus [BG98, NR01], but adapted to our needs. One of the key concepts concerns the construction of a model of a clause set under certain conditions. That model constructed is presented as a (convergent) rewrite system. We will describe these concepts next.

A.1 Orderings and Rewrite Rules

Our approach makes heavy use of term rewrite systems and term orderings. We only mention here some details specific to our framework and refer to the literature [BG98, NR01, e.g.] for standard definitions otherwise.

We assume a reduction ordering $\succ$ that is total on ground Σ-terms. The non-strict ordering induced by $\succ$ is denoted by $\succeq$, and $\prec$ and $\preceq$ denote the converse of $\succ$ and $\succeq$, respectively.

A *(rewrite) rule* is an expression of the form $l \rightarrow r$ where l and r are Σ-terms. A *rewrite system* is a (possibly infinite) set of rewrite rules. A ground rewrite system R is *ordered by* $\succ$ iff $l \succ r$, for every rule $l \rightarrow r \in R$, and R is *lhs-irreducible* if it contains no two different rules of the forms $l \rightarrow r$ and $s[l] \rightarrow t$. In other words, no left hand side of a rule can be rewritten by another rule.

Notice that any ground rewrite system ordered by $\succ$ and without lhs-overlaps is a convergent ground rewrite system.[17] It is well known that for any convergent rewrite system R, and any two terms s and t, $R \models_{\mathrm{E}} s \simeq t$ if and only if they have the same normal, i.e. there is a term u such that $s \rightarrow_R^\star u$ and $t \rightarrow_R^\star u$ and u cannot be rewritten further. This result thus applies in particular to ground lhs-irreducible convergent rewrite systems.

In the sequel, the letter R will always denote a ground lhs-irreducible rewrite system.

By a slight abuse of notation we will write $R \models F$ for a ground rewrite system R and clause (set) F iff the interpretation $\{l \simeq r \mid l \rightarrow r \in R\}$ satisfies F (similarly for $R \models_{\mathrm{E}} F$).

A.2 Model Construction

This section presents the proof of Theorem 3.2 (Static Completeness). Let $\mathcal{C}$ be a (possibly infinite) set of clauses. (In the completeness proof $\mathcal{C}$ will be obtained as a certain limit branch of a tableau.) We show how $\mathcal{C}$ induces a ground lhs-irreducible rewrite system $R_{\mathcal{C}}$.

First, for a positive ground Σ-clause C we define by induction on the term ordering $\succ$ sets of rewrite rules ϵ_C and R_C as follows (we leave the parameter $\mathcal{C}$ implicit). Assume that ϵ_D has already been defined for all ground Σ-clauses D with $C \succ D$. With $R_C =$

[17] A *convergent* rewrite system is one that is confluent and terminating.

$\bigcup_{C \succ D} \epsilon_D$, we define

$$\epsilon_C = \begin{cases} \{l \to r\} & \text{if } C = l \simeq r \leftarrow \text{ is a ground instance of some positive} \\ & \text{unit clause in } \mathcal{C},\ l \succ r, \text{ and } l \text{ is irreducible wrt. } R_C \\ \emptyset & \text{otherwise} \end{cases}$$

Then $R_{\mathcal{C}} = \bigcup_C \epsilon_C$, where C ranges over all ground Σ-clauses.

By construction, $R_{\mathcal{C}}$ has no critical pairs, and is thus an lhs-irreducible rewrite system. Since $\succ$ is a well-founded ordering, $R_{\mathcal{C}}$ is a convergent rewrite system by construction. The given clause set $\mathcal{C}$ comes into play only in the first condition of the definition of ϵ_C. An important detail is that according to our convention the equations $s \simeq t$ and $t \simeq s$ are treated as the same. Thus, if $s \prec t$ then $s \simeq t \leftarrow$ may still be turned into the rewrite rule $t \to s$ in $R_{\mathcal{C}}$ by means of its symmetric version $t \simeq s \leftarrow$.

Observe that even if $\mathcal{C}$ is a set of positive unit clauses, then $R_{\mathcal{C}}$, even if convergent, may be incomplete wrt. the equational theory presented by it. For instance, with $\mathcal{C} = \{(a \simeq b \leftarrow\ ), (a \simeq c \leftarrow\ )\}$ and the ordering $a \succ b \succ c$ the induced rewrite system $R_{\mathcal{C}} = \{a \to c\}$ is clearly incomplete wrt. the equational theory $\{a \simeq b, a \simeq c\}$. In general then it might be necessary to add enough positive unit clauses to $\mathcal{C}$ to make $R_{\mathcal{C}}$ complete, which the E-hyper tableau calculus does. Positive non-unit clauses are handled differently, by splitting.

The following lemma states that satisfaction of a clause C in R_C is preserved as R_C is being extended.

Lemma A.1
Let $\mathcal{C}$ be a clause set, C a ground clause, and R and R' rewrite systems such that $R_C \subseteq R \subseteq R' \subseteq R_{\mathcal{C}}$. If $R \models_{\mathrm{E}} C$ then $R' \models_{\mathrm{E}} C$.

Proof. Writing C as the clause $\mathcal{A} \leftarrow \mathcal{B}$, we suppose $R \models_{\mathrm{E}} \mathcal{A} \leftarrow \mathcal{B}$ and show $R' \models_{\mathrm{E}} \mathcal{A} \leftarrow \mathcal{B}$.

If $R \models_{\mathrm{E}} \mathcal{B}$ (reading $\mathcal{B}$ as a conjunction of atoms) then with $R \models_{\mathrm{E}} \mathcal{A} \leftarrow \mathcal{B}$ it follows $R \models_{\mathrm{E}} A$, for some head atom A of $\mathcal{A} \leftarrow \mathcal{B}$. From monotonicity of first-order logic with equality, and with $R' \supseteq R$ it follows $R' \models_{\mathrm{E}} A$ and, trivially, $R' \models_{\mathrm{E}} \mathcal{A} \leftarrow \mathcal{B}$. Hence assume $R \not\models_{\mathrm{E}} \mathcal{B}$ from now on.

By way of contradiction assume $R' \models_{\mathrm{E}} \mathcal{B}$ but $R' \not\models_{\mathrm{E}} A$, for any head atom A (of $\mathcal{A} \leftarrow \mathcal{B}$). As $R' \models_{\mathrm{E}} \mathcal{B}$ holds while $R \models_{\mathrm{E}} \mathcal{B}$ does not hold, there is at least one body equation $s \simeq t$ in $\mathcal{B}$ such that $R' \models_{\mathrm{E}} s \simeq t$ but $R \not\models_{\mathrm{E}} s \simeq t$. Because $R_{\mathcal{C}}$ is convergent (this follows easily from its construction) and hence also its subsets R and R' are convergent, conclude that $s \simeq t$ is joinable by R' but not by R.

Every rule $l \to r \in R_{\mathcal{C}}$ is obtained from a ground instance $l \simeq r \leftarrow$ of a positive unit clause from the clause set $\mathcal{C}$. From $l \to r \in (R' \setminus R)$ and $R \supseteq R_C$ it follows $l \to r \notin R_C$. By definition of R_C then $(l \simeq r \leftarrow\ ) \succeq C$. (In fact even $(l \simeq r \leftarrow\ ) \succ C$ because these two clauses are different.) This entails that the head atom $l \simeq r$ (of the unit clause $l \simeq r \leftarrow$) is greater or equal than the body atom $s \simeq t$, i.e. $\{l, r\} \succeq \{s, s, t, t\}$. It follows that l is greater than even the maximum of s and t. But then it is impossible (essentially, by the subterm property of reduction orderings) that the rule $l \to r$ can be used to rewrite the term s or the term t. Because this holds for every rule in $(R' \setminus R)$, the R'- and R-normalforms of s and t are the same. This leads to a contradiction to the assumption that $R' \models_{\mathrm{E}} s \simeq t$ holds but $R \models_{\mathrm{E}} s \simeq t$ does not hold. Hence, the assumption that $R' \models_{\mathrm{E}} \mathcal{B}$ holds but $R_{\mathcal{C}} \models_{\mathrm{E}} A$ does not hold must be given up. This entails $R' \not\models_{\mathrm{E}} \mathcal{B}$ or $R' \models_{\mathrm{E}} A$, for some head atom A. Equivalently, $R' \models_{\mathrm{E}} \mathcal{A} \leftarrow \mathcal{B}$. □

Occasionally the following lemma comes in handy.

Lemma A.2
Let $\mathcal{C}$ be a clause set and C and D ground clauses. If $C \succ D$ then $R_D \cup \epsilon_D \subseteq R_C$

Proof. By definition $R_C = \bigcup_{C \succ E} \epsilon_E$ and $R_D = \bigcup_{D \succ E} \epsilon_E$. With $C \succ D$ it follows $\epsilon_D \subseteq R_C$ and $R_D \subseteq R_C$. Together, thus, $R_D \cup \epsilon_D \subseteq R_C$. □

Proposition A.3 (Model construction)
Let $\mathcal{C}$ be a clause set that is saturated up to redundancy and such that $\Box \notin \mathcal{C}$. Then, for every ground instance C of every clause from $\mathcal{C}$ the following holds:

1. *If $\mathcal{C}_C \models_{\mathrm{E}} C$ then $\epsilon_C = \emptyset$ and $R_C \models_{\mathrm{E}} C$.*
2. *If $\mathcal{C}_C \not\models_{\mathrm{E}} C$ then $R_C \cup \epsilon_C \models_{\mathrm{E}} C$.*

That is, either C is redundant wrt. $\mathcal{C}$ and R_C already satisfies C, or else, when C is not redundant wrt. $\mathcal{C}$, extension of R_C by ϵ_C will satisfy C. However, the case $\epsilon_C = \emptyset$ is possible. For example, when $C = \ \leftarrow a \simeq b$ and $\mathcal{C}_C = \emptyset$.

But, in either case the proposition gives $R_C \cup \epsilon_C \models_{\mathrm{E}} C$.

Proof. The claim is proved by well-founded induction on the ground instances of the clauses from $\mathcal{C}$. Hence choose arbitrarily any ground instance C of a clause from $\mathcal{C}$ and assume that the proposition holds for all ground instances D of all clauses from $\mathcal{C}$ such that $C \succ D$.

1. $\mathcal{C}_C \models_{\mathrm{E}} C$.
Regarding item 1, assume $\mathcal{C}_C \models_{\mathrm{E}} C$, i.e. C is redundant wrt. $\mathcal{C}$. By induction, combining cases 1 and 2, we get $R_D \cup \epsilon_D \models_{\mathrm{E}} D$, for every clause $D \in \mathcal{C}_C$. With Lemma A.2 conclude $R_D \cup \epsilon_D \subseteq R_C$, and with Lemma A.1 it follows $R_C \models_{\mathrm{E}} D$, for every clause $D \in \mathcal{C}_C$. Equivalently, $R_C \models_{\mathrm{E}} \mathcal{C}_C$. With $\mathcal{C}_C \models_{\mathrm{E}} C$ conclude $R_C \models_{\mathrm{E}} C$, as desired. This completes the proof of the second part of item 1.

To show $\epsilon_C = \emptyset$ assume, by contradiction, $\epsilon_C = \{l \to r\}$, where $C = l \simeq r \leftarrow$. Recall we have just shown $R_C \models C$. As for any convergent rewrite system, two (ground) terms are equal in the E-interpretation induced by R_C iff their normal forms wrt. R_C are the same. Applied to the situation here, this means that l and r have the same R_C-normal form. In particular, thus, some rule from R_C must be applicable to the larger term (wrt. $\succ$) of l and r, which is l. But then, by definition we have $\epsilon_C = \emptyset$, which is a plain contradiction. This completes the proof of the first item.

2. $\mathcal{C}_C \not\models_{\mathrm{E}} C$.

Turning to item 2, suppose from now on $\mathcal{C}_C \not\models_{\mathrm{E}} C$, i.e., C is not redundant wrt. $\mathcal{C}$. It follows that no clause $D \in \mathcal{C}$ that C is a ground instance of can be redundant wrt. $\mathcal{C}$ either. We use this fact below to enable using items 1-3 of Definition 3.1.

We distinguish various cases on the form of C, most of them leading to a contradiction, though, thus ruling out that these forms are possible (in fact, when C is of any of these forms it will be redundant wrt. $\mathcal{C}$). For the (two) non-contradictory subcases we will show $R_C \cup \epsilon_C \models_{\mathrm{E}} C$.

2-1. $C = (D[x])\gamma$ and $x\gamma$ is reducible wrt. R_C.
Suppose $C = D\gamma$, for some clause $D \in \mathcal{C}$ and some (grounding) substitution γ, such that D contains a variable x, i.e., $D = D[x]$, and $x\gamma$ is reducible wrt. R_C. That is, $x\gamma = x\gamma[l]$ for some rule $l \to r \in R_C$.

Let γ' be the substitution that is the same as γ, except for x, where we set $x\gamma' = x\gamma[r]$. That is, γ' is like γ but with the rewrite rule $l \to r$ applied to $x\gamma$. From $l \succ r$ it follows $D\gamma' \prec D\gamma$. By the induction hypothesis $R_{D\gamma'} \cup \epsilon_{D\gamma'} \models_{\mathrm{E}} D\gamma'$. From $D\gamma' \prec D\gamma$ conclude $R_{D\gamma'} \cup \epsilon_{D\gamma'} \subseteq R_{D\gamma}$. Together with Lemma A.1 it follows $R_{D\gamma} \models_{\mathrm{E}} D\gamma'$. Because of $l \to r \in R_C$, $D\gamma = C$ and by definition of γ' conclude with congruence $R_C \models_{\mathrm{E}} C$, a plain contradiction to $\mathcal{C}_C \not\models_{\mathrm{E}} C$ as assumed above.

2-2. $C = (\mathcal{A} \leftarrow s \simeq t, \mathcal{B})\gamma$ and $s\gamma = t\gamma$.
If $C = (\mathcal{A} \leftarrow s \simeq t, \mathcal{B})\gamma$, for some clause $(\mathcal{A} \leftarrow s \simeq t, \mathcal{B}) \in \mathcal{C}$ and grounding substitution γ, and $s\gamma = t\gamma$ then there is an inference $(\mathcal{A} \leftarrow s \simeq t, \mathcal{B}) \Rightarrow_{\mathsf{ref}(\sigma)} (\mathcal{A} \leftarrow \mathcal{B})\sigma$, where σ is a mgu of s and t (and there is a substitution δ such that $\gamma = \sigma\delta$).

Neither the clause $\mathcal{A} \leftarrow s \simeq t, \mathcal{B}$ nor the clause $(\mathcal{A} \leftarrow s \simeq t, \mathcal{B})\sigma$ is redundant wrt. $\mathcal{C}$. This follows trivially from the assumption of case 2, that their instance C is not redundant wrt. $\mathcal{C}$. By saturation (Definition 3.1-3), the inference $(\mathcal{A} \leftarrow s \simeq t, \mathcal{B}) \Rightarrow_{\mathsf{ref}(\sigma)} (\mathcal{A} \leftarrow \mathcal{B})\sigma$ is redundant wrt. $\mathcal{C}$. In particular, thus, its ground instance $C \Rightarrow_{\mathsf{ref}(\epsilon)} (\mathcal{A} \leftarrow \mathcal{B})\gamma$ is redundant wrt. $\mathcal{C}$. By definition of redundancy, $\mathcal{C}_C \models_{\mathrm{E}} (\mathcal{A} \leftarrow \mathcal{B})\gamma$. It follows trivially that $\mathcal{C}_C \models_{\mathrm{E}} C$, a plain contradiction to $\mathcal{C}_C \not\models_{\mathrm{E}} C$ as assumed above.

2-3. $C = (\mathcal{A} \leftarrow s \simeq t, \mathcal{B})\gamma$, $s\gamma \succ t\gamma$ and $s\gamma$ is irreducible wrt. R_C.
Assume $C = (\mathcal{A} \leftarrow s \simeq t, \mathcal{B})\gamma$ for some clause $(\mathcal{A} \leftarrow s \simeq t, \mathcal{B}) \in \mathcal{C}$ and grounding substitution γ. We may assume $s\gamma \neq t\gamma$ because otherwise case 2-2 applies. Without loss of generality let $s\gamma$ be the larger side of the equation $(s \simeq t)\gamma$, i.e. $s\gamma \succ t\gamma$. Assume further $s\gamma$ is irreducible wrt. R_C. This entails that $s\gamma$ and $t\gamma$ are not joinable wrt. R_C. Thus, $R_C \not\models_{\mathrm{E}} s\gamma \simeq t\gamma$, which trivially entails $R_C \models_{\mathrm{E}} C$. Finally, as C is not a positive unit clause we trivially have $\epsilon_C = \emptyset$, which concludes this case.

2-4. $C = (s \simeq t \leftarrow)\gamma$, $s\gamma \succ t\gamma$ and $s\gamma$ is irreducible wrt. R_C.
Assume $C = (s \simeq t \leftarrow)\gamma$ for some positive unit clause $(s \simeq t \leftarrow) \in \mathcal{C}$ and grounding substitution γ. We may assume $s\gamma \neq t\gamma$ because otherwise the claim follows trivially. Without loss of generality let $s\gamma$ be the larger side of the equation $(s \simeq t)\gamma$, i.e. $s\gamma \succ t\gamma$. Further assume that $s\gamma$ is irreducible wrt. R_C. Thus, $\epsilon_C = \{s\gamma \to t\gamma\}$, which trivially entails $R_C \cup \epsilon_C \models_{\mathrm{E}} C$.

2-5. $C = (A_1, \ldots, A_m \leftarrow)\gamma$, for some $m \geq 2$.

Assume $C = D\gamma$ for some positive non-unit clause $D = (A_1, \ldots, A_m \leftarrow\ ) \in \mathcal{C}$, where $m \geq 2$, and with grounding substitution γ. It is not difficult to see that γ can be obtained by composition of some purifying substitution π for D and some other substitution δ, i.e. $\gamma = \pi\delta$. Such a substitution π always exists, it could be γ itself.

Neither D nor $D\pi$ is redundant wrt. $\mathcal{C}$. This follows trivially from the assumption of case 2, that their instance $C = D\gamma = D\pi\delta$ is not redundant wrt. $\mathcal{C}$. By saturation (Definition 3.1-1) the inference $D \Rightarrow_{\mathsf{split}(\pi)} A_1\pi \leftarrow\ , \ldots, A_m\pi \leftarrow$ is redundant wrt. $\mathcal{C}$. In particular, thus, its ground instance $C \Rightarrow_{\mathsf{split}(\epsilon)} A_1\gamma \leftarrow\ , \ldots, A_m\gamma \leftarrow$ is redundant wrt. $\mathcal{C}$. By definition of redundancy, $\mathcal{C}_C \models_{\mathrm{E}} A_i\gamma \leftarrow$, for some i with $1 \leq i \leq m$. It follows trivially $\mathcal{C}_C \models_{\mathrm{E}} C$, a plain contradiction to $\mathcal{C}_C \not\models_{\mathrm{E}} C$ as assumed above.

2-6. $C = (D[s])\gamma$ and $s\gamma$ is reducible at a non-variable position.
To make the case analysis exhaustive assume that C does not fall into one of the cases 2-1 – 2-5. We further analyze the form C can take. Assume $C = D\gamma$ for some clause $D \in \mathcal{C}$ and grounding substitution γ

In the first case D has a non-empty body. Because the cases 2-2 and 2-3 are excluded, D can be written as $\mathcal{A} \leftarrow s \simeq t, \mathcal{B}$, where $s\gamma \succ t\gamma$ and $s\gamma$ is reducible wrt. R_C.

In the second case D has an empty body. Because the cases 2-4 and 2-5 are excluded, and we are given that $\mathcal{C}$ does not contain the empty clause, D must be a positive unit clause and can be written as $s \simeq t \leftarrow$, where $s\gamma \succ t\gamma$ and $s\gamma$ is reducible wrt. R_C.

Doing both cases together, consider any rule $l \to r \in R_C$ that rewrites $s\gamma$. Because case 2-1 is excluded, $l \to r$ does not rewrite $s\gamma$ at or below a variable position of s. That is, any position p such that $s\gamma[l]_p$ holds is a non-variable position of s.

We continue the proof doing both cases together.

By construction the rewrite rule $l \to r$ is obtained from a ground instance of some positive unit equation from $\mathcal{C}$. Let $E = l' \simeq r' \leftarrow$ be a fresh variant of that positive unit equation. Because it is fresh, we may assume γ has been extended so as to give $l'\gamma = l$ and $r'\gamma = r$.

We must have $C \succ E\gamma$ because otherwise $l'\gamma \to r'\gamma \in R_C$ (i.e., $l \to r \in R_C$) would be impossible. Therefore we can apply induction to $E\gamma$. If case 1 applies, i.e. $\mathcal{C}_{E\gamma} \models_{\mathrm{E}} E\gamma$ then $\epsilon_{E\gamma} = \emptyset$ and so $l'\gamma \to r'\gamma (= l \to r)$ could not be a rewrite rule in $R_{\mathcal{C}}$ and thus neither in R_C. Case 1 is thus impossible. Therefore we must have $\mathcal{C}_{E\gamma} \not\models_{\mathrm{E}} E\gamma$. In other words, $E\gamma$ is not redundant wrt. $\mathcal{C}$.

As said above, D can take two different forms. If D is of the form $\mathcal{A} \leftarrow s \simeq t, \mathcal{B}$ consider the ground $\mathsf{sup\text{-}left}$ inference

$$(\mathcal{A}\gamma \leftarrow s\gamma[l'\gamma]_p \simeq t\gamma, \mathcal{B}\gamma), E\gamma \Rightarrow_{\mathsf{sup\text{-}left}(\epsilon)} (\mathcal{A}\gamma \leftarrow s\gamma[r'\gamma]_p \simeq t\gamma, \mathcal{B}\gamma)\ . \tag{1}$$

Because p is a position of a non-variable term in s, say, l'', the $\mathsf{sup\text{-}left}$ inference

$$(\mathcal{A} \leftarrow s[l'']_p \simeq t, \mathcal{B}), E \Rightarrow_{\mathsf{sup\text{-}left}(\sigma)} (\mathcal{A} \leftarrow s[r']_p \simeq t, \mathcal{B})\sigma \tag{2}$$

exists, where σ is a mgu of l' and l'', and $\gamma = \sigma\delta$ for some substitution δ. The ground $\mathsf{sup\text{-}left}$ inference (1) then is a ground instance of the $\mathsf{sup\text{-}left}$ inference (2).
(*) Above we concluded that $E\gamma$ is not redundant wrt. $\mathcal{C}$. Therefore the more general clause $E\sigma$ cannot be redundant wrt. $\mathcal{C}$ either. A global assumption in case 2 is that D

is not redundant wrt. $\mathcal{C}$. By saturation (Definition 3.1-2) the inference (2) is redundant wrt. $\mathcal{C}$. In particular, thus, its ground instance (1) is redundant wrt. $\mathcal{C}$. For economy of notation let $F = \mathcal{A}\gamma \leftarrow s\gamma[r'\gamma]_p \simeq t\gamma, \mathcal{B}\gamma$ be the conclusion of the inference (1).

By definition of redundancy $\mathcal{C}_C \cup \{E\gamma\} \models_{\mathrm{E}} F$. By induction, combining cases 1 and 2, we get $R_G \cup \epsilon_G \models_{\mathrm{E}} G$, for every clause $G \in \mathcal{C}_C$. With Lemma A.2 conclude $R_G \cup \epsilon_G \subseteq R_C$, and with Lemma A.1 it follows $R_C \models_{\mathrm{E}} G$, for every clause $G \in \mathcal{C}_C$. Equivalently, $R_C \models_{\mathrm{E}} \mathcal{C}_C$.

Because $E\gamma = (l' \simeq r')\gamma$ is present as a rewrite rule $(l'\gamma \to r'\gamma) = (l \to r) \in R_C$ it trivially follows that $R_C \models_{\mathrm{E}} E\gamma$. Together with $R_C \models_{\mathrm{E}} \mathcal{C}_C$ and $\mathcal{C}_C \cup \{E\gamma\} \models_{\mathrm{E}} F$ (by redundancy of the inference, as mentioned above) conclude $R_C \models_{\mathrm{E}} F$. From $l \to r \in R_C$ conclude by congruence $R_C \models_{\mathrm{E}} C$, which is a plain contradiction to $\mathcal{C}_C \not\models_{\mathrm{E}} C$ as assumed to hold for case 2. This case is thus impossible.

Similarly, if D is of the form $s \simeq t \leftarrow$ consider the ground unit-sup-right inference

$$(s\gamma[l'\gamma]_p \simeq t\gamma \leftarrow\ ), E\gamma \Rightarrow_{\textsf{unit-sup-right}(\epsilon)} (s\gamma[r'\gamma]_p \simeq t\gamma \leftarrow\ )\ . \tag{3}$$

Because p is a position of a non-variable term in s, say, l'', the unit-sup-right inference

$$(s[l'']_p \simeq t \leftarrow\ ), E \Rightarrow_{\textsf{unit-sup-right}(\sigma)} (s[r']_p \simeq t \leftarrow\ )\sigma \tag{4}$$

exists, where σ is a mgu of l' and l'', and $\gamma = \sigma\delta$ for some substitution δ. The ground unit-sup-right inference (3) then is a ground instance of the unit-sup-right inference (4).

The rest of the proof of this case is the same as from (*) above and is omitted (obviously, $F = (s\gamma[r'\gamma]_p \simeq t\gamma \leftarrow\ )$ this time).

In conclusion, the case 2-6 is impossible, too. □

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.*

Proof. Suppose $\square \notin \mathcal{C}$. To show that $\mathcal{C}$ is E-satisfiable it suffices we show that $R_{\mathcal{C}}$ is an E-model of $\mathcal{C}$. For this, it suffices to show $R_{\mathcal{C}} \models_{\mathrm{E}} C\gamma$ for an arbitrarily chosen clause $C \in \mathcal{C}$ and an arbitrarily chosen grounding substitution γ for C. To prove $R_{\mathcal{C}} \models_{\mathrm{E}} C\gamma$, we first use Proposition A.3 and conclude $R_{C\gamma} \cup \epsilon_{C\gamma} \models_{\mathrm{E}} C\gamma$. From that, $R_{\mathcal{C}} \models_{\mathrm{E}} C\gamma$ follows immediately by Lemma A.1. □

A.3 Soundness

Lemma A.5
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.*

Proof. Let us first focus on the inference rules sup-left and unit-sup-right. Assume the premises of such a rule are E-satisfiable and let I be a E-model; from the axioms of congruence we can immediately conclude for both rules, that I is an E-model for the conclusion as well. For ref the claim follows directly from reflexivity.

For Equality the claim is an immediate consequence from the above. For Split assume that there is an E-model I for the premise $\mathbf{B}$. Let $A_1, \cdots, A_m \leftarrow$ be the selected clause from $\mathbf{B}$. Then I is an E-model for $(A_1, \cdots, A_m \leftarrow)\pi$ where π is the purifying subsitution for $A_1, \ldots, A_m$. $A_1\pi, \ldots, A_m\pi$ have no variables in common and all variables are implicitly universally quantified; hence $\forall(A_1\pi \vee \ldots \vee A_m\pi)$ is equivalent to $\forall A_1\pi \vee \ldots \vee \forall A_m\pi$ and we conclude that I is an E-model for $\forall A_1\pi \vee \ldots \vee \forall A_m\pi$.

Hence there is an E-model for one of $\mathbf{B} \cdot A_1\pi \leftarrow^{\mathrm{d}}, \ldots, \mathbf{B} \cdot A_m\pi \leftarrow^{\mathrm{d}}$.

For Del the claim obviously holds, and for Simp assume an E-model I for the premise. Let $C, D, \mathbf{B}$ and $\mathbf{B}_1$ as in the definition of Simp; from condition (1) there, $(\mathbf{B} \cdot C \cdot \mathbf{B}_1) \models_{\mathrm{E}} D$, we conclude that D also holds in I. Here we make also use of condition (3) in the definition of Simp, the rationale behind which was explained in Section 4.2. □

Theorem A.6 (Soundness of E-Hyper Tableaux)
Let $\mathcal{C}$ be a clause set that has a refutation. Then $\mathcal{C}$ is E-unsatisfiable.

Proof. Let $\mathbf{T}$ be the resulting closed tree of the refutation. From the contrapositive of Lemma A.5 we conclude that if a tree $\mathbf{T}_i$ of a derivation contains only E-unsatisfiable branches, this holds for its predecessor $\mathbf{T}_{i-1}$ as well. The final tableau T of the refutation clearly consists only of E-unsatisfiable branches and hence by induction of the length of the refutation (which is by definition a finite derivation), we can conclude that the initial tableau $\mathbf{T}_0$, which consists of one branch with the tableau clauses from C, is E-unsatisfiable. □

A.4 Completeness

Lemma A.7
Let C_1 and C_2 be ground clauses and $\mathcal{C}$ a set of ground clauses. If $(\mathbf{B}_j)_{C_1} \cup \mathcal{C} \models_{\mathrm{E}} C_2$ for some $j < \kappa$ then $(\mathbf{B}_\infty)_{C_1} \cup \mathcal{C} \models_{\mathrm{E}} C_2$.

Proof. The proof is by well-founded induction. Suppose the result to hold for all ground clauses C_1' such that $C_1' \prec C_1$.

Suppose $(\mathbf{B}_j)_{C_1} \cup \mathcal{C} \models_{\mathrm{E}} C_2$ holds for some $j < \kappa$. Let $\mathcal{D}'$ be a finite subset of $(\mathbf{B}_j)_{C_1}$ such that $\mathcal{D}' \cup \mathcal{C} \models_{\mathrm{E}} C_2$. Such a set $\mathcal{D}'$ exists by compactness of first-order logic with equality.

The first step is to deal with Del applications applied to $\mathbf{B}_j, \mathbf{B}_{j+1}, \ldots$ that remove a clause by non-proper subsumption that can be instantiated to a clause in $\mathcal{D}'$. To trace such applications, let initially $\mathcal{D} = \mathbf{B}_j$. Then, for all $j, j+1, \ldots$, if Del is applied to $\mathbf{B}_j$ to remove a clause by non-proper subsumption that is also in $\mathcal{D}$ then replace in $\mathcal{D}$ the removed clause by the non-proper subsuming clause of that step. It is easy to see that this process maintains the invariant $\mathcal{D}' \subseteq \mathcal{D}_{C_1}$. With $\mathcal{D}' \cup \mathcal{C} \models_{\mathrm{E}} C_2$ this entails $\mathcal{D}_{C_1} \cup \mathcal{C} \models_{\mathrm{E}} C_2$.

It is a simple inductive consequence of the definition of the Split and Equality derivation rules that no clause set derived can contain a clause and a variant of it. Hence, in the considered Del application the non-proper subsuming clause and the subsumed clause cannot be variants. Because the ordering based on the converse relation, proper generalization, is well-founded, there is a time l such that no clause that is also in $\mathcal{D}$ is removed by non-proper subsumption deletion from $\mathbf{B}_l, \mathbf{B}_{l+1}, \ldots$. (It is possible that clauses from $\mathcal{D}$ do not occur in these branches at all, because they have been removed by some prior Del or Simp application.) Together with the definition of $\mathcal{D}$ this implies that $\mathcal{D}$ is a set of clauses so that no non-proper subsumption Del step is ever applied to any of them.

If $\mathcal{D}_{C_1} \subseteq (\mathbf{B}_\infty)_{C_1}$ then the claim follows from the monotonicity of first-order logic with equality and $\mathcal{D}_{C_1} \cup \mathcal{C} \models_{\mathrm{E}} C_2$, as concluded above.

Otherwise let $\mathbf{B}' := \mathcal{D}' \setminus (\mathbf{B}_\infty)_{C_1}$ be those clauses from $\mathcal{D}'$ that are not an instance of any persisting clause in $\mathbf{B}_\infty$. Choose any clause $C' \in \mathbf{B}'$ arbitrarily. By construction, it is a ground instance of some clause $C \in \mathcal{D}$ such that $C \notin \mathbf{B}_\infty$. This means that C has been removed from the clause set $\mathbf{B}_k$ labeling the node $\mathbf{N}_k$ of the branch $\mathbf{B}$, for some $k < \kappa$. In other words, the Del or Simp derivation rule has been applied to $\mathbf{B}_k$ with selected clause C. However, as argued above, by the definition of $\mathcal{D}$ this cannot have been a non-proper subsumption deletion Del application.

Hence, by definition of the Del and Simp derivation rules, the clause C, and hence its instance C' is redundant wrt. a specific subset $\mathbf{B}'' \subseteq \mathbf{B}_{k+1}$.[18] That subset $\mathbf{B}''$ is specified in the definition of the Del and Simp derivation rules. For our purpose the only important fact is that with $\mathbf{B}'' \subseteq \mathbf{B}_{k+1}$ it (trivially) follows that C' is redundant wrt. $\mathbf{B}_{k+1}$ as well.

That C' is redundant wrt. $\mathbf{B}_{k+1}$ means by definition of redundancy $(\mathbf{B}_{k+1})_{C'} \models_{\mathrm{E}} C'$. This implies by monotonicity of first-order logic with equality $(\mathbf{B}_{k+1})_{C'} \cup \mathcal{C} \models_{\mathrm{E}} C'$. With $C' \in \mathbf{B}' \subseteq \mathcal{D}' \subseteq \mathcal{D}_{C_1}$ it follows $C' \prec C_1$. By the induction hypothesis then

$$(\mathbf{B}_\infty)_{C'} \cup \mathcal{C} \models_{\mathrm{E}} C' \ . \tag{5}$$

From $C' \prec C_1$ it easily follows that $(\mathbf{B}_\infty)_{C'} \subseteq (\mathbf{B}_\infty)_{C_1}$. Together with (5) and by monotonicity of first-order logic with equality it follows

$$(\mathbf{B}_\infty)_{C_1} \cup \mathcal{C} \models_{\mathrm{E}} C' \ . \tag{6}$$

Recall from above $\mathcal{D}' \cup \mathcal{C} \models_{\mathrm{E}} C_2$. Because $C' \in \mathcal{D}'$ we can replace in this entailment C' in $\mathcal{D}'$ by the stronger set $(\mathbf{B}_\infty)_{C_1} \cup \mathcal{C}$. More formally, $\mathcal{D}' \cup \mathcal{C} \models_{\mathrm{E}} C_2$ and (6) entail

$$((\mathbf{B}_\infty)_{C_1} \cup \mathcal{C}) \cup (\mathcal{D}' \setminus \{C'\}) \cup \mathcal{C} \models_{\mathrm{E}} C_2 \ . \tag{7}$$

Repeating this procedure for each of the (finitely many) members of $\mathbf{B}'$ allows to conclude

$$((\mathbf{B}_\infty)_{C_1} \cup \mathcal{C}) \cup (\mathcal{D}' \setminus \mathbf{B}') \cup \mathcal{C} \models_{\mathrm{E}} C_2 \ . \tag{8}$$

Recall that $\mathbf{B}' = \mathcal{D}' \setminus (\mathbf{B}_\infty)_{C_1}$, which implies by elementary set theory $\mathcal{D}' \setminus \mathbf{B}' \subseteq (\mathbf{B}_\infty)_{C_1}$. But then, $(\mathbf{B}_\infty)_{C_1} \cup \mathcal{C} \models_{\mathrm{E}} C_2$ follows from (8) immediately. □

[18]This argument uses the fact that Del and Simp can be applied to clauses within the same “decision level” only, as explained in Section 4.2. Formally, the conditions (2) and (3) in Del and Simp, respectively, ensure that these rules do not touch a clause in a *different* branch in the derivation tree, a branch that would not provide the required justification by redundancy or non-proper subsumption, as stated in the other applicability conditions of Del and and Simp.

Lemma A.8
If C is redundant wrt. $\mathbf{B}_j$, for some $j < \kappa$ then C is redundant wrt. $\mathbf{B}_\infty$.

Proof. Suppose C is redundant wrt. $\mathbf{B}_j$, for some $j < \kappa$. Let D be an arbitrarily chosen ground instance of C. By definition, D is redundant wrt. $\mathbf{B}_j$, which means $(\mathbf{B}_j)_D \models_{\mathrm{E}} D$. With Lemma A.7 it follows $(\mathbf{B}_\infty)_D \models_{\mathrm{E}} D$. In other words D is redundant wrt. $\mathbf{B}_\infty$. Because D was chosen as an arbitrary ground instance of C, C is redundant wrt. $\mathbf{B}_\infty$. □

Lemma A.9
If $R \models_{\mathrm{E}} \mathcal{C}$ and C is redundant wrt. $\mathcal{C}$ then $R \models_{\mathrm{E}} C$.

Proof. Suppose $R \models_{\mathrm{E}} \mathcal{C}$ and C is redundant wrt. $\mathcal{C}$. Let D be an arbitrarily chosen ground instance of C. It suffices to show $R \models_{\mathrm{E}} D$. Since C is redundant wrt. $\mathcal{C}$, by definition, its ground instance D is redundant wrt. $\mathcal{C}$. Equivalently, $\mathcal{C}_D \models_R D$, which entails $R \models_{\mathrm{E}} D$ provided $R \models_{\mathrm{E}} \mathcal{C}_D$ holds. The latter however follows immediately from $R \models_{\mathrm{E}} \mathcal{C}$ and the trivial fact that $\mathcal{C}_D$ is a subset of the set of all ground instances of all clauses from $\mathcal{C}$. □

Lemma A.10
Let C be a clause and D a positive unit clause. Then, any inference $C, D \Rightarrow_{R(\sigma)} E$, where $R \in \{\textsf{sup-left}, \textsf{unit-sup-right}\}$, or $C \Rightarrow_{\textsf{ref}(\sigma)} E$ that is redundant wrt. $\mathbf{B}_j$, for some $j < \kappa$, is redundant wrt. $\mathbf{B}_\infty$.

Proof. Suppose an inference $C, D \Rightarrow_{R(\sigma)} E$, where $R \in \{\textsf{sup-left}, \textsf{unit-sup-right}\}$, redundant wrt. $\mathbf{B}_j$, for some $j < \kappa$. Let γ be an arbitrary ground substitution for C and D such that $\gamma = \sigma\delta$ for some substitution δ and such that $C\gamma \Rightarrow_{R(\epsilon)} E\delta$ is a ground instance of $C, D \Rightarrow_{R(\sigma)} E$. Because chosen arbitrarily, it suffices to show that this ground instance $C\gamma \Rightarrow_{R(\epsilon)} E\delta$ is redundant wrt. $\mathbf{B}_\infty$.

Because the inference $C, D \Rightarrow_{R(\sigma)} E$ is redundant wrt. $\mathbf{B}_j$, its instance $C\gamma \Rightarrow_{R(\epsilon)} E\delta$ is redundant wrt. $\mathbf{B}_j$. By definition of redundancy this means

$$(\mathbf{B}_j)_{C\gamma} \cup \{D\gamma\} \models_{\mathrm{E}} E\delta \ . \tag{9}$$

By Lemma A.7 then

$$(\mathbf{B}_\infty)_{C\gamma} \cup \{D\gamma\} \models_{\mathrm{E}} E\delta \ , \tag{10}$$

which, by definition, means that the inference $C\gamma \Rightarrow_{R(\epsilon)} E\delta$ is redundant wrt. $\mathbf{B}_\infty$, which was to be shown.

The proof of the case of an inference $C \Rightarrow_{\textsf{ref}(\sigma)} E$ is similar and is omitted. □

Lemma A.11
Let C be a positive clause and π a purifying substitution for C. If the inference $C \Rightarrow_{\textsf{split}(\pi)} A_1 \leftarrow \ , \ldots , A_m \leftarrow$ is redundant wrt. $\mathbf{B}_j$, for some $j < \kappa$, then it is redundant wrt. $\mathbf{B}_\infty$.

Proof. Suppose an inference $C \Rightarrow_{\mathsf{split}(\pi)} A_1 \leftarrow, \ldots, A_m \leftarrow$ that is redundant wrt. $\mathbf{B}_j$, for some $j < \kappa$. Let γ be an arbitrary ground substitution for C such that $\gamma = \pi\delta$ for some substitution δ and such that $C\gamma \Rightarrow_{\mathsf{split}(\epsilon)} A_1\delta \leftarrow, \ldots, A_m\delta \leftarrow$ is a ground instance of $C \Rightarrow_{\mathsf{split}(\pi)} A_1 \leftarrow, \ldots, A_m \leftarrow$. Because chosen arbitrarily, it suffices to show that the ground inference $C\gamma \Rightarrow_{\mathsf{split}(\epsilon)} A_1\delta \leftarrow, \ldots, A_m\delta \leftarrow$ is redundant wrt. $\mathbf{B}_\infty$.

Because the inference $C \Rightarrow_{\mathsf{split}(\pi)} A_1 \leftarrow, \ldots, A_m \leftarrow$ is redundant wrt. $\mathbf{B}_j$, its instance $C\gamma \Rightarrow_{\mathsf{split}(\epsilon)} A_1\delta \leftarrow, \ldots, A_m\delta \leftarrow$ is redundant wrt. $\mathbf{B}_j$. By definition of redundancy this means that $A_i\delta \leftarrow$ is redundant wrt. $\mathbf{B}_j$, for some i with $1 \leq i \leq m$. By Lemma A.8 then $A_i\delta \leftarrow$ is redundant wrt. $\mathbf{B}_\infty$. It follows immediately that the ground inference $C\gamma \Rightarrow_{\mathsf{split}(\epsilon)} A_1\delta \leftarrow, \ldots, A_m\delta \leftarrow$ is redundant wrt. $\mathbf{B}_\infty$, which remained to be shown. □

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.*

Proof. Suppose $\mathbf{B}$ is an exhausted branch of a limit tree of some fair derivation. According to Definition 3.1 it suffices to choose arbitrarily a clause $C \in \mathbf{B}_\infty$ that is not redundant wrt. $\mathbf{B}_\infty$ and to prove the properties 1-3 claimed there for C.

Before doing that, notice that if there is a $j < \kappa$ such that C is redundant wrt. $\mathbf{B}_j$, then by Lemma A.8 the clause C is redundant wrt. $\mathbf{B}_\infty$ and nothing remains to be shown for C. Hence suppose from now on that C is not redundant wrt. $\mathbf{B}_j$, for all $j < \kappa$.

1. $C \Rightarrow_{\mathsf{split}(\pi)} A_1 \leftarrow, \ldots, A_m \leftarrow$
Suppose there is an inference $C \Rightarrow_{\mathsf{split}(\pi)} A_1 \leftarrow, \ldots, A_m \leftarrow$. It suffices to show that this inference is redundant wrt. $\mathbf{B}_\infty$, or that $C\pi$ is redundant wrt. $\mathbf{B}_\infty$.

If there is a $j < \kappa$ such that $C\pi$ is redundant wrt. $\mathbf{B}_j$, then by Lemma A.8 $C\pi$ is redundant wrt. $\mathbf{B}_\infty$, and nothing remains to be shown. Hence suppose that $C\pi$ is not redundant wrt. $\mathbf{B}_j$, for all $j < \kappa$.

It suffices to show that an arbitrarily chosen ground instance of the inference $C \Rightarrow_{\mathsf{split}(\pi)} A_1 \leftarrow, \ldots, A_m \leftarrow$ is redundant wrt. $\mathbf{B}_\infty$. Hence let γ be an arbitrary ground substitution for C such that $\gamma = \pi\delta$ for some substitution δ, and such that $C\gamma \Rightarrow_{\mathsf{split}(\epsilon)} A_1\delta \leftarrow, \ldots, A_m\delta \leftarrow$ is a ground instance of the inference $C \Rightarrow_{\mathsf{split}(\pi)} A_1 \leftarrow, \ldots, A_m \leftarrow$. We will show that this ground inference is redundant wrt. $\mathbf{B}_\infty$.

From $C \in \mathbf{B}_\infty$ it follows there is an $i < \kappa$ such that for all $j \geq i$ with $j < \kappa$ it holds $C \in \mathbf{B}_j$. Because of $C \Rightarrow_{\mathsf{split}(\pi)} A_1 \leftarrow, \ldots, A_m \leftarrow$ Split is applicable (in particular) to $\mathbf{B}_i$ with underlying inference $C \Rightarrow_{\mathsf{split}(\pi)} A_1 \leftarrow, \ldots, A_m \leftarrow$ unless $A_j \leftarrow$, for some j with $i \leq j \leq m$ is contained as a variant in $\mathbf{B}_i$. In this case, by virtue of the ground instance $A_j\delta \leftarrow$ of $A_j \leftarrow$ it follows that the ground instance $C\gamma \Rightarrow_{\mathsf{split}(\epsilon)} A_1\delta \leftarrow, \ldots, A_m\delta \leftarrow$ is redundant wrt. $\mathbf{B}_i$ and nothing remains to be shown.

Recall we are considering the case that $C\pi$ is not redundant wrt. $\mathbf{B}_j$, for every $j < \kappa$.

But then, by Definition 4.2-1 there is a $k < \kappa$ such that the inference $C \Rightarrow_{\mathsf{split}(\pi)} A_1 \leftarrow, \ldots, A_m \leftarrow$ is redundant wrt. $\mathbf{B}_k$. By Lemma A.11 then, this inference is redundant wrt. $\mathbf{B}_\infty$. Therefore, in particular its (ground) instance $C\gamma \Rightarrow_{\mathsf{split}(\epsilon)} A_1\delta \leftarrow, \ldots, A_m\delta \leftarrow$ is redundant wrt. $\mathbf{B}_\infty$, which remained to be shown.

2. $C, D \Rightarrow_{R(\sigma)} E$*, where* $R \in \{\mathsf{sup\text{-}left}, \mathsf{unit\text{-}sup\text{-}right}\}$
This case is concerned with Equality inferences. More precisely, suppose there is an inference $C, D \Rightarrow_{R(\sigma)} E$, where $R \in \{\mathsf{sup\text{-}left}, \mathsf{unit\text{-}sup\text{-}right}\}$ and where D is a fresh variant of a positive unit clause from $\mathbf{B}_\infty$ and σ is some substitution.

It suffices to show that this inference is redundant wrt. $\mathbf{B}_\infty$, or that $C\sigma$ or $D\sigma$ is redundant wrt. $\mathbf{B}_\infty$.

If there is a $j < \kappa$ such that $C\sigma$ is redundant wrt. $\mathbf{B}_j$, then by Lemma A.8 $C\sigma$ is redundant wrt. $\mathbf{B}_\infty$, and nothing remains to be shown. Hence suppose that $C\sigma$ is not redundant wrt. $\mathbf{B}_j$, for all $j < \kappa$. By exactly the same argumentation, this time applied to $D\sigma$, we may assume that $D\sigma$ is not redundant wrt. $\mathbf{B}_j$, for all $j < \kappa$.

It suffices to show that an arbitrarily chosen ground instance of the inference $C, D \Rightarrow_{R(\sigma)} E$ is redundant wrt. $\mathbf{B}_\infty$. Hence let γ be an arbitrary ground substitution for C and D such that $\gamma = \sigma\delta$ for some substitution δ, and such that $C\gamma, D\gamma \Rightarrow_{R(\epsilon)} E\delta$ is a ground instance of of the inference $C, D \Rightarrow_{R(\sigma)} E$. Hence we will show that this ground inference $C\gamma, D\gamma \Rightarrow_{R(\epsilon)} E\delta$ is redundant wrt. $\mathbf{B}_\infty$.

From $C \in \mathbf{B}_\infty$ it follows there is an $i < \kappa$ such that for all $j \geq i$ with $j < \kappa$ it holds $C \in \mathbf{B}_j$. Likewise, from D being a variant of a clause in $\mathbf{B}_\infty$ it follows there is an i' such that for all $j' \geq i'$ it holds D is a variant of a clause in $\mathbf{B}_{j'}$. Without loss of generality assume $i \geq i'$. It follows D is a variant of a clause in $\mathbf{B}_j$, for all $j \geq i$.

From the just said, and because of $C, D \Rightarrow_{R(\sigma)} E$, Equality is applicable (in particular) to $\mathbf{B}_i$ with underlying inference $C, D \Rightarrow_{R(\sigma)} E$ unless E is contained as a variant in $\mathbf{B}_i$. In this case, the inference $C, D \Rightarrow_{R(\sigma)} E$ is redundant wrt. $\mathbf{B}_i$ and nothing remains to be shown.

Recall that we are currently considering the case of neither $C\sigma$ nor $D\sigma$ being redundant wrt. $\mathbf{B}_j$, for every $j < \kappa$.

But then, by Definition 4.2-2 there is a $k < \kappa$ such that the inference $C, D \Rightarrow_{R(\sigma)} E$ is redundant wrt. $\mathbf{B}_k$. By Lemma A.10 then, this inference is also redundant wrt. $\mathbf{B}_\infty$. Therefore, in particular its (ground) instance $C\gamma, D\gamma \Rightarrow_{R(\epsilon)} E\delta$ is redundant wrt. $\mathbf{B}_\infty$, which remained to be shown.

3. $C \Rightarrow_{\mathsf{ref}(\sigma)} E$
This case is concerned with an Equality inference, more precisely with an application of the ref rule. The proof is done analogously to case 2 and is omitted.

□

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

Proof. Suppose that $\mathbf{D}$ is not a refutation. Therefore its limit tree $\mathbf{T}$ has an exhausted branch. Let $\mathbf{B}$ be any such exhausted branch.

By Proposition 4.4 the clause set $\mathbf{B}_\infty$ is saturated up to redundancy. Moreover, $\mathbf{B}_\infty$ cannot contain the empty clause, because if it did, then $\mathbf{B}$ would also contain the empty clause, but no exhausted branch can contain the empty clause.

With Theorem 3.2 it follows $\mathbf{B}_\infty$ is satisfiable. Moreover, the proof of Theorem 3.2 gives us a convergent rewrite system $R_{\mathbf{B}_\infty}$ such that $R_{\mathbf{B}_\infty} \models_{\mathrm{E}} \mathbf{B}_\infty$.

To prove the theorem it suffices to show $R_{\mathbf{B}_\infty} \models_{\mathrm{E}} \mathcal{C}$. To show that, let C be any clause from $\mathcal{C}$, and it suffices to show $R_{\mathbf{B}_\infty} \models_{\mathrm{E}} C$. By definition of derivation, $C \in \mathbf{B}_0$, where $\mathbf{B}_0$ is the (single) branch of the initial tableau $\mathbf{T}_0$ of the derivation $\mathbf{D}$.

As a first step we trace possible Del applications that remove C by non-proper subsumption. More formally, let $C_0 = C$, and for all $j = 0, 1, \ldots$, if Del is applied to $\mathbf{B}_j$ with selected clause C_j then let C_{j+1} be the non-proper subsuming clause of that Del application, and otherwise let $C_{j+1} = C_j$. Clearly, to show $R_{\mathbf{B}_\infty} \models_{\mathrm{E}} C$ it suffices to show $R_{\mathbf{B}_\infty} \models_{\mathrm{E}} C_j$, for any $j \geq 0$.

As an easy inductive consequence of the definition of the Split and Equality derivation rules, no clause set $\mathbf{B}_0, \mathbf{B}_1, \ldots$ ever derived can contain both a clause and a variant of it. Hence, if C_j is removed by non-proper subsumption, C_j must be a proper instance of C_{j+1}. Because the ordering based on the converse relation, proper generalization, is well-founded, there is a time l such that C_l is a tableau clause in $\mathbf{B}_l$ (the case $l = 0$ is possible) that is not removed from any branch $\mathbf{B}_l, \mathbf{B}_{l+1}, \ldots$ by non-proper subsumption (C_l it could be removed by some other Del or Simp application, though).

If $C_l \in \mathbf{B}_\infty$ then with $R_{\mathbf{B}_\infty} \models_{\mathrm{E}} \mathbf{B}_\infty$ immediately conclude $R_{\mathbf{B}_\infty} \models_{\mathrm{E}} C_l$, and so $R_{\mathbf{B}_\infty} \models_{\mathrm{E}} C$. Hence suppose $C_l \notin \mathbf{B}_\infty$ from now on.

From $C_l \in \mathbf{B}_l$ and $C_l \notin \mathbf{B}_\infty$ it follows that C_l has been removed at some time $k < \kappa$ from the clause set $\mathbf{B}_k$ by an application of the Del or the Simp derivation rule. We deal with both cases at once. But notice we have already excluded the possibility that C_l has been removed in a Del application by means of non-proper subsumption.

This means, C_l is redundant wrt. a specific subset $\mathbf{B}'$ of the derived branch $\mathbf{B}_{k+1}$, where $\mathbf{B}'$ is specified in the definition of the Del and Simp derivation rules.[19] Because $\mathbf{B}' \subseteq \mathbf{B}_{k+1}$ it trivially follows that C_l is redundant wrt. $\mathbf{B}_{k+1}$.

By Lemma A.8, C_l is redundant wrt. $\mathbf{B}_\infty$. With $R_{\mathbf{B}_\infty} \models_{\mathrm{E}} \mathbf{B}_\infty$ and Lemma A.9 it follows $R_{\mathbf{B}_\infty} \models_{\mathrm{E}} C_l$, and so $R_{\mathbf{B}_\infty} \models_{\mathrm{E}} C$, which remained to be shown. □

Proposition 7.2 (Correctness of FD_d)
Let $\mathcal{C}$ be a clause set and d a non-negative integer. Then $\mathcal{C}$ has a finite model with d domain elements if and only if $\mathsf{FD}_d(\mathcal{C})$ is E-satisfiable.

Proof. For the soundness direction suppose that $\mathsf{FD}_d(\mathcal{C})$ is E-satisfiable. We have to show that $\mathcal{C}$ has a finite model with d domain elements. The proof is easiest done by exploiting the soundness and completeness results of E-Hyper tableaux, Theorems A.6 and 4.5.

Consider any fair derivation $\mathbf{D}$ of $\mathsf{FD}_d(\mathcal{C})$. Because $\mathsf{FD}_d(\mathcal{C})$ is E-satisfiable, by Theorem A.6 the derivation $\mathbf{D}$ cannot be a refutation. By Theorem 4.5 and the comments following it, $\mathbf{D}$ contains an exhausted branch $\mathbf{B}$. It is clear from inspection of the clause set $\mathsf{FD}_d(\mathcal{C})$ and how the calculus works, that the persistent clauses $\mathbf{B}_\infty$ of $\mathbf{B}$ contain the clauses (1) and (2) from the definition of FD_d above (we make the assumption, though, that the domain elements $1, \ldots, d$ are the smallest terms in the ordering). Further, as per scheme (3), for every n-ary function symbol f in $\mathcal{C}$ and any n integers $i_1, \ldots, i_n$

[19]For this fact to hold, the same argument as in the proof of Lemma A.7 on "decision levels" is implicitly used here.

with $1 \leq i_1, \ldots, i_n \leq d$, there is a k with $1 \leq k \leq d$ such that $\mathbf{B}_\infty$ contains the clause $f(i_1, \ldots, i_n) \simeq k \leftarrow$. These clauses together define a finite interpretation I_d for all function symbols occurring in $\mathcal{C}$ on the domain $\{1, \ldots, d\}$ in the obvious way.

Again with the completeness theorem and the underlying model construction, all ground instances of all clauses (4) in $\mathsf{FD}_d(\mathcal{C})$ are satisfied in the E-Herbrand interpretation induced by $\mathbf{B}_\infty$. In particular, thus, all those ground instances that can be obtained by instantiating with domain elements $1, \ldots, d$ only are satisfied by that interpretation. Because $1, \ldots, d$ are exactly the domain elements of I_d it is not difficult to see that I_d is a (non-Herbrand) E-model of $\mathcal{C}$.

For the completeness direction suppose there is a finite model I_d of $\mathcal{C}$ with d domain elements. Let $1, \ldots, d$ be these elements. It is clear from inspection, that I_d satisfies the clauses (1), (2) and (3) from the definition of FD_d. Because I_d satisfies $\mathcal{C}$, I_d trivially also satisfies the clauses (4). Together, thus I_d satisfies $\mathsf{FD}_d(\mathcal{C})$. In other words, $\mathsf{FD}_d(\mathcal{C})$ is E-satisfiable, which was to be shown. $\square$