UPMAIL Technical Report No. 90
December 7, 1994

Equality Elimination for Semantic Tableaux

Anatoli Degtyarev[*] [†]
Andrei Voronkov[‡]

Computing Science Department
Uppsala University
Box 311, S-751 05 Uppsala,
Sweden

email {anatoli,voronkov}@csd.uu.se

[*]Supported by a grant from the Swedish Institute
[†]On leave from Kiev University
[‡]Supported by a TFR grant

Abstract

We introduce the *equality elimination method* for semantic tableaux with equality. The method combines top-down tableau rules with bottom-up equation solving. The equality elimination method enjoys three features no other tableau method with equality has. First, the method is local, i.e. one can close tableau branches one by one, instead of closing the whole tableau as proposed in other methods. Second, we use ordinary unification instead of rigid or mixed E-unification. Third, we can use powerful strategies, for example ordering and subsumption, to decrease the search space. Bottom-up equation-solving with ordering strategies is made by using *basic superposition.*

Section 1

Semantics tableaux and equality: back to Kanger?

When we began to study literature on semantic tableaux with equality, we found two tendencies in this area. The first tendency is to search for a proof by methods bringing us back to classical papers in proof theory, mainly to [Pra 60] and [Kan 63] and also to results on normalization of proofs in sequential calculi [Lif 68, Ore 69]. Since many notions have been (for good or for bad) reinvented in papers on tableau methods, we would like to recall fundamental ideas of earlier papers.

Prawitz and Kanger proposed the method of applications of proof rules ($\forall \rightarrow$) and ($\rightarrow \exists$) corresponding to γ-rules of the tableau method [Fit 90][1]. The idea was to introduce a new kind of variables (called "dummies" in both [Pra 60, Kan 63]), and to delay instantiation of these variables until necessary information for it is obtained. Comparing this approach to an earlier work of Beth [Bet 58, Bet 59], Prawitz [Pra 60] noted: "the solution proposed here is quite different but well-suited for mechanical use". In Russia this method was independently proposed by N. Shanin[2] and has been characterized as the "metavariable method"[3]. Information for instantiation is provided to constructing an uninstantiated proof, and checking from time to time whether one can find values for dummies which make it a valid proof. In [Kan 63] this check is reduced to verifying that the top sequents are "directly demonstrable", i.e. can be obtained from axioms by application of equality rules. The possibility of applying all equality rules before all other rules has also been demonstrated in [Lif 68, Ore 69].

Dummies or metavariables have later been renamed "free variables" in [Fit 90]. Free variables are substituted for existentially quantified variables in the goal, when γ-rules are applied. Directly demonstrable proofs become closed tableaux, the procedure for a "direct demonstration" is a demonstration for rigid E-unifiability in the set of "unification problems for potentially closing atoms" [Bec 94a].

The second tendency in tableaux-based theorem proving with equality is that of "globality". The main idea of using metavariables — to provide a smallest instantiation of metavariables — comes out through the notion of a most general unifier. But another advantage of post-Kanger procedures is that the idea of metavariables is combined with the principle of *locality*. Among these procedures are resolution, hyperresolution, the inverse method and their numerous modifications [DV 86b, DV 86a, DV 87]. The tableau method trying to achieve goal-orientation sacrifices the

[1]There are differences in *representation* of inferences between [Kan 63] and [Bec 94a] — Kanger uses sequent proofs. Sequents in [Kan 63] correspond to branches in tableaux [Fit 90].

[2]G. Mints, private communications.

[3]The metavariable method is discussed in detail in [MM 83].

principle of locality. A substitution in tableau methods is applied to the whole tableau (see e.g. the "MGU replacement rule" from [Fit 90]), which corresponds to a set of sequences, while in the inverse method [Vor 92] a substitution is applied to a sequent. One can argue pro and contra the principle of locality, but let us note that *all problems considered hard in modern theorem proving are solved by local methods.* The same phenomenon appears in more applied areas of computer science, for example deductive databases [BMSU 86, Vie 89], parsing [TR 94], production systems [MAT 94], constraint logic programming. In all these areas local methods based on bottom-up algorithms[4] show superior performance over top-down methods like tableau. Methods like Early deduction [Ear 70] in parsing or SLD-AL resolution [Vie 89] and magic sets [BMSU 86] use some information from a top-down search but the main information processing is carried out in the bottom-up (not goal-oriented, but local) way.

The denial of the locality principle comes to an extreme in tableaux with equality based on the notion of (rigid, mixed) E-unifiers. If in ordinary tableaux we can at least search for a closing substitution on one branch of a tableau, then in [BH 92, Bec 94a] it is proposed *to close all branches of a tableau simultaneously.* Now search for a substitution should be made by the inspection of the whole tableau. Although in [Bec 94b] it is stated that "methods have to be developed for composing a substitution that closes all branches of a tableau simultaneously from a great number of substitutions closing single branches", it is not clear that such *incremental* methods exist at all[5]. Even if one proves that there are complete incremental methods based on rigid E-unification, it still creates a great deal of non-locality: all equations on a branch participate in solving a rigid E-unification problem that is NP-complete (and even undecidable in the case of mixed E-unification), different branches correspond to different sets of equations and different ways of closing a branch correspond to different E-unification problems. Thus, even a *single unification step* may require solution of an NP-complete or even undecidable problem, substitutions provided by such a unification step are global for the whole table. Moreover, the tableau is expanding and changing during proof-search. This all suggests that *the problem of coupling tableaux and equality should look for a very different methodology.* Another suggestion to use a variant of the paramodulation rule proposed in [Fit 90] is much more friendly to the locality principle: at least a single inference step involves only a single equation and a term (but also creates a large non-determinism due to the omnivorous nature of paramodulation — there may be many paramodulation steps creating no future solutions).

In this paper we propose a method of handling equality in tableau methods which tries to overcome the above mentioned problems. Our method is based on building-in in a tableaux prover a bottom-up equation solver using basic superposition. Solutions to equations are generated by this equation solver and used to close branches of a tableau. Thus, the method combines top-down (non-local) tableaux proof search with bottom-up (local) equation solving. Only completely solved equations are used in the tableau part of the proof, thus reducing non-determinism created by applications of paramodulation in [Fit 90]. The equation solution is even more restricted by two factors: the use of orderings provided by superposition and the basic restriction of superposition. It means that we only have *a fixed finite number of unification problems* and *the same set of equalities used for solution of these unification problems.* This solves the problem of the complete sets-based approach: equation solution never tries to solve problems arising dynamically and the

[4]By "bottom-up" we here mean procedures trying to construct a proof by the search from initial clauses, or axioms to the goal.

[5]For example, a simultaneous E-unifier for the set of E-unification problems $\{\langle\{a=b\},x,y\rangle,\langle\emptyset,f(x,y),f(a,b)\rangle\}$ is $[x/a,y/b]$. If we try to solve this problems incrementally, then a complete set of unifiers of the first problem is $[x/y]$ which makes the second problem unsolvable. In [DV 94] we proved that in a similar situation with SLDE$^+$-resolution [GR 86], the incremental equation solution based on complete sets of E-unifiers is incomplete.

equation solution never uses different sets of equations. And finally, the use of bottom-up equation solving gives rise to many optimizations which are generally adopted in the automated deduction community.

Section 2

Preliminaries

We present here a brief overview of notions and preliminary definitions necessary for understanding the paper. We assume basic knowledge of substitutions and unification.

Let Σ be a signature, and X a set of variables. $T(\Sigma, X)$ will denote the set of all terms in the signature Σ with variables from X. The set of all *ground* terms in the signature Σ will be denoted by $T(\Sigma)$.

A *literal* is either an atomic formula or a negation of an atomic formula. We shall always write $s \neq t$ for the literal $\neg(s = t)$.

A *clause* is a finite set of literals, denoted $L_1, \ldots, L_n$. If L is a literal, C a clause, L, C will denote the clause $\{L\} \bigcup C$. An *equation* is an expression $s = t$, where s, t are terms in $T(\Sigma, X)$. By a *ground expression* (i.e. term, equation, clause etc.) we mean an expression containing no variables. We write $A[s]$ to indicate that an expression A contains s as a subexpression and *ambiguously* denote by $A[t]$ the result of replacing particular occurrences of s in A by t. By $A\sigma$ we denote the result of applying the substitution σ to A.

We shall sometimes denote tuples of equations $s_1 = t_1, \ldots, s_n = t_n$ by $\bar{s} = \bar{t}$. The overbar notation $\bar{t}$ can also be used to denote sequences of terms. Substitutions θ with the domain $x_1, \ldots, x_n$ will be denoted by $[x_1/x_1\theta, \ldots, x_n/x_n\theta]$.

Let $\succ$ be a partial ordering on $T(\Sigma, X)$. It is called a *reduction ordering* iff

1. $\succ$ is well-founded;

2. if $s \succ t$ then $u[s\sigma] \succ u[t\sigma]$, for all terms s, t, u and substitutions σ.

We shall always assume that reduction orderings are total on the set $T(\Sigma)$.

Following [BGLS 92] we distinguish terms occurring in the original formula from terms introduced by substitution by using *closures,* i.e. pairs $C \cdot \sigma$, where C is a clause, σ a substitution. The application of a substitution ρ to such a closure is defined by $(C \cdot \sigma)\rho = C \cdot \sigma\rho$. The clause C will often be identified with the closure $C \cdot \varepsilon$, where ε is the empty substitution. The set $Var(C \cdot \sigma)$ is defined as the set of all variables of $C\sigma$. Two closures $C_1 \cdot \sigma_1$ and $C_2 \cdot \sigma_2$ have *disjoint variables* iff $Var(C_1 \bigcup C_1\sigma_1) \bigcap Var(C_2 \bigcup C_2\sigma_2) = \emptyset$. An *instance of the closure* $C \cdot \rho$ is any closure $C \cdot \rho\sigma$. The instance is ground iff $Var(C\rho\sigma) = \emptyset$. Two closures are *variants* iff both are instances of each other.

When we use notation $\models$ or $\vdash$, they state for truth and provability *in classical first order logic.*

A *tableau* is an expression of the form

$C_1 \mid \ldots \mid C_n$

where C_i are clauses and $n >= 0$. The tableau with $n = 0$ is called *the empty tableau.* Tableaux are considered commutative, so that there is no difference between the above tableau and

$$C_n \mid \ldots \mid C_1$$

Traditional tree-like representation of tableaux can be transformed to our representation in the following way. For each branch B_i of the tree-like tableau, take the set of all formulas $\varphi_1, \ldots, \varphi_n$ on this branch and replace these formulas on their *names* (literals, defined below), then we obtain the clause C_i.

A formula is in *skolemized negation normal form* iff it is constructed from literals using connectives $\wedge, \vee$ and the quantifier $\exists$. There is a structure-preserving translation of formulas without equivalences into formulas in skolemized negation normal with the same number of occurrences of atoms.

2.1 Subset unification

Our algorithm uses *subset unification.* We shall formally define it and prove some properties. A variant of subset unification has been used in [MRL 91] as θ-subsumption. A substitution σ is called a *subset unifier* of clauses D_1 and D_2 iff $D_1\sigma \subseteq D_2\sigma$. It is a *most general subset unifier* of D_1 and D_2 iff there is no other subset unifier σ' of D_1 and D_2 such that $\sigma' \leq \sigma$ and not $\sigma \leq \sigma'$.

The following example shows that there may be several most general subset unifiers:

Example 2.1 *Consider clauses $P(x)$ and $P(a), P(b)$. Then there are two most general subset unifiers $[x/a]$ and $[x/b]$.*

A longer clause may have a subset unifier with a shorter clause:

Example 2.2 *Consider clauses $P(x), P(y)$ and $P(a)$. They have a subset unifier $[x/a, y/a]$*

Lemma 2.1 *Let the substitution σ be a (most general) subset unifier of $B_1, \ldots, B_l$ and $A_1, \ldots, A_k$. Then there is a function $f : \{B_1, \ldots, B_l\} \to \{A_1, \ldots, A_k\}$ such that σ is a (most general) unifier of tuples $\langle B_1, \ldots, B_l\rangle$ and $\langle f(B_1), \ldots, f(B_l)\rangle$.*

Proof. We have $B_1\sigma, \ldots, B_l\sigma \subseteq A_1\sigma, \ldots, A_k\sigma$. Hence, there is a function f such that $B_i\sigma = D_{f(i)}\sigma$. Thus, σ is a unifier of tuples $\langle B_1, \ldots, B_l\rangle$ and $\langle f(B_1), \ldots, f(B_l)\rangle$.

It is easy to see that every such unifier of these tuples is a subset unifier of $B_1, \ldots, B_l$ and $A_1, \ldots, A_k$. Thus, σ must be a most general unifier of the tuples. $\square$

Let us note that not every most general unifier of the tuples is a most general subset unifier. Consider the following example:

Example 2.3 *Assume that the clause $P(x, x)$ is subset-unified with the clause $P(y, y), P(y, a)$. There are two possible choices of the function f defined by $f(1) = 1$ and $f(1) = 2$. In the first case a most general tuple unifier (of $P(x, x)$ and $P(y, y)$) is $[x/y]$, in the second case (of $P(x, x)$ and $P(y, a)$) it is $[x/a, y/a]$. Only the first of these tuple unifiers is a most general subset unifier.*

Lemma 2.2 *There is only a finite number (up to renaming variables) of most general subset unifiers of S_1 and S_2.*

```
subset_unify( [], _).
subset_unify( [A|As], Bs) :-
     subset_unify_one( A, Bs),
     subset_unify( As, Bs).

subset_unify_one( A, [B|_]) :-
     unify( A, B),
subset_unify_one( A, [_|Bs]) :-
     subset_unify_one( A, Bs).
```

Here clauses are represented via lists. It is also assumed that `unify` unifies two terms.

Figure 2.1: A subset unification algorithm.

Proof. Immediate from Lemma 2.1, since there is only a finite number of functions f between two finite sets, and for every choice of f there is at most one most general unifier (up to renaming of variables). $\square$

As Example 2.3 shows, the search for all *most general* subset unifiers can be very difficult, since it requires comparison of unifiers obtained for different choices of the function f. In practice, it is easier to search for a *complete set* U of subset unifiers, i.e. such a set that for every subset unifier σ, there is $\sigma' \in U$ such that $\sigma' \leq \sigma$. A subset unification algorithm on Figure 2.1 expressed in Prolog-like notation computes a complete set of subset unifiers:

This unification algorithm is exponential, which cannot be improved in general:

Example 2.4 *Consider subset unification of the clauses* $A(x_1), \ldots, A(x_n)$ *with* $A(y_1), \ldots, A(y_m)$*. Then there are* m^n *most general unifiers, each one corresponding to a function from* $\{A(x_1), \ldots, A(x_m)\}$ *to* $\{A(y_1), \ldots, A(y_n)\}$

Let us note that the problem of *subset-unifiability*, i.e. that of existence of a subset unifier is NP-complete:

Theorem 2.1 *The subset-unifiability problem is NP-complete.*

Proof.

1. By using the subset unification algorithm from Figure 2.1 as a non-deterministic algorithm and by using any polynomial `unify` algorithm, we obtain that the problem is in NP.

2. The multi-literal subsumption problem is polynomially reduced to the subset-unifiability problem as follows: for clauses D_1, D_2 we have $D_1 \leq D_2$ iff D_1 is subset unifiable with a clause obtained from D_2 by a substitution $[x_1/c_1, \ldots, x_n/c_n]$ of pairwise different constants $c_1, \ldots, c_n$ for all variables $x_1, \ldots, x_n$ of D_2. The subsumption relation $\leq$ is known to be NP-complete [GJ 79].

$\square$

Section 3

The equality elimination method

For the rest of this section we assume that γ is a closed formula in skolemized negation normal form to be proved (the "goal"). We assume that all different occurrences of quantifiers in γ bind different variables. For example, γ cannot have the form $\exists x A \vee \exists x B$. All formulas in this section are assumed to be subformulas of γ.

We shall identify subformulas and superformulas with their *occurrences* in γ. For example, in the formula γ of the form

$$A \wedge (\mathbf{A} \vee \mathbf{B})$$

the boldfaced $\mathbf{A}$ is considered as a subformula of $(\mathbf{A} \vee \mathbf{B})$, but the ordinary A is not a subformula of $(\mathbf{A} \vee \mathbf{B})$. A subformula φ of γ is called *conjunctive* iff it is an occurrence in a subformula $\varphi \wedge \psi$ or in $\psi \wedge \varphi$. A *conjunctive superformula* of φ is a superformula ψ of φ that is conjunctive. The *least conjunctive superformula* of φ is the conjunctive superformula ψ of φ such that any other conjunctive superformula of φ is a superformula of ψ. Let us note that conjunctive superformulas do not necessarily exist. For example, γ has no conjunctive superformula. Any formula having a conjunctive superformula has the unique least conjunctive superformula.

We can enumerate all conjunctive subformulas, for example in the order of their occurrences in γ. Thus we can unambiguously use "the nth conjunctive (sub)formula".

Let $A_1, A_2, \ldots$ be a set of predicate symbols not occurring in γ. We say that $A_n(x_1, \ldots, x_m)$ is *the γ-name for a subformula φ* iff there is the subformula ψ such that

1. ψ is the n-th conjunctive subformula of γ.

2. ψ is the least conjunctive superformula of φ.

3. $x_1, \ldots, x_m$ are all free variables of ψ in the order of their occurrences in ψ.

γ-names not always exist. We shall use *the set of γ-names* of a subformula. The set of γ-names is either $\emptyset$ or a singleton $\{A_i(x_1, \ldots, x_k)\}$. Thus, we can consider sets of γ-names as clauses. Also note that different formulas may have the same γ-names.

The following example illustrates least conjunctive superformulas and γ-names:

Example 3.1 *Let γ be the formula*

$$(\exists x (A(x) \wedge (B(x) \vee \exists y C(x, y))) \wedge \exists z D(z)) \vee E$$

All subformulas of γ and their γ-names are shown in the following table:

Subformula	*least conjunctive superformula*	*name*
$(\exists x(A(x) \wedge (B(x) \vee \exists y C(x,y))) \wedge \exists z D(z)) \vee E$		
$(\exists x(A(x) \wedge (B(x) \vee \exists y C(x,y))) \wedge \exists z D(z))$		
$\exists x(A(x) \wedge (B(x) \vee \exists y C(x,y))$	$\exists x(A(x) \wedge (B(x) \vee \exists y C(x,y))$	A_1
$A(x) \wedge (B(x) \vee \exists y C(x,y)$	$\exists x(A(x) \wedge (B(x) \vee \exists y C(x,y))$	A_1
$A(x)$	$A(x)$	$A_2(x)$
$B(x) \vee \exists y C(x,y)$	$B(x) \vee \exists y C(x,y)$	$A_3(x)$
$B(x)$	$B(x) \vee \exists y C(x,y)$	$A_3(x)$
$\exists y C(x,y)$	$B(x) \vee \exists y C(x,y)$	$A_3(x)$
$C(x,y)$	$B(x) \vee \exists y C(x,y)$	$A_3(x)$
$\exists z D(z)$	$\exists z D(z)$	A_4
$D(z)$	$\exists z D(z)$	A_4
E		

Lemma 3.1 *Let φ be a subformula of γ. Then*

1. *If ψ is the least conjunctive superformula of φ then $\models \forall(\varphi \supset \psi)$.*
2. *If there is no conjunctive superformula of φ then $\models \forall(\varphi \supset \gamma)$.*

This lemma partially explains the need for introducing least conjunctive superformulas. There are deterministic chains of inferences in sequent systems, where formulas with $\vee$ or $\exists$ occur. For example,

$$\frac{\dfrac{\Gamma, \varphi}{\Gamma, \varphi \vee \psi}}{\Gamma, \exists x(\varphi \vee \psi)}$$

(Since we are dealing with formulas in negation normal form, we represent sequents as multisets of formulas, similar to [Sch 60].) By restricting ourselves to conjunctive superformulas only, we eliminate these deterministic chains, making them in one step.

Now we can formulate the tableau deductive system we are going to use. The proof-search consists of two parts: the equality solution part and the tableau part. In the first part, we try to solve equalities in the formula, generating a set of clauses not containing equality. A clause not containing equality will be called a *solution clause.* This set of solution clauses is obtained by basic superposition from *initial closures* defined below. In the second part, the usual tableau proof search is performed. The solution clauses generated in the first part are used to close branches in the tableau proof tree. One solution clause can in general close many branches.

Initial closures are generated according to one of the three rules:

1. Whenever a literal $s \neq t$ occurs in γ and C is the set of γ-names for this occurrence of $s \neq t$, the closure $s = t, C \cdot \varepsilon$ is an initial closure.
2. Whenever $s = t$ occurs in γ and C is the set of γ-names for this occurrence of $s = t$, the closure $s \neq t, C \cdot \varepsilon$ is an initial closure.
3. Let literals $P(s_1, \ldots, s_n)$ and $\neg P(t_1, \ldots, t_n)$ occur in γ and C_1, C_2 are their sets of γ-names. Let the substitution σ rename variables such that variables of $C_1\sigma$ and C_2 are disjoint. Then the closure $s_1\sigma \neq t_1, \ldots, s_n\sigma \neq t_n, C_1\sigma, C_2 \cdot \varepsilon$ is an initial closure.

Branch expansion rules are formed as follows. Assume that there is a subformula $\varphi \wedge \psi$. Let N_1, N_2 and N be the sets of names of φ, ψ and $\varphi \wedge \psi$. Then

$$
\begin{tikzcd}
 & N \arrow[dl, dash] \arrow[dr, dash] & \\
N_1 & & N_2
\end{tikzcd}
$$

is a branch expansion rule for γ.

Intuitively, branch expansion rules encode the conjunction rule of sequent calculi

$$\frac{\Gamma, \varphi \qquad \Gamma, \psi}{\Gamma, \varphi \wedge \psi}$$

but turned upside down for consistency with the tableau notation.

The equality elimination method consists of five inference rules. The *basic superposition* rules are defined as in [BGLS 92]. We always assume that premises of rules have disjoint variables. The first three rules derive clauses. The last two rules derive tableaux. The *initial tableau* is the tableau consisting of the empty clause. The proof always starts from the initial tableau. The goal is to derive the empty tableau.

Basic right superposition

$$\frac{(C, s = t) \cdot \sigma_1 \qquad (u[s'] = v, D) \cdot \sigma_2}{(u[t] = v, C, D) \cdot \sigma_1 \sigma_2 \rho}$$

where

1. ρ is a most general unifier of $s\sigma_1$ and $s'\sigma_2$.
2. $t\sigma_1\rho \not\succeq s\sigma_1\rho$;
3. $v\sigma_2\rho \not\succeq u[s']\sigma_2\rho$;
4. s' is not a variable.

(We also remind that variables of the premise must be disjoint.)

Basic left superposition

$$\frac{(C, s = t) \cdot \sigma_1 \qquad (u[s'] \neq v, D) \cdot \sigma_2}{(u[t] \neq v, C, D) \cdot \sigma_1 \sigma_2 \rho}$$

with the same conditions as for basic right superposition. We shall use one more restriction on the basic left superposition, when there are several inequalities in the first clause: we only allow for superposition in the leftmost inequality.[1]

[1] According to our definitions, a clause is a set of literals, so the leftmost literal may sound as nonsense, but this restriction can easily be formalized using a variant of *locking* [Boy 71].

Equality solution

$$\frac{(C, s \neq t) \cdot \sigma}{C \cdot \sigma\rho}$$

where ρ is a most general unifier of $s\sigma, t\sigma$.

Branch expansion

1. If there is a branch expansion rule

   $$\begin{array}{ccc} & A(x_1, \ldots, x_n) & \\ \swarrow & & \searrow \\ A_1 & & A_2 \end{array}$$

   then from the tableau

   $$A(t_1, \ldots, t_n), C_1 \mid C_2 \mid \ldots \mid C_m$$

   one can derive the tableau

   $$A_1\sigma, A(t_1, \ldots, t_n), C_1 \mid A_2\sigma, A(t_1, \ldots, t_n), C_1 \mid C_2 \mid \ldots \mid C_m$$

   where σ is the substitution $[x_1/t_1, \ldots, x_n/t_n]$.

2. If there is a branch expansion rule

   $$\begin{array}{ccc} & & \\ \swarrow & & \searrow \\ A_1 & & A_2 \end{array}$$

   then from the tableau

   $$C_1 \mid C_2 \mid \ldots \mid C_m$$

   then one can derive the tableau

   $$A_1, C_1 \mid A_2, C_1 \mid C_2 \mid \ldots \mid C_m$$

Branch closure Let C be a solution clause and σ be a most general subset unifier of C and a clause C_1. Then from the tableau

$$C_1 \mid C_2 \mid \ldots \mid C_m$$

one can derive the tableau[2]

$$C_2\sigma \mid \ldots \mid C_m\sigma$$

We shall call it a *branch closure against the clause C*. Note that only solution clauses can be used for closing branches. There are no rules in our system that can add an equality literal in a tableau. Thus, a clause containing equality cannot be made a subset of a tableau clause.

[2] As can be seen from this rule, we partially sacrifice locality when closing tableau branches — a tribute to the tableau method. The equality solution part of our method is fully local.

The following theorems are true about the method:

Theorem 3.1 (Soundness) *If there is a derivation of the empty tableau from the initial tableau then γ is provable.*

Theorem 3.2 (Completeness) *If γ is provable then there is a derivation of the empty tableau from the initial tableau.*

Proofs of these two theorems are give in Appendix A.
Consider several examples. The following example is taken from [BH 92].

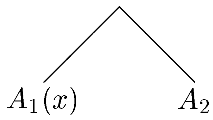
Example 3.2 *The formula to be proved is*

$$\exists x(g(f(x)) \neq c \vee (f(x) \neq g(x) \wedge (B \vee \neg B))) \vee g(g(a)) = g(f(b))$$

There are two conjunctive subformulas with the following names:

$$\begin{aligned} A_1(x) &\equiv f(x) \neq g(x) \\ A_2 &\equiv B \vee \neg B \end{aligned}$$

The corresponding branch expansion rule is



The initial closures are

1. $g(f(x)) = c \cdot \varepsilon$
2. $f(x) = g(x), A_1(x) \cdot \varepsilon$
3. $g(f(b)) \neq g(g(a)) \cdot \varepsilon$
4. $A_2 \cdot \varepsilon$

the first three are generated by equalities in the goal, the last one by the pair $B, \neg B$.
As the order $\succ$ we consider the recursive path ordering with $f > g > c > a > b$. Basic superposition and equality solution generate the following clauses:

5. $g(g(a)) \neq c \cdot \varepsilon$ [left superposition from 1,3]
6. $g(g(x)) = c, A_1(y) \cdot [x/y]$ [right superposition from 2,1]
7. $A_1(y) \cdot [y/a]$ [left superposition + equality solution from 6,5]

The tableau part of the proof is very easy:

8. $A_1(x) \mid A_2$ [branch expansion]
9. A_2 [branch closure from 7,8]
10. $\square$ [branch closure from 4,9]

Another example is from [Bec 94a]:

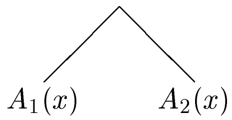
Example 3.3 *The formula to be proved is*

$$\exists x(g(x) \neq f(x) \wedge x = a) \vee \exists x(g(f(x)) \neq x) \vee b \neq c \vee \neg P(g(g(a)), b) \vee P(a, c)$$

There are two conjunctive subformulas with the following names:

$$\begin{array}{lcl} A_1(x) & \equiv & g(x) \neq f(x) \\ A_2(x) & \equiv & x = a \end{array}$$

The corresponding branch expansion rule is



The initial closures are

1. $f(x) = g(x), A_1(x) \cdot \varepsilon$
2. $A_2(x) \vee x \neq a \cdot \varepsilon$
3. $g(f(x)) = x \cdot \varepsilon$
4. $b = c \cdot \varepsilon$
5. $g(g(a)) \neq a, b \neq c \cdot \varepsilon$

the first four are generated by equalities in the goal, the last one by the pair $\neg P(g(g(a)), b), P(a, c)$.

As the order $\succ$ *we consider the recursive path ordering with* $f > g > b > c > a$. *Basic superposition and equality solution generate the following clauses:*

6. $g(g(x)) = x, A_1(y) \cdot [x/y]$ [right superposition from 1,3]
7. $A_2(x) \cdot [x/a]$ [equality solution from 2]
8. $A_1(y), b \neq c \cdot [y/a]$ [left superposition + equality solution from 6,5]
9. $A_1(y) \cdot [y/a]$ [left superposition + equality solution from 8,4]

The tableau part of the proof is also very easy:

10. $A_1(x) \mid A_2(x)$ [branch expansion]
11. $A_2(a)$ [branch closure from 7,10]
12. $\Box$ [branch closure from 9,11]

There are many optimizations not considered in this paper. The most powerful one is *subsumption* that we can apply in the equality solution part in order to prevent generation of unnecessary clauses. Subsumption for the basic strategy has been explained in [Deg 82, BGLS 92].

Appendix A

Proof of main theorems

In order to prove Theorems 3.1 and 3.2 we define several notions and state and prove several lemmas. They will be illustrated on Example 3.3, i.e. when γ is

$$\exists x(g(x) \neq f(x) \wedge x = a) \vee \exists x(g(f(x)) \neq x) \vee b \neq c \vee \neg P(g(g(a)), b) \vee P(a, c)$$

Extend the naming relation to *all* positive subformulas in the following way. Let N be any one-to-one function from the set of all (occurrences) of positive subformulas of γ to $\{A_1, A_2, \ldots\}$ such that for the nth conjunctive subformula φ_n of γ we have $N(\varphi_n) = A_n$. Then γ-names for an arbitrary positive subformula φ of γ are defined as $A_k(x_1, \ldots, x_m)$, where $N(\varphi) = A_k$ and $x_1, \ldots, x_m$ are all variables of φ. For example, possible names of all positive subformulas of γ from our example are shown in the following table:

name	formula
A_3	$\exists x(g(x) \neq f(x) \wedge x = a) \vee \exists x(g(f(x)) \neq x) \vee b \neq c \vee \neg P(g(g(a)), b) \vee P(a, c)$
A_4	$\exists x(g(x) \neq f(x) \wedge x = a) \vee \exists x(g(f(x)) \neq x) \vee b \neq c \vee \neg P(g(g(a)), b)$
A_5	$P(a, c)$
A_6	$\exists x(g(x) \neq f(x) \wedge x = a) \vee \exists x(g(f(x)) \neq x) \vee b \neq c$
A_7	$\neg P(g(g(a)), b)$
A_8	$\exists x(g(x) \neq f(x) \wedge x = a) \vee \exists x(g(f(x)) \neq x)$
A_9	$b \neq c$
A_{10}	$\exists x(g(x) \neq f(x) \wedge x = a)$
A_{11}	$\exists x(g(f(x)) \neq x)$
$A_{12}(x)$	$g(x) \neq f(x) \wedge x = a$
$A_1(x)$	$g(x) \neq f(x)$
$A_2(x)$	$x = a$
$A_{13}(x)$	$g(f(x)) \neq x$

Denote the name of a formula φ by $\overline{\varphi}$.
Consider now the set Φ_1 of formulas defined as

$$\Phi_1 = \{\forall(\varphi \equiv \overline{\varphi}) \mid \varphi \text{ is a positive subformula of } \gamma\} \bigcup \{\neg\overline{\gamma}\}$$

Lemma A.1 *The formula γ is provable iff Φ_1 is inconsistent.*

Proof. First note that γ is provable iff $\neg\gamma$ is inconsistent. Thus, the statement can be reformulated as: $\neg\gamma$ is consistent iff Φ_1 is consistent.

1. Let M be a model for Φ_1. Then $M \models \neg\overline{\gamma}$ and $M \models \gamma \equiv \overline{\gamma}$. Thus, $M \models \neg\gamma$.

2. Let $M \models \neg\gamma$. Consider the model M' defined as follows: predicates from the language of γ are interpreted as in M, and the interpretation of new predicates is given by $\varphi \equiv \overline{\varphi}$. Obviously, $M' \models \Phi_1$.

$\square$

Let Φ_2 be the following set of formulas:

$$
\begin{aligned}
\Phi_2 = \quad & \{\forall(\overline{\varphi \wedge \psi} \equiv \overline{\varphi} \wedge \overline{\psi}) \mid \varphi \wedge \psi \text{ is a subformula of } \gamma\} \bigcup \\
& \{\forall(\overline{\varphi \vee \psi} \equiv \overline{\varphi} \vee \overline{\psi}) \mid \varphi \vee \psi \text{ is a subformula of } \gamma\} \bigcup \\
& \{\forall(\overline{\exists x \varphi} \equiv \exists x \overline{\varphi}) \mid \exists x \varphi \text{ is a subformula of } \gamma\} \bigcup \\
& \{\forall(\overline{L} \equiv L) \mid L \text{ is a literal in } \gamma\} \bigcup \\
& \{\neg\overline{\gamma}\}
\end{aligned}
$$

In our example Φ_2 is the set of the following formulas

$$
\begin{aligned}
& A_3 \equiv A_4 \vee A_5 \\
& A_4 \equiv A_6 \vee A_7 \\
& A_5 \equiv P(a, c) \\
& A_6 \equiv A_8 \vee A_9 \\
& A_7 \equiv \neg P(g(g(a)), b) \\
& A_8 \equiv A_{10} \vee A_{11} \\
& A_9 \equiv b \neq c \\
& A_{10} \equiv \exists x A_{12}(x) \\
& A_{11} \equiv \exists x A_{13}(x) \\
& \forall x(A_{12}(x) \equiv A_1(x) \wedge A_2(x)) \\
& \forall x(A_{13}(x) \equiv g(f(x)) \neq x) \\
& \forall x(A_1(x) \equiv g(x) \neq f(x)) \\
& \forall x(A_2(x) \equiv x = a) \\
& \neg A_3
\end{aligned}
$$

Lemma A.2 *γ is provable iff Φ_2 is inconsistent.*

Proof. It is easy to see that Φ_1 and Φ_2 are equivalent: each of them can be obtained from the other by substitution of equivalent formulas for equivalent. Then apply Lemma A.1. $\square$

Let Φ_3 be the set of formulas defined as

$$\Phi_3 = \{\forall(A \supset \overline{B}) \mid \forall(\overline{B} \equiv A) \in \Phi_2\} \bigcup \{\neg\overline{\gamma}\}$$

Lemma A.3 *γ is provable iff Φ_3 is inconsistent.*

Proof. First prove that Φ_3 is consistent iff Φ_2 is consistent.

1. Obviously $\Phi_2 \models \Phi_3$.

2. Assume that $M \models \Phi_3$. Let $\varphi_1, \ldots, \varphi_n$ be any enumeration of all proper subformulas of γ such that if φ_i is a subformula of φ_j, then $i \leq j$. Let $\psi_i \in \Phi_3$ are formulas having the form $\forall(\delta_i \supset \overline{\varphi_i})$, for some δ_i. Let χ_i be formulas $\forall(\delta_i \equiv \overline{\varphi_i})$. Note that $\Phi_3 = \{\psi_1, \ldots, \psi_n, \neg\overline{\gamma}\}$ and $\Phi_2 = \{\chi_1, \ldots, \chi_n, \neg\overline{\gamma}\}$. Denote by Π_i sets $\{\chi_1, \ldots, \chi_{i-1}, \psi_i, \ldots, \psi_n, \neg\gamma\}$. Then $\Pi_1 = \Phi_3$. We prove, by induction on i, that all Π_i are satisfiable. For $i = 1$ the model $M_1 = M$ satisfies Π_1. Let M_{i-1} be a model for Π_{i-1}. The predicate symbol of $\overline{\varphi_i}$ has only two occurrences in Π_i: one in $\forall(\delta_i \supset \overline{\varphi_i})$ and one negative occurrence in some ψ_j with $i < j$. Consider the model M_i in which the interpretation of the predicate of $\overline{\varphi_i}$ is defined by the formula $\delta_i \equiv \overline{\varphi_i}$ and the interpretation of other predicates is the same as in M_{i-1}. Obviously, $M_i \models \Pi_i$.

Note that $\Pi_{n+1} = \Phi_2$. These prove that Φ_2 is inconsistent iff Φ_3 is inconsistent. Then apply Lemma A.2. □

In our example Φ_3 is the set of the following formulas

$A_4 \vee A_5 \supset A_3$
$A_6 \vee A_7 \supset A_4$
$P(a,c) \supset A_5$
$A_8 \vee A_9 \supset A_6$
$\neg P(g(g(a)),b) \supset A_7$
$A_{10} \vee A_{11} \supset A_8$
$b \neq c \supset A_9$
$\exists x A_{12}(x) \supset A_{10}$
$\exists x A_{13}(x) \supset A_{11}$
$\forall x(A_1(x) \wedge A_2(x) \supset A_{12}(x))$
$\forall x(g(f(x)) \neq x \supset A_{13}(x))$
$\forall x(g(x) \neq f(x) \supset A_1(x))$
$\forall x(x = a \supset A_2(x))$
$\neg A_3$

One can note that this set can be considerably simplified, so that only the names of conjunctive subformulas will matter. The following lemma shows that names of non-conjunctive subformulas can be eliminated:

Lemma A.4 *The following are true:*

1. *Assume that Γ is a set of formulas such that*

$$\Gamma = \Delta \bigcup \{\forall(\overline{\varphi_1} \vee \overline{\varphi_2} \supset \overline{\varphi_3}), \forall(\psi_1 \supset \overline{\varphi_1}), \ldots, \forall(\psi_m \supset \overline{\varphi_1}), \forall(\chi_1 \supset \overline{\varphi_2}) \ldots, \forall(\chi_n \supset \overline{\varphi_2})\}$$

where Δ, ψ_i, χ_j do not have occurrences of the predicate symbols of $\overline{\varphi_1}, \overline{\varphi_2}$. Then Γ is satisfiable iff the set

$$\Gamma = \Delta \bigcup \{\forall(\psi_1 \supset \overline{\varphi_3}), \ldots, \forall(\psi_m \supset \overline{\varphi_3}), \forall(\chi_1 \supset \overline{\varphi_3}), \ldots, \forall(\chi_n \supset \overline{\varphi_3})\}$$

is satisfiable.

2. *Assume that* Γ *is a set of formulas such that*

$$\Gamma = \Delta \bigcup \{\forall(\exists x \overline{\varphi} \supset \overline{\psi}), \forall(\chi_1 \supset \overline{\varphi}) \ldots, \forall(\chi_n \supset \overline{\varphi})\}$$

where $\Delta, \overline{\psi}, \chi_i$ *do not have occurrences of the predicate symbol of* $\overline{\varphi}$. *Then* Γ *is satisfiable iff the set*

$$\Gamma = \Delta \bigcup \{\forall(\chi \supset \overline{\psi})\}$$

is satisfiable.

3. *Assume that* Γ *is a set of formulas such that*

$$\Gamma = \Delta \bigcup \{\forall(\varphi_1 \supset \overline{\gamma}), \ldots, \forall(\varphi_n \supset \overline{\gamma}), \neg\overline{\gamma}\}$$

where Δ, φ_i *do not have occurrences of the predicate symbol of* $\overline{\gamma}$. *Then* Γ *is satisfiable iff the set*

$$\Gamma = \Delta \bigcup \{\forall(\neg\varphi_1), \ldots, \forall(\neg\varphi_n)\}$$

is satisfiable.

Proof. Obvious. □

Consider how we can simplify Φ_3 using repeated applications of this lemma. On the first step we eliminate all disjunctions obtaining

$P(a,c) \supset A_3$
$\neg P(g(g(a)),b) \supset A_3$
$b \neq c \supset A_3$
$\exists x A_{12}(x) \supset A_3$
$\exists x A_{13}(x) \supset A_3$
$\forall x(A_1(x) \wedge A_2(x) \supset A_{12}(x))$
$\forall x(g(f(x)) \neq x \supset A_{13}(x))$
$\forall x(g(x) \neq f(x) \supset A_1(x))$
$\forall x(x = a \supset A_2(x))$
$\neg A_3$

Then we eliminate all occurrences of $\exists$ obtaining

$P(a,c) \supset A_3$
$\neg P(g(g(a)),b) \supset A_3$
$b \neq c \supset A_3$
$\forall x(A_1(x) \wedge A_2(x) \supset A_3)$
$\forall x(g(f(x)) \neq x \supset A_3)$
$\forall x(g(x) \neq f(x) \supset A_1(x))$
$\forall x(x = a \supset A_2(x))$
$\neg A_3$

Finally, we get rid of $\overline{\gamma}$, i.e. A_3:

$$\begin{array}{l}\neg P(a,c)\\ P(g(g(a)),b)\\ b=c\\ \forall x \neg(A_1(x)\wedge A_2(x))\\ \forall x(g(f(x))=x)\\ \forall x(g(x)\neq f(x)\supset A_1(x))\\ \forall x(x=a\supset A_2(x))\end{array}$$

Note that this set can be considered as a set of clauses.

$$\begin{array}{l}\neg P(a,c)\\ P(g(g(a)),b)\\ b=c\\ \neg A_1(x),\neg A_2(x)\\ g(f(x))=x\\ g(x)=f(x),A_1(x))\\ x\neq a,A_2(x)\end{array}$$

From now on we shall work with clauses instead of formulas. Denote the set of clauses obtained after this simplification by Φ_4. Lemma A.4 immediately gives

Lemma A.5 γ *is provable iff* Φ_4 *is inconsistent.*

The following trick allows one to get rid of the predicate symbols occurring in γ, thus only leaving new predicate symbols and equality:

Lemma A.6 *Let* P *be a predicate symbol different from equality,* S *a set of clauses such that*

$$S=S'\bigcup\{P(\bar{t_i}),C_i \mid i\in\{1,\dots,n\}\}\bigcup\{\neg P(\bar{s_j}),D_j \mid j\in\{1,\dots,m\}\}$$

where C_i,D_j *are clauses and* P *does not occur in* S' *or any of* C_i,D_j*. Then* S *is consistent iff the set*

$$R=S'\bigcup\{\bar{t_i}\neq\bar{s_j},C_i,D_j \mid i\in\{1,\dots,n\},j\in\{1,\dots,m\}\}$$

is consistent.

Proof.

$\Rightarrow$ is obvious, since $S\models R$.

$\Leftarrow$ Let R be consistent. Then there is a term model $M_R\models R$. Let $\approx$ be the congruence on $T(\Sigma)$ defining the carrier set of M_R. For a term t, denote by $[t]_\approx$ the equivalence class of t under $\approx$. Enrich M_R by the new predicate P obtaining the model M_S such that $M_S\models S$ as follows. For simplicity, but without loss of generality, we assume that P is unary. Obviously, it is enough to define P only on elements of the form $[t_i\sigma]_\approx$ and $[s_j\sigma]_\approx$. Let $[t_i\sigma]_\approx$ is such that $[s_j\theta]_\approx\neq[t_i\sigma]_\approx$, for all j and θ. Then we let $M_S\models P(t_i\sigma)$. Let $[s_j\theta]_\approx$ is such that $[t_i\theta]_\approx\neq[s_j\theta]_\approx$, for all i and σ. Then we let $M_S\models\neg P(s_j\theta)$.

Let an equivalence class w.r.t. $\approx$ in M_R contain terms $t_{i_1}\sigma_{i_1}, t_{i_2}\sigma_{i_2}, \dots$ and terms $s_{j_1}\sigma_{j_1}, s_{j_2}\sigma_{j_2}, \dots$. We prove that at least one of the truth values for $P([t_{i_1}\sigma_{i_1}])$ will not make S false in M_S. Assume the inverse. Let $M_S \not\models P([t_{i_1}\sigma_{i_1}])$ makes S false. Then there is p such that $M_S \not\models P(t_{i_p}\sigma_{i_p}) \vee C_{i_p}\sigma_{i_p}$.

Assume also that $M_S \models P([s_{j_1}\theta_{j_1}])$ makes S false. Then there is q such that $M_S \not\models \neg P(s_{j_q}\theta_{j_q}) \vee D_{j_q}\theta_{j_q}$. Then we obtain $M_R \not\models (t_{i_p} \neq s_{j_q} \vee C_{i_p} \vee D_{j_q})\delta$, where $\delta = \sigma_{i_p} \bigcup \theta_{j_q}$ (in the union operation $\bigcup$ we use that domains of σ_{i_p} and θ_{j_q} are disjoint). Contradiction.

Arbitrarily define P on all other equivalence classes. It follows that $M_S \models S$.

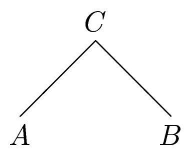
□

According to this lemma, we can replace all clauses containing predicate symbols from the original signature as stated in the lemma. Denote thus obtained set of clauses by S_γ. In our example the set S_γ looks as follows:

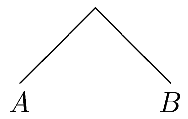
$a \neq g(g(a)), c \neq b$
$b = c$
$\neg A_1(x), \neg A_2(x)$
$g(f(x)) = x$
$g(x) = f(x), A_1(x))$
$x \neq a, A_2(x)$

Lemma A.7 *The set of clauses S_γ consists of*

1. *All clauses C, where $C \cdot \varepsilon$ is an initial closure.*
2. *All clauses $\neg A, \neg B, C$, where*

   is a branch expansion rule.
3. *All clauses $\neg A, \neg B$, where*

   is a branch expansion rule.

Proof. By routine inspection of the construction of S_γ. □

Lemmas A.5 and A.6 prove

Lemma A.8 *The formula γ is provable iff the set S_γ is inconsistent.*

The following lemma shows that all basic superpositions can be made in the beginning of the proof:

Lemma A.9 *S_γ is inconsistent iff there is a set of clauses S derived from S_γ by basic superpositions and equality solutions such that*

1. *S does not contain equality;*
2. *S is inconsistent.*

Proof. Due to the basic restriction, there is refutation of S_γ in which all paramodulations precede all other rules, and all equality solutions precede all resolutions (see [Deg 82, DV 86b]). Let us call such refutations *regular*. In S_γ all arguments of non-equality predicates are variables. Then the basic strategy requires that all paramodulations be made into equations, thus becoming left or right superpositions. For regular refutations, equality solutions can only be made into clauses containing no positive equations, and resolution can only be applied to clauses without equations. Take as S the set of all clauses in a regular refutation which are obtained by only superpositions and equality solutions and which do not contain equality. □

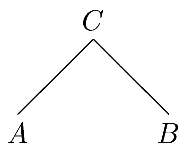
In our example a possible set S consists of the following three clauses:

$\neg A_1(x), \neg A_2(x)$
$A_1(a)$
$A_2(a)$

See Example 3.3 for derivations of $A_1(a)$ and $A_2(a)$ by basic superpositions and equality solutions. In the terminology of Section 3, $A_1(a)$ and $A_2(a)$ are *solution clauses.*

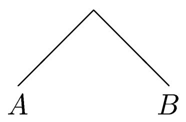
Lemma A.10 *Let S be an set of clauses without equality such that the clauses of S are the following:*

1. *All clauses $\neg A, \neg B, C$, where*



   is a branch expansion rule.

2. *All clauses $\neg A, \neg B$, where*



   is a branch expansion rule.

3. Some positive clauses.

Then S is inconsistent iff there is a derivation of the empty tableau from the initial tableau using the branch expansion rules and branch closure rules against positive clauses from S.

Proof. Introduce some notations. Denote by $\square$ both the empty clause and the initial tableau, i.e. the tableau consisting of one empty clause. By $\diamond$ we denote the empty tableau. For every set S as in the statement of this lemma, and every tableaux T_1, T_2, the figure $T_1 \overset{S}{\longmapsto} T_2$ denotes that T_2 can be obtained from T_1 by a tableau expansion or a branch closure against a clause in S. As usual, $\overset{S}{\longmapsto} *$ denotes the transitive closure of $\overset{S}{\longmapsto}$.

$\Rightarrow$ Let S be inconsistent. Then there is a refutation of S by positive hyperresolution. We use induction on the number of hyperresolution inference steps n. For simplicity we omit factoring.

$n = 1$ Then the inference step has the form

$$\frac{B_1 \qquad B_2 \qquad \neg A_1, \neg A_2}{\square}$$

such that there is a simultaneous most general unifier σ of $\{\langle A_1, B_1\rangle, \langle A_2, B_2\rangle\}$. Let δ be a most general unifier of A_1, B_1. Then the tableau proof is

$$\square \overset{S}{\longmapsto} A_1 \mid A_2 \overset{S}{\longmapsto} A_2\delta \overset{S}{\longmapsto} \diamond$$

$n > 1$ For simplicity, we consider the ground case and then apply standard lifting. Let the first step of the hyperresolution refutation is

$$\frac{A_1, C_1 \qquad A_2, C_2 \qquad \neg A_1, \neg A_2, A}{A, C_1, C_2}$$

where C_1, C_2 are clauses. Induction hypothesis gives that there is a tableau proof

$$\square \overset{S\cup\{(A,C_1,C_2)\}}{\longmapsto} * \diamond$$

Note that the clause A, C_1, C_2 can only be used in branch closures. We show that every branch closure against this clause can be eliminated. Consider such a branch closure:

$$T \mid A, C_1, C_2, C_3 \overset{S\cup\{(A,C_1,C_2)\}}{\longmapsto} T$$

It can be replaced with

$$\begin{aligned}
&T \mid A, C_1, C_2, C_3 \overset{S}{\longmapsto}\\
&\quad T \mid A, A_1, C_1, C_2, C_3 \mid A, A_2, C_1, C_2, C_3 \overset{S}{\longmapsto}\\
&\qquad T \mid A, A_2, C_1, C_2, C_3 \overset{S}{\longmapsto} T
\end{aligned}$$

where the first step is a branch expansion, and the other two steps are branch closures against A_1, C_1 and A_2, C_2.

$\Leftarrow$ Let $T = (C_1 \mid \ldots \mid C_n)$ be a tableau. The *formula image* $\mathcal{FI}(T)$ of T is the formula

$$\forall \bigwedge_{i=1}^{n} \bigvee C_i$$

Then one can show the following:

$$T_1 \overset{S}{\longmapsto} T_2 \text{ implies } S \vdash \mathcal{FI}(T_2) \supset \mathcal{FI}(T_1).$$

Since $\mathcal{FI}(\diamond) = \mathit{true}$ and $\mathcal{FI}(\Box) = \mathit{false}$ and $\Box \overset{S}{\longmapsto} *\diamond$ we have $S \vdash \mathit{true} \supset \mathit{false}$. i.e. S is inconsistent.

$\Box$

Combining Lemmas A.8, A.9 and A.10 we obtain

Theorem A.1 (Soundness) *If there is a derivation of the empty tableau from the initial tableau then γ is provable.*

Theorem A.2 (Completeness) *If γ is provable then there is a derivation of the empty tableau from the initial tableau.*

Bibliography

[BDP 89] L. Bachmair, N. Dershowitz, and D. Plaisted. Completion without failure. In H. Aït Kaci and M. Nivat, editors, *Resolution of Equations in Algebraic Structures*, volume 2, pages 1–30. Academic Press, 1989.

[BGLS 92] L. Bachmair, H. Ganzinger, C. Lynch, and W. Snyder. Basic paramodulation and superposition. In D. Kapur, editor, *11th International Conference on Automated Deduction*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 462–476, Saratoga Springs, NY, USA, June 1992. Springer Verlag.

[BG 94] L. Bachmair and H. Ganzinger. Rewrite-based equational theorem proving with selection and simplification. *Journal of Logic and Computation*, 4(3):217–247, 1994.

[BMSU 86] F. Bancilhon, D. Maier, Y. Sagiv, and J.D. Ullman. Magic sets and other strange ways to implement logic programs. In *Proceedings of the 5th ACM SIGMOD-SIGACT Symposium on Principles of Database Systems*, pages 1–15, Cambridge, MA, March 1986.

[BH 92] B. Beckert and R. Hähnle. An improved method for adding equality to free variable semantic tableaux. In D. Kapur, editor, *11th International Conference on Automated Deduction (CADE)*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 678–692, Saratoga Springs, NY, USA, June 1992. Springer Verlag.

[Bec 94a] B. Beckert. Adding equality to semantic tableaux. In K. Broda, M. D'Agostino, R. Goré, R. Jahnson, and S. Reeves, editors, *Proceedings, 3rd Workshop on Theorem Proving with Analytic Tableaux and Related Methods, Abingdon*, pages 29–41, Imperial College, London, TR-94/5, 1994.

[Bec 94b] B. Beckert. A completion-based method for mixed universal and rigid E-unification. In A. Bundy, editor, *Automated Deduction — CADE-12. 12th International Conference on Automated Deduction.*, volume 814 of *Lecture Notes in Artificial Intelligence*, pages 678–692, Nancy, France, June/July 1994.

[Bet 58] E.W. Beth. On machines which prove theorems. In J. Siekmann and G. Wrightson, editors, *Automation of Reasoning. Classical Papers on Computational Logic*, volume 1, pages 79–92. Springer Verlag, 1983. Originally appeared in 1958.

[Bet 59] E. Beth. *The Foundations of Mathematics.* North Holland, 1959.

[Boy 90] T. Boy de la Tour. Minimizing the number of clauses by renaming. In M.E. Stickel, editor, *Proc. 10th CADE*, volume 449 of *Lecture Notes in Artificial Intelligence*, pages 558–572, 1990.

[Boy 71] R.S. Boyer. *Locking: a Restriction to Resolution.* PhD thesis, Univ. of Texas, Austin, 1971.

[Bra 75] D. Brandt. Proving theorems with the modification method. *SIAM Journal of Computing*, 4:412–430, 1975.

[CP 86] P.T. Cox and T. Pietrzykowski. Incorporating equality into logic programming via surface deduction. *Annals of Pure and Applied Logic*, 31:177–189, 1986.

[Deg 79] A.I. Degtyarev. The strategy of monotone paramodulation (in Russian). In *Fifth Soviet All-Union Conference on Mathematical Logic*, page 39, Novosibirsk, 1979.

[Deg 82] A.I. Degtyarev. On the forms of inference in calculi with equality and paramodulation. In Yu.V. Kapitonova, editor, *Automation of Research in Mathematics*, pages 14–26. Institute of Cybernetics, Kiev, 1982.

[DV 86a] A.I. Degtyarev and A.A. Voronkov. Automated theorem proving I (in Russian). *Kibernetika*, (3), 1986. The journal is cover-to-cover translated to English.

[DV 86b] A.I. Degtyarev and A.A. Voronkov. Methods of handling equality in machine theorem proving (in Russian). *Kibernetika*, (3), 1986. The journal is cover-to-cover translated to English.

[DV 87] A.I. Degtyarev and A.A. Voronkov. Automated theorem proving II (in Russian). *Kibernetika*, (4), 1987. The journal is cover-to-cover translated to English.

[DV 94] A. Degtyarev and A. Voronkov. A new procedural interpretation of Horn clauses with equality. UPMAIL Technical Report 89, Uppsala University, Computing Science Department, November 1994.

[DJ 90] N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In J. Van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Methods and Semantics, chapter 6, pages 243–309. North Holland, Amsterdam, 1990.

[Ear 70] J. Earley. An efficient context-free parsing algorithm. *Communications of the ACM*, 13(2):94–102, 1970.

[Fit 90] M. Fitting. *First Order Logic and Automated Theorem Proving*. Springer Verlag, New York, 1990.

[DH 86] V.J. Digricoli and M.C. Harrison. Equality-based binary resolution. *Journal of the Association for Computing Machinery*, 33:253–289, 1986.

[GR 86] J.H. Gallier and S. Raatz. Extending SLD-resolution methods for Horn clauses with equality based on E-unification. In *Symposium on Logic Programming*, pages 168–179, 1986.

[GR 89] J.H. Gallier and S. Raatz. Extending SLD-resolution to equational Horn clauses using E-unification. *Journal of Logic Programming*, 6(3):3–44, 1989.

[GNRS 92] J. Gallier, P. Narendran, S. Raatz, and W. Snyder. Theorem proving using equational matings and rigid E-unification. *Journal of the Association for Computing Machinery*, 39(2):3–44, 1992.

[Gal 92] J. Gallier. Unification procedures in automated deduction methods based on matings: a survey. In M. Nivat and A. Podelski, editors, *Tree Automata and Languages*, pages 439–485. Elsevier Science, 1992.

[GJ 79] M.R. Garey and D.S. Johnson. *Computers and Intractability*. Freeman, San Francisco, 1979.

[Kan 63] S. Kanger. A simplified proof method for elementary logic. In J. Siekmann and G. Wrightson, editors, *Automation of Reasoning. Classical Papers on Computational Logic*, volume 1, pages 364–371. Springer Verlag, 1983. Originally appeared in 1963.

[Lif 68] V. Lifschitz. Specialized forms of derivation in predicate calculus with equality and functional symbols (in Russian). In *Trudy MIAN*, volume 98, pages 5–25. 1968. English translation in: Proc. Steklov Institute of Math., AMS, Providence, 1971.

[MAT 94] A.M. Maeda, J.-I. Aoe, and H. Tomabechi. Signature-check based unification filter. *Software — Practice and Experience*, 24(7):603–622, July 1994.

[MM 83] S.Yu. Maslov and G. Mints. The proof-search theory and the inverse method (in Russian). In Mints G., editor, *Mathematical Logic and Automatic Theorem Proving*, pages 291–314. Nauka, Moscow, 1983.

[Mas 87] S.Yu. Maslov. *Theory of Deductive Systems and its Applications.* MIT Press, 1987.

[MRL 91] J. Minker, A. Rajasekar, and J. Lobo. Theory of disjunctive logic programs. In Jean-Louis Lassez and Gordon Plotkin, editors, *Computational Logic. Essays in Honor of Alan Robinson.*, pages 613–639. The MIT Press, Cambridge, Massachusetts, 1991.

[Mor 69] J.B. Morris. E-resolution: Extension of resolution to include the equality relation. In Donald E. Walker and Lewis M. Norton, editors, *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-69)*, pages 287–294. Washington, D.C., May 1969.

[NR 92a] R. Nieuwenhuis and A. Rubio. Basic superposition is complete. In *ESOP'92*, volume 582 of *Lecture Notes in Computer Science*, pages 371–389. Springer Verlag, 1992.

[NR 92b] R. Nieuwenhuis and A. Rubio. Theorem proving with ordering constrained clauses. In D. Kapur, editor, *11th International Conference on Automated Deduction (CADE)*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 477–491, Saratoga Springs, NY, USA, June 1992. Springer Verlag.

[Ore 69] V.P. Orevkov. On non-increasing rule applications for equality (in Russian). *Zapiski Nauchnyh Seminarov LOMI*, 16:152–156, 1969. English Translation in: Seminars in Mathematics: Steklov Math. Inst. 16, Consultants Bureau, NY-London, 1971.

[Pra 60] D. Prawitz. An improved proof procedure. In J. Siekmann and G. Wrightson, editors, *Automation of Reasoning. Classical Papers on Computational Logic*, volume 1, pages 162–201. Springer Verlag, 1983. Originally appeared in 1960.

[Pet 93] U. Petermann. A framework for integrating equality reasoning into extension procedure. In *Proc. Workshop on Theorem Proving with Analytic Tableaux and Related Methods*, pages 195–207, 1993.

[Ree 87] S.V. Reeves. Adding equality to semantic tableau. *Journal of Automated Reasoning*, 3:225–246, 1987.

[Rus 91] M. Rusinowitch. Theorem proving with resolution and superposition: an extension of the Knuth and Bendix completion procedure as a complete set of inference rules. *Journal of Symbolic Computations*, 11:21–49, 1991.

[Sch 60] Schütte, K. *Beweistheorie (in German).* Springer Verlag, 1960.

[TR 94] M. Tambe and P.S. Rosenbloom. Investigating production system representations for non-combinatorial match. *Artificial Intelligence*, 68:155–199, 1994.

[Vie 89] Laurent Vieille. Recursive query processing: the power of logic. *Theoretical Computer Science*, 69(1):1–53, 1989.

[Vor 92] A.A. Voronkov. Theorem proving in non-standard logics based on the inverse method. In D. Kapur, editor, *11th International Conference on Automated Deduction*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 648–662, Saratoga Springs, NY, USA, June 1992. Springer Verlag.