MAX-PLANCK-INSTITUT FÜR INFORMATIK

Equality Reasoning in
Sequent-Based Calculi

Anatoli Degtyarev and and Andrei
Voronkov

MPI–I–98–2–011 July 1998

Im Stadtwald
D 66123 Saarbrücken
Germany

Author's Address

Anatoli Degtyarev: Computing Science Department, Uppsala University, Box 311, S-75105, Uppsala, Sweden.
anatoli@csd.uu.se, http://www.csd.uu.se/~anatoli.

Andrei Voronkov: Computing Science Department, Uppsala University, Box 311, S-75105, Uppsala, Sweden.
voronkov@csd.uu.se, http://www.csd.uu.se/~voronkov.

Publication Notes

Anatoli Degtyarev is partially supported by grants from the Swedish Royal Academy of Sciences and TFR. Andrei Voronkov is partially supported by grants from TFR and Max-Planck-Institute.

This report is a preliminary version of the article to appear in the Handbook of Automated Reasoning by Elsevier Science in 1999, edited by Alan Robinson and Andrei Voronkov.

Acknowledgements

We thank Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Gabriel Kuper.

Abstract

We overview methods of equality reasoning in sequent-based systems. We consider the history of handling equality in sequent systems, methods based on rigid E-unification, paramodulation-based methods, the equality elimination method and equality reasoning in nonclassical logics.

Keywords

Equality reasoning, sequent calculi, tableau-based theorem proving.

Contents

1 Introduction

Handling equality in resolution theorem proving is one of the central topics of research in automated reasoning. The first general method based on paramodulation was proposed by Robinson and Wos already in 1969 [134] (although there were some earlier publications based on other techniques). The treatment of equality in sequent-based machine-oriented calculi has much longer history started by Wang in 1960 [162] and Kanger in 1963 [91]. Ideas of Kanger related to equality handling were more thoroughly expressed by Lifschitz [97] and Orevkov [125].

The next generation of works in this area is based on rigid unification introduced by Gallier, Raatz and Snyder in 1987 [72]. Recently, this area witnessed a rapid development of new results and techniques, including results on rigid E-unification, the equality elimination method, rigid superposition and rigid paramodulation. It has also been discovered that rigid unification has close connections with intuitionistic logic with equality, second-order unification and some combinatorial problems.

In this article we describe automated reasoning techniques for all known sequent-based methods of automated deduction, including the tableau method, the connection method, model elimination and the inverse method. We illustrate main results and ideas on the tableau method and sometimes on the inverse method (as a typical backward and a typical forward proof search methods in sequent calculi).

The paper covers the following topics:

- methods of proof-search in sequent calculi;
- early history of sequent-based automated deduction;
- translation of logic with equality into logic without equality;
- theorem proving using simultaneous rigid E-unification;
- tableau calculi with rigid paramodulation/superposition;
- the equality elimination method; and
- equality reasoning in nonclassical logics.

Due to the rapidly growing interest in tableau-based theorem proving, and because the usual techniques of handling equality in automated reasoning do not straightforwardly generalize to tableaux, we believe that equality reasoning in tableaux and related procedures will be among the central topics in automated deduction in the nearest future. We present many new results that have not yet been presented in any systematic way. We do not assume that readers have any special knowledge of equality reasoning.

In this section we introduce the reader in the area of equality reasoning. In Section 1.1 we introduce the main notation used in the paper. In Section 1.2 we discuss the standard axiomatization of equality. In Section 1.2 we consider several techniques of handling equality in resolution-based procedures and discuss various versions of the paramodulation rule. In Section 1.4 we introduce the main sequent calculus for equality reasoning $\mathbf{LK}^=$ which corresponds to the ground version of semantic tableaux.

1.1 Some useful notation

We assume some knowledge of most fundamental notions of mathematical logic, like those of a term or a formula. They may be found in the standard textbooks on mathematical logic, e.g. Kleene [92], Smullyan [144], Chang and Lee [95], Gallier [67] or Fitting [64].

- *Formulas* are defined as usual, using atomic formulas, the connectives $\wedge, \vee, \supset, \neg$ and the quantifiers $\forall, \exists$.
- A *literal* is either an atomic formula or a negation of an atomic formula.
- A *clause* is a finite multiset of literals, denoted $L_1, \ldots, L_n$.
- The *empty clause* is denoted $\Box$.

Atomic formulas will also be called *atoms*. If L is a literal and $C = L_1, \ldots, L_n$ is a clause, then L, C will denote the clause $L, L_1, \ldots, L_n$ and similar for C, L. Alternatively, we shall sometimes consider a clause $L_1, \ldots, L_n$ as the formula $L_1 \vee \ldots \vee L_n$.

We use the following notation for substitutions and unification:

- *substitutions* will be denoted by $[x_1 \mapsto t_1, \ldots, x_n \mapsto t_n]$;
- the *empty substitution* will be denoted ε;

- the *domain* of a substitution θ, i.e. the set of variables $\{x \mid x\theta \neq x\}$ will be denoted by $dom(\theta)$;
- the *application* of a substitution $\theta = [x_1 \mapsto t_1, \dots, x_n \mapsto t_n]$ to any expression E (i.e. term, formula, clause, multiset of formulas etc.) is the expression obtained by the simultaneous replacement of free occurrences of $x_1, \dots, x_n$ by $t_1, \dots, t_n$, respectively, with renaming, if necessary, those bound variables in E that coincide with a variable occurring in any t_i. The result of the application is denoted by $E\theta$.
- The *composition* of substitutions σ and θ, denoted $\sigma\theta$, is the substitution defined by $x(\sigma\theta) \rightleftharpoons (x\sigma)\theta$, for every variable x.
- A substitution σ is *more general* than a substitution θ, denoted $\sigma \leq \theta$, if and only if there exists a substitution τ such that $\sigma\tau = \theta$.
- A term, literal, or clause is called *ground*, if it contains no variables.
- A formula is called *closed*, if it contains no free occurrences of variables.
- The *universal (respectively, existential) closure* of a formula φ, denoted $\forall\varphi$ (respectively, $\exists\varphi$), is the formula $\forall x_1 \dots \forall x_n \varphi$ (respectively, $\exists x_1 \dots \exists x_n \varphi$), where $x_1, \dots, x_n$ are all free variables of φ in the order of their occurrence in φ.
- A substitution σ is called *grounding* for a set of variables V if for every variable $v \in V$ the term $v\sigma$ is ground.

The notation $\rightleftharpoons$ stands for "equal by definition". We denote the equality predicate symbol by $\approx$. For any expression E (like clause, formula or a multiset of formulas), the set $vars(E)$ is defined as the set of all (free) variables occurring in E.

- A clause C is called an *instance* of a clause D if there is a substitution σ such that $C\sigma = D$;
- A clause C is called a *variant* of a clause D if C is an instance of D and D is an instance of C.
- An *equation* is any formula of the form $s \approx t$;

- A *disequation* is any formula of the form $\neg(s \approx t)$, denoted $s \not\approx t$.

We do not distinguish the formula $s \approx t$ from $t \approx s$.

We denote

- *constants* by a, b, c, d, e;
- *variables* by x, y, z, u, v, w;
- *function symbols* by f, g, h;
- *predicate symbols* by P, Q, R;
- *terms* by p, q, r, s, t;
- *atoms* by A, B;
- *literals* by L, M, N;
- *clauses* by C, D;
- *formulas* by φ, χ, ψ;
- *sets or multisets of formulas* by $\Gamma, \Delta, \Sigma, \Xi$;
- *substitutions* by $\theta, \sigma, \delta, \rho$,

maybe with indices.

We denote

- $\mathrm{mgu}(s, t)$ any idempotent most general unifier of terms s and t; and
- $\mathrm{mgu}(\langle s_1, \ldots, s_n \rangle, \langle t_1, \ldots, t_n \rangle)$ any idempotent simultaneous most general unifier of s_1 and t_1, ... , s_n and t_n.
- We assume that all terms are written in a *fixed finite function signature* $\mathcal{F}$,

unless the opposite is explicitly stated. *Constants* are function symbols of arity 0. We denote

- $T_{\mathcal{F}}$ the set of ground terms in the signature $\mathcal{F}$;
- $T_{\mathcal{F}}(X)$ the set of terms in the signature $\mathcal{F}$ with variables in a set X.

To avoid expressions with many parentheses, we shall denote applications of unary functions to arguments without parentheses, for example fgx instead of $f(g(x))$.

1.2 Equality. The first axiomatization

The *equality predicate* $\approx$ plays a special role in automated reasoning. Mathematical theorems often contain the equality predicate and assume that it satisfies special *equality axioms.* The same is true for many areas of computer science where first-order logic is used.

The necessity of handling equality in automated reasoning has been recognized already in the very early papers in the area (Wang [162], Kanger [91], Wos et.al. [164], Robinson [136], Darlington [36] and Robinson and Wos [134]). Equality in first-order logic can be axiomatized by the following *equality axioms*:

- *reflexivity* axiom: $x \approx x$;
- *symmetry* axiom: $x \approx y \supset y \approx x$;
- *transitivity* axiom: $x \approx y \wedge y \approx z \supset x \approx z$;
- *function substitution* axioms: $x_1 \approx y_1 \wedge \ldots \wedge x_n \approx y_n \supset f(x_1, \ldots, x_n) \approx f(y_1, \ldots, y_n)$, for every function symbol f;
- *predicate substitution* axioms: $x_1 \approx y_1 \wedge \ldots \wedge x_n \approx y_n \wedge P(x_1, \ldots, x_n) \supset P(y_1, \ldots, y_n)$ for every predicate symbol P.

In principle, for a theorem φ with equality one can add the conjunction χ of these equality axioms as a premise and try to prove $\forall \chi \supset \varphi$ by any existing method for logic without equality. However, this leads to a combinatorial explosion due to the universal applicability of equality axioms.

Example 1.1 Suppose that $\mathcal{F}$ contains a binary function symbol f. Then from $a \approx b$ we can derive any equation of the form

$$f(s_1, f(s_2, \ldots, f(s_{n-1}, s_n) \ldots)) \approx f(t_1, f(t_2, \ldots, f(t_{n-1}, t_n) \ldots))$$

such that $\{s_1, \ldots, s_n, t_1, \ldots, t_n\} \subseteq \{a, b\}$. For example, $f(a, f(a, b)) \approx f(a, f(b, a))$ has the following derivation by positive hyperresolution:

$$\frac{a \approx a \qquad \dfrac{a \approx b \quad b \approx a \quad x_1 \not\approx y_1 \vee x_2 \not\approx y_2 \vee f(x_1, x_2) \approx f(y_1, y_2)}{f(a, b) \approx f(b, a)}, \quad x_1 \not\approx y_1 \vee x_2 \not\approx y_2 \vee f(x_1, x_2) \approx f(y_1, y_2)}{f(a, f(a, b)) \approx f(a, f(b, a))}$$

where $b \approx a$ and $a \approx a$ have the folowing derivations:

$$\frac{a \approx b \quad x \not\approx y \vee y \approx x}{b \approx a} \; ;$$

$$\frac{a \approx b \quad b \approx a \quad x \not\approx y \vee y \not\approx z \vee x \approx z}{a \approx a} \; .$$

Thus, even the early methods of handling equality in automated reasoning tried to avoid the use of the equality axioms. In the rest of this section we shall consider some of the main ideas developed in resolution-based theorem proving with equality.

1.3 Paramodulation and its refinements

► We write $\varphi[s]$ to indicate that the formula φ contains an occurrence of the term s such that all occurrences of variables in s are free in φ.

We also denote by $\varphi[t]$ the result of replacing this particular occurrence of s in φ by t. We shall use similar notation for occurrences of subterms in terms $r[s]$ and terms in sets or multisets of formulas $\Gamma[s]$.

In first-order logic, the rule or replacement of equal by equal is a derived rule:

$$\frac{\varphi[s] \quad s \approx t}{\varphi[t]} \; .$$

A suitable formulation of this rule can alternatively be used to replace all equality axioms except for reflexivity. Later, we shall give some sequent systems for first-order logic using such a replacement rule. In automated reasoning, a rule of this form has already been used in Wang [162].

A modification of this rule for resolution-based reasoning known as *paramodulation* has been introduced by Robinson and Wos [134]. The paramodulation rule is defined on clauses and uses most general unifiers.

► *Paramodulation* is the following inference rule:

$$\frac{L[s'] \vee C_1 \quad s \approx t \vee C_2}{(L[t] \vee C_1 \vee C_2)\mathrm{mgu}(s, s')} \; (par) \qquad (1)$$

Robinson and Wos [133] proved completeness of the system consisting of resolution, factoring and paramodulation only in presence of an additional

- *function reflexivity axiom*: $f(x_1, \dots, x_n) \approx f(x_1, \dots, x_n)$, for every function symbol $f \in \mathcal{F}$.

This axiom seems very similar to the reflexivity axiom $x \approx x$, but there is a fundamental difference important for understanding equality reasoning in general. In order to explain the meaning of the function reflexivity axiom we first show that paramodulation rule leads to *nonliftable derivations*. A standard technique of proving completeness of various methods or strategies in automated deduction is to prove the existence of a ground derivation and then to make *lifting* to the nonground case. Lifting for an inference system $\mathcal{R}$ is a technique for proving completeness that can abstractly be explained in the following way.

- Let an inference system $\mathcal{R}$ have the following property. Suppose that $C_1, \dots, C_n$ be clauses and $C'_1, \dots, C'_n$ be their ground instances. Furthermore, suppose that

  $$\frac{C'_1 \quad \cdots \quad C'_n}{C'}$$

  is an inference of $\mathcal{R}$, where C' is a ground clause. Then there exists a clause C such that C' is an instance of C and such that C is derivable from $C_1, \dots, C_n$ in $\mathcal{R}$. Then $\mathcal{R}$ is said to have the *lifting property*.

The logical system consisting of resolution and factoring, as well as many modifications of resolution, has the lifting property. However, the addition of paramodulation leads to nonliftable derivations. Indeed, consider two clauses $P(x, x)$ and $a \approx b$ and their ground instances $P(fa, fa)$ and $a \approx b$. Consider the following ground derivation:

$$\frac{P(fa, fa) \quad a \approx b}{P(fa, fb)}\ (par) \tag{2}$$

This derivation is nonliftable.

The system consisting of resolution, factoring, paramodulation *and the function reflexivity axiom* has the lifting property. For example, in this system the above derivation can be lifted as follows, using the axiom $fx \approx fx$:

$$\frac{P(x,x) \qquad \dfrac{fx \approx fx \quad a \approx b}{fa \approx fb}\ (par)}{P(fa, fb)}\ (par)$$

Although function reflexivity leads to a liftable system, it is hard to find any reasonable implementation of function reflexivity, as Example 1.1 demonstrates. As it was shown later, resolution with factoring and paramodulation is complete *without* function reflexivity (Brand [33], Peterson [128]). However, the understanding of the nature of function reflexivity is important for understanding some problems arising in automating sequent systems with equality. The effect of the function reflexivity axioms is *the possibility of substituting almost an arbitrary term for a variable.* For example, to lift ground derivation (2), we had to substitute fa for x, which can be achieved by using the instance $fx \approx fx$ of the function reflexivity axiom.

Getting rid of function reflexivity drastically reduces search space in methods based on resolution and paramodulation. However, unrestricted applications of paramodulation also lead to combinatorial explosion because of a too general applicability of paramodulation. Further history of paramodulation-based theorem proving had been aiming at restricting the applicability of the paramodulation rule. Among the restrictions, we shall briefly consider the following:

1. paramodulation into a variable is prohibited;

2. the use of reduction orderings;

3. the basic strategy of paramodulation;

4. simplification.

Many of these refinements of paramodulation aimed at the development of "a refutation complete set of inference rules for all first-order logic with equality which reduces to the Knuth-Bendix procedure when restricted to equality units" (Peterson [128]). This refers to the famous completion procedure of Knuth and Bendix [93] for solving word problems in universal algebras.

The history of the development of these and other paramodulation restrictions can be seen from Brand [33], Degtyarev [38, 39], Peterson [128], Degtyarev and Voronkov [46], Hsiang and Rusinowitch [86, 87], Pais and Peterson [126], Zhang and Kapur [165], Bachmair and Ganzinger [8, 10], Rusinowitch [138], Bachmair et.al. [12, 13] and Nieuwenhuis and Rubio [122, 123], Lynch [103], Bachmair and Ganzinger [11].

We can avoid applying *paramodulation into a variable*:

$$\frac{L[x] \vee C_1 \quad s \approx t \vee C_2}{(L[t] \vee C_1 \vee C_2)[x \mapsto s]}\ (par)$$

Note that such an application of paramodulation is *always possible* when we have a nonground clause $L[x] \vee C_1$, because x is unifiable with s. It is known that the system consisting of paramodulation, resolution and factoring is complete when paramodulation into a variable is prohibited (i.e. the term s' in paramodulation rule (1) is not a variable) (Brand [33], Peterson [128])[1]

Often, paramodulation allows one to repeatedly apply the same equation obtaining larger and larger terms.

Example 1.2 The equation $x \approx fx$ can repeatedly be applied as in the derivation below:

$$\cfrac{\cfrac{Pa \quad x \approx fx}{P(fa)}\ (par) \qquad x \approx fx}{P(ffa)}\ (par)$$

$$\begin{array}{c} \vdots \\ P(f \dots fa) \end{array}$$

obtaining longer and longer literals $P(f^n a)$.

Such *increasing applications* of paramodulations can be avoided by the introduction of ordering restrictions based on *reduction orderings*.

▶ A *reduction ordering* on $T_{\mathcal{F}}(X)$ is any ordering $\succ$ on $T_{\mathcal{F}}(X)$ such that

1. $\succ$ is well-founded;
2. if $s \succ t$ then $r[s\sigma] \succ r[t\sigma]$, for all terms s, t, r and substitutions σ.

[1] Strictly speaking, neither Brand nor Peterson could entirely prohibit paramodulations into variables. This problem was comletely solved later by Bachmair et.al. [12] and Nieuwenhuis and Rubio [122].

- A reduction ordering is called a *simplification ordering* if for any term $t[s]$ such that s is a proper subterm of t we have $t[s] \succ s$.

The results involving reduction orderings in this paper are true for any reduction ordering total on $T_{\mathcal{F}}$.

Reduction orderings are used for two main purposes: restricting paramodulation and simplification. Here and below we write

- $s \succeq t$ to denote that $s \succ t$ or $s = t$.

The simplest ordering restriction on paramodulation gives us

- *ordered paramodulation* that can be formulated as follows:

$$\frac{L[s'] \vee C_1 \quad s \approx t \vee C_2}{(L[t] \vee C_1 \vee C_2)\sigma}$$

such that

1. $\sigma = \mathrm{mgu}(s, s')$;
2. s' is not a variable;
3. $t\sigma \not\succeq s\sigma$.

Note that when $s\sigma$ and $t\sigma$ are ground terms and $\succ$ is linear on ground terms, the last condition is equivalent to $s\sigma \succ t\sigma$. Ordered paramodulation has been originally considered in Peterson [128] and Hsiang and Rusinowitch [86].

If we use ordered paramodulation, no derivation from $P(a)$ and $x \approx fx$ is possible (compare with Example 1.2).

A stronger restriction is the so-called *maximal paramodulation* (in the terminology of Pais and Peterson [126]) that takes into attention not only ordering on terms but also ordering on literals of clauses.

- *Maximal paramodulation* is the following inference rule:

$$\frac{L[s'] \vee C_1 \quad s \approx t \vee C_2}{(L[t] \vee C_1 \vee C_2)}$$

such that

1. $\sigma = \mathrm{mgu}(s, s')$;
2. term s' is not a variable;
3. $t\sigma \not\succeq s\sigma$;
4. $L[s']\sigma$ is maximal w.r.t. $\succ$ in $(L[s'] \vee C_1)\sigma$;
5. $(s \approx t)\sigma$ is maximal w.r.t. $\succ$ in $(s \approx t \vee C_2)\sigma$.

The next refinement of paramodulation is known as *superposition.* The notion of superposition comes from Knuth and Bendix [93], where superposition has been defined for positive unit clauses (equations). The above definition of maximal paramodulation can be strengthened to the definition of superposition in the following way. For simplicity, we assume that all atoms are equations. Then, the literal $L[s']$ in the paramodulation rule has either the form $p[s'] \approx q$ or $p[s'] \not\approx q$. We only consider the former case.

► *Superposition* is the following inference rule:

$$\frac{p[s'] \approx q \vee C_1 \quad s \approx t \vee C_2}{(p[t] \approx q \vee C_1 \vee C_2)\sigma}$$

such that

1. $\sigma = \mathrm{mgu}(s, s')$;
2. s' is not a variable;
3. $t\sigma \not\succeq s\sigma$;
4. $(p[s'] \approx q)\sigma$ is maximal w.r.t. $\succ$ in $(p[s'] \approx q \vee C_1)\sigma$;
5. $(s \approx t)\sigma$ is maximal w.r.t. $\succ$ in $(s \approx t \vee C_2)\sigma$;
6. $q\sigma \not\succeq p[s']\sigma$.

Calculus with such rule was proposed by Zhang and Capur in [165]. If we drop condition (5), we come to *extended superposition* (Rusinowitch [138]). There are further refinements of superposition, for example *strict superposition* of Bachmair and Ganzinger [8, 9, 11].

Recent results in this area are related to the so-called *basic strategy.* The idea of the basic strategy is to forbid paramodulation in terms introduced by substitutions applied during previous inference steps. The basic strategy (without ordering restrictions) has been explicitly introduced for the first time in Degtyarev [38, 39] by the name of *monotone paramodulation.* It has also been described in Degtyarev and Voronkov [46] in terms of the so-called *conditional clauses* which are now known by the name of *equality constraint clauses* (Nieuwenhuis and Rubio [121]). Such clauses have been defined in Degtyarev and Voronkov [46] as pairs $C \cdot \mathcal{C}$ where C is a clause and $\mathcal{C}$ is a set of equations $\{s_1 \approx t_1, \ldots, s_n \approx t_n\}$. The equations $s_i \approx t_i$ in $\mathcal{C}$ are considered as constraints to be solved by a *simultaneous most general unifier* of (s_i, t_i), i.e. a substitution σ such that $s_i\sigma = t_i\sigma$ for all $i \in \{1, \ldots, n\}$. Thus, the substitution σ introduced by the previous inference steps has been divided from the "skeleton" C of the clause $C\sigma$. Following [39], Degtyarev and Voronkov [46] described two forms of *monotone paramodulation:*

$$\frac{L[s'] \vee C_1 \cdot \mathcal{C}_1 \quad s \approx t \vee C_2 \cdot \mathcal{C}_2}{L[t] \vee C_1 \vee C_2 \cdot \mathcal{C}_1 \cup \mathcal{C}_2 \cup \{s' \approx s\}}$$

and

$$\frac{L[s'] \vee C_1 \cdot \mathcal{C}_1 \quad s \approx t \vee C_2 \cdot \mathcal{C}_2}{L[z] \vee C_1 \vee C_2 \cdot \mathcal{C}_1 \cup \mathcal{C}_2 \cup \{s' \approx s, z \approx t\}}$$

where in both rules s' is not a variable and in the second rule z is a new variable[2].

The basic strategy with ordering restrictions has later been independently proposed in Nieuwenhuis and Rubio [121, 122] and Bachmair et.al. [12] as an extension of a *basic narrowing* technique proposed first in [88] in the Knuth-Bendix framework to refutational theorem proving with arbitrary clauses. In these papers the basic strategy has been applied to superposition, i.e. combined with ordering restrictions. The ordering restrictions can be elegantly added by extending equational constraints to *ordering constraints.* A variant of basic superposition with *ordering constraint inheritance* proposed in Nieuwenhuis and Rubio [122, 123] is defined and used in Section 6.2. Bachmair et.al. [13] described further refinements of the basic strategy which included *term selection functions* and *redex ordering.*

Resolution-based theorem proving is usually based on the *saturation procedures*. Such procedures start with a set of clauses S, adding to this set new clauses obtained by application of inference rules to clauses in S. The

[2] In the second form, after paramodulation into s', subsequent paramodulations are only possible into positions "orthogonal" to s' or into positions of superterms of s' in $L[s']$. Hence the name "monotone paramodulation".

search space for such procedures grows rapidly. Saturation procedures become much more efficient when they are augmented with *redundancy criteria* allowing us to remove redundant clauses from the search space. Examples of redundancy deletion rules are *subsumption* and *simplification*. Subsumption was introduced by Robinson [135] for the general resolution and simplification has been introduced for equality reasoning by Knuth and Bendix [93]. In order to formalize simplification, we introduce inference rules working on multisets of clauses. Suppose S is a multiset of clauses. Then

▶ *simplification* is the following inference rule:

$$\frac{S \cup \{L[s'] \vee C, s \approx t\}}{S \cup \{L[t\sigma] \vee C, s \approx t\}} \tag{3}$$

such that $s\sigma = s'$, $s\sigma \succ t\sigma$ and $L[s\sigma] \succ (s\sigma \approx t\sigma)$ (Peterson [128]).

The meaning of this inference rule is that s' can be *replaced* by $t\sigma$, thus discarding the clause $L[s'] \vee C$. Simplification has been recognized a very strong strategy for equational reasoning. However, until very recently, simplification has not been introduced in sequent-based methods. There are other formulations of simplification, where the condition $L[s\sigma] \succ (s\sigma \approx t\sigma)$ is replaced by other conditions (e.g. Rusinowitch [138] and Bachmair and Ganzinger [10]). As far as we know, these condition are needed in order to apply the corresponding completeness proofs. It is not known whether these conditions are necessary.

Discussion of various aspects of redundancy criteria can be found in Wos et.al. [164] Knuth and Bendix [93], Slagle [143], Lankford [94], Loveland [101], Peterson [128], Wos, Overbeek and Lusk [163], Rusinowitch [138], Lusk [102], Voronkov [155], Bachmair and Ganzinger [10], Bachmair et.al. [13], Nieuwenhuis and Rubio [123], Tammet [147] and Mints, Orevkov and Tammet [117], Lynch [103], Bachmair and Ganzinger [11].

In all examples of this paper we shall use a special kind of reduction ordering, called the *lexicographic path ordering*.

▶ A *precedence relation* on $\mathcal{F}$ is any total ordering on $\mathcal{F}$.

▶ Let $\succ_{\mathcal{F}}$ be a precedence relation on $\mathcal{F}$. The *lexicographic path ordering* $\succ$ induced by $\succ_{\mathcal{F}}$ is the ordering on terms defined recursively as follows:

$t \succ x$ if $x \neq t$ and x occurs in t;

It is not the case that $x \succ t$;

Let $s = f(s_1, \dots, s_m)$ and $t = g(t_1, \dots, t_n)$. Then $s \succ t$ if one of the following is true:

$s_i \succeq t$, for some i with $1 \leq i \leq m$;

$f \succ_{\mathcal{F}} g$ and $s \succ t_j$, for all j with $1 \leq j \leq n$;

$f = g$, $(s_1, \dots, s_m) \succ^{lex} (t_1, \dots, t_n)$ and $s \succ t_j$, for all j with $1 \leq j \leq n$,

where $(s_1, \dots, s_m) \succ^{lex} (t_1, \dots, t_m)$ if there is $j \leq m$ such that $s_j \succ t_j$ and for all $i < j$ we have $s_i = t_i$.

The reader can check that lexicographic path ordering is a simplification ordering, and that it is total on $T_{\mathcal{F}}$ when $\succ_{\mathcal{F}}$ is total on $\mathcal{F}$.

1.4 Equality in sequent systems

In this section we introduce the sequent calculus $\mathbf{LK}^=$ for classical first-order logic with equality. We also explain the relation between ground version of semantic tableaux and derivations in $\mathbf{LK}^=$.

- A *sequent* is any expression of the form $\Gamma \to \Delta$ where Γ, Δ are multisets of formulas.
  - Γ (respectively Δ) is called the *antecedent* (respectively, the *succedent*) of the sequent $\Gamma \to \Delta$.
  - A sequent $\Gamma \to \Delta$ is *closed* if all formulas in Γ, Δ are closed.

We shall denote sequents by $\mathcal{S}$, maybe with indices.

The intuitive semantics of a sequent $\varphi_1, \dots, \varphi_n \to \psi_1, \dots, \psi_m$ is

$$\bigwedge_{i \in \{1,\dots,n\}} \varphi_i \supset \bigvee_{j \in \{1,\dots,m\}} \psi_m.$$

Thus, this sequent is inconsistent if and only if all φ_i are true and all ψ_j are false. For this reason, instead of sequents one can dually consider multisets of *signed formulas*, where *signs* T and F correspond to “true” and “false”. For example, the above sequent can alternatively be represented as the multiset of signed formulas $T\,\varphi_1, \dots, T\,\varphi_n, F\,\psi_1, \dots, F\,\psi_n$. Conventional formalizations of the tableau method deal with signed formulas. For convenience, we shall sometimes alternatively consider sequents as multisets of signed formulas. Multisets of signed formulas will also be denoted by Γ, Δ, maybe with indices.

▶ The *sequent calculus* $\mathbf{LK}^=$ is shown in Figure 1.

This calculus is similar to the calculi introduced by Kanger [90, 91].

In view of the relation between sequents and multisets of signed formulas, the calculus $\mathbf{LK}^=$ can be reformulated in terms of signed formulas. Inference rules working on signed formulas are traditionally classified into α-, β-, γ-, δ- and $\neg$-*rules*.

The correspondence is given in the following table:

Signed formulas	Sequents
α	$(\wedge \to)$, $(\to \vee)$, $(\to \supset)$
β	$(\to \wedge)$, $(\vee \to)$, $(\supset \to)$
γ	$(\forall \to)$, $(\to \exists)$
δ	$(\to \forall)$, $(\exists \to)$
$\neg$	$(\to \neg)$, $(\neg \to)$

The calculus $\mathbf{LK}^=$ represented for signed formulas is shown in Figure 2.

The ordinary $\mathbf{LK}^=$ derives sequents, $\mathbf{LK}^=$ for signed formulas derives multisets of formulas. Derivations of $\mathbf{LK}^=$ for signed formulas are sometimes represented in the form of so-called *tableaux* (see Fitting [64]). We shall give examples below.

▶ We write $\Gamma \vdash \varphi$ to denote that the sequent $\Gamma \to \varphi$ is derivable in $\mathbf{LK}^=$.

▶ We speak about *derivations* or *derivability* of a formula φ meaning derivations or derivability of the sequent $\to \varphi$.

▶ A derivation Π in $\mathbf{LK}^=$ is called *regular* if it satisfies the following properties:

1. equality rules are applied before all other rules, i.e. above applications of $(\approx)$ in Π there can only be applications of $(\approx)$, (Ax) and (refl);

2. in the rules ($\approx$), the occurrences of t replaced by s are in atomic formulas only.

We give an example which will be used several times in this article.

In order to simplify derivations, we shall sometimes use a modified form of the rules ($\to \exists$) and (γ) without repeating their principal formula $\exists x\varphi$ in the premises, i.e., use the rules

$$\frac{\Gamma \to \Delta, \varphi[x \mapsto t]}{\Gamma \to \Delta, \exists x\varphi}\,(\to \exists) \quad \text{and} \quad \frac{\Gamma, T\,\forall x\varphi, T\,\varphi[x \mapsto t]}{\Gamma, T\,\forall x\varphi}\,(\gamma).$$

Example 1.3 (Main example, sequent calculus) Assume that we want to prove the formula

$$\exists xyuv\big((a \approx b \supset g(x,u,v) \approx g(y,fc,fd)) \land \\ (c \approx d \supset g(u,x,y) \approx g(v,fa,fb))\big).$$

We show its regular derivation in $\mathbf{LK}^=$ in the following three forms:

1. In $\mathbf{LK}^=$ using sequents in Figure 3;

2. In $\mathbf{LK}^=$ using multisets of signed formulas (without duplicating) in Figure 4;

3. In a ground version of a tableau system in Figure 5.

Tableaux are trees whose nodes are signed formulas. A tableau can be though of as a multiset of *branches*, every branch is understood as a multiset of formulas occurring in this branch. A tableau $\mathcal{T}$ represents a $\mathbf{LK}^=$-derivation D such that the leaves of D are the branches of the tableau. We can note the following inessential difference between tableaux and derivations in $\mathbf{LK}^=$. When we counterapply an α-rule in $\mathbf{LK}^=$, for example

$$\frac{T\,a \approx b, F\,g(fa,fc,fd) \approx g(fb,fc,fd)}{F\,a \approx b \supset g(fa,fc,fd) \approx g(fb,fc,fd)}\,(\alpha),$$

we *remove* the principal formula (in this case $F\,a \approx b \supset g(fa,fc,fd) \approx g(fb,fc,fd)$) from the premise, while it remains on the tableau branch.

As can be seen from these examples, sequent derivations, corresponding derivations on the multisets of signed formulas and the ground version of semantic tableaux are different ways of representing derivations in $\mathbf{LK}^=$. For the rest of this paper, we shall mostly use $\mathbf{LK}^=$ in the form of signed formulas.

Many techniques considered in this paper are implicitly based on the fact that we can restrict ourselves by regular derivations:

Theorem 1.4 (existence of regular derivations in $\mathbf{LK}^=$) Any sequent derivable in $\mathbf{LK}^=$ has a regular derivation.

For simplicity, in this paper we do not consider derivations in the method of matings as described by Andrews [1, 2, 3] or the connection method as described by Bibel and Schreiber [31] and Bibel [28, 29]. For those, who is used to connections or matings, we shall give the same example in Section 3, after we consider the free variable version of semantic tableaux.

- ▶ A formula is said to be in *Skolem negation normal form* if it is constructed from literals using the connectives $\wedge, \vee$ and the quantifier $\exists$.

There is a provability-preserving translation of formulas without equivalences into formulas in skolem negation normal form consisting of the standard skolemization and a translation into negation normal form used, e.g. in Andrews [2].

Sequent systems based on signed formulas avoid some redundancies by joining similar inference rules under one name, for example, α-rule. For formulas in negation normal form, there are more elegant sequent systems. One of such systems has already been described in Schütte [140]. First, for formulas in negation normal form signs are not needed any more. Instead of using signs, we can consider all formulas in a sequent as having the sign F by considering axioms of the form $\Gamma, A, \neg A$ instead of $\Gamma, T\,A, F\,A$. Second, there is only one inference rule for each connective: all α-rules become a rule for disjunction, all β-rules become a rule for conjunction, γ- and δ-rules become quantifier rules (see Figure 6). The Schütte system did not even have the $(\vee)$-rule, because he considered disjunctions of formulas instead of sequents.

For formulas in skolem negation normal forms, we do not need the $(\forall)$-rule. Sequent systems for formulas in skolem negation normal forms can further be refined. For example in Voronkov [154] a sequent system is defined whose only rule is the $(\wedge)$-rule.

In this paper, when it concerns classical logic, we only work with skolemized formula to avoid the use of δ-rules (i.e. the rules $(\rightarrow \forall)$ and $(\exists \rightarrow)$) of $\mathbf{LK}^{=}$. Various forms of δ-rules have been considered in Beckert, Hähnle and Schmitt [23], Hähnle and Schmitt [85], Baaz and Leitsch [7], Baaz and Fermüller [6]. We cannot, however, avoid using δ-rules in sequent calculi for intuitionistic logic (see Section 9).

$$\frac{}{\Gamma, A \to \Delta, A}\ (Ax)$$

$$\frac{}{\Gamma \to \Delta, t{\approx}t}\ (\mathit{refl}) \qquad \frac{\Gamma[x \mapsto t], s \approx t \to \Delta[x \mapsto t]}{\Gamma[x \mapsto s], s \approx t \to \Delta[x \mapsto s]}\ (\approx)$$

$$\frac{\Gamma, \varphi, \psi \to \Delta}{\Gamma, \varphi \wedge \psi \to \Delta}\ (\wedge \to) \qquad \frac{\Gamma \to \Delta, \varphi \quad \Gamma \to \Delta, \psi}{\Gamma \to \Delta, \varphi \wedge \psi}\ (\to \wedge)$$

$$\frac{\Gamma, \varphi \to \Delta \quad \Gamma, \psi \to \Delta}{\Gamma, \varphi \vee \psi \to \Delta}\ (\vee \to) \qquad \frac{\Gamma \to \Delta, \varphi, \psi}{\Gamma \to \Delta, \varphi \vee \psi}\ (\to \vee)$$

$$\frac{\Gamma, \psi \to \Delta \quad \Gamma \to \Delta, \varphi}{\Gamma, \varphi \supset \psi \to \Delta}\ (\supset\to) \qquad \frac{\varphi, \Gamma \to \Delta, \psi}{\Gamma \to \Delta, \varphi \supset \psi}\ (\to\supset)$$

$$\frac{\Gamma \to \Delta, \varphi}{\Gamma, \neg\varphi \to \Delta}\ (\neg \to) \qquad \frac{\varphi, \Gamma \to \Delta}{\Gamma \to \Delta, \neg\varphi}\ (\to \neg)$$

$$\frac{\Gamma, \forall x\varphi \to \Delta, \varphi[x \mapsto t]}{\Gamma, \forall x\varphi \to \Delta}\ (\forall \to) \qquad \frac{\Gamma \to \Delta, \varphi[x \mapsto y]}{\Gamma \to \Delta, \forall x\varphi}\ (\to \forall)$$

$$\frac{\Gamma, \varphi[x \mapsto y] \to \Delta}{\Gamma, \exists x\varphi \to \Delta}\ (\exists \to) \qquad \frac{\Gamma \to \Delta, \exists x\varphi, \varphi[x \mapsto t]}{\Gamma \to \Delta, \exists x\varphi}\ (\to \exists)$$

In the rule (Ax), A is an atomic formula. In the rules $(\to \forall)$ and $(\exists \to)$ the variable y has no free occurrences in the conclusions of the rules. The variable y is called the *eigenvariable* of these rules. This condition on the rules $(\to \forall)$ and $(\exists \to)$ is called the *eigenvariable condition*.

Figure 1: Calculus $\mathbf{LK}^{=}$

$$\frac{}{\Gamma, T\,A, F\,A}\,(Ax) \qquad \frac{}{\Gamma, F\,t \approx t}\,(\mathit{refl}) \qquad \frac{\Gamma[x \mapsto t], T\,s \approx t}{\Gamma[x \mapsto s], T\,s \approx t}\,(\approx)$$

$$\frac{\Gamma, T\,\varphi, T\,\psi}{\Gamma, T\,\varphi \wedge \psi}\,(\alpha) \qquad \frac{\Gamma, F\,\varphi \quad \Gamma, F\,\psi}{\Gamma, F\,\varphi \wedge \psi}\,(\beta)$$

$$\frac{\Gamma, T\,\varphi \quad \Gamma, T\,\psi}{\Gamma, T\,\varphi \vee \psi}\,(\beta) \qquad \frac{\Gamma, F\,\varphi, F\,\psi}{\Gamma, F\,\varphi \vee \psi}\,(\alpha)$$

$$\frac{\Gamma, T\,\psi \quad \Gamma, F\,\varphi}{\Gamma, T\,\varphi \supset \psi}\,(\beta) \qquad \frac{\Gamma, T\,\varphi, F\,\psi}{\Gamma, F\,\varphi \supset \psi}\,(\alpha)$$

$$\frac{\Gamma, F\,\varphi}{\Gamma, T\,\neg\varphi}\,(\neg) \qquad \frac{\Gamma, T\,\varphi}{\Gamma, F\,\neg\varphi}\,(\neg)$$

$$\frac{\Gamma, T\,\forall x\varphi, T\,\varphi[x \mapsto t]}{\Gamma, T\,\forall x\varphi}\,(\gamma) \qquad \frac{\Gamma, F\,\varphi[x \mapsto y]}{\Gamma, F\,\forall x\varphi}\,(\delta)$$

$$\frac{\Gamma, T\,\varphi[x \mapsto y]}{\Gamma, T\,\exists x\varphi}\,(\delta) \qquad \frac{\Gamma, F\,\exists x\varphi, F\,\varphi[x \mapsto t]}{\Gamma, F\,\exists x\varphi}\,(\gamma)$$

In the rule (Ax), A is an atomic formula. Both rules (δ) satisfy the eigenvariable condition.

Figure 2: Calculus $\mathbf{LK}^{=}$ for signed formulas

$$
\begin{prooftree}
\AxiomC{}
\RightLabel{(refl)}
\UnaryInfC{$a \approx b \rightarrow g(fa, fc, fd) \approx g(fa, fc, fd)$}
\RightLabel{$(\approx)$}
\UnaryInfC{$a \approx b \rightarrow g(fa, fc, fd) \approx g(fb, fc, fd)$}
\RightLabel{$(\rightarrow\supset)$}
\UnaryInfC{$\rightarrow a \approx b \supset g(fa, fc, fd) \approx g(fb, fc, fd)$}
\AxiomC{}
\RightLabel{(refl)}
\UnaryInfC{$c \approx d \rightarrow g(fc, fa, fb) \approx g(fc, fa, fb)$}
\RightLabel{$(\approx)$}
\UnaryInfC{$c \approx d \rightarrow g(fc, fa, fb) \approx g(fd, fa, fb)$}
\RightLabel{$(\rightarrow\supset)$}
\UnaryInfC{$\rightarrow c \approx d \supset g(fc, fa, fb) \approx g(fd, fa, fb)$}
\RightLabel{$(\rightarrow \wedge)$}
\BinaryInfC{$\rightarrow (a \approx b \supset g(fa, fc, fd) \approx g(fb, fc, fd)) \wedge (c \approx d \supset g(fc, fa, fb) \approx g(fd, fa, fb))$}
\RightLabel{$(\rightarrow \exists)$}
\UnaryInfC{$\rightarrow \exists v((a \approx b \supset g(fa, fc, v) \approx g(fb, fc, fd)) \wedge (c \approx d \supset g(fc, fa, fb) \approx g(v, fa, fb)))$}
\RightLabel{$(\rightarrow \exists)$}
\UnaryInfC{$\rightarrow \exists uv((a \approx b \supset g(fa, u, v) \approx g(fb, fc, fd)) \wedge (c \approx d \supset g(u, fa, fb) \approx g(v, fa, fb)))$}
\RightLabel{$(\rightarrow \exists)$}
\UnaryInfC{$\rightarrow \exists yuv((a \approx b \supset g(fa, u, v) \approx g(y, fc, fd)) \wedge (c \approx d \supset g(u, fa, y) \approx g(v, fa, fb)))$}
\RightLabel{$(\rightarrow \exists)$}
\UnaryInfC{$\rightarrow \exists xyuv((a \approx b \supset g(x, u, v) \approx g(y, fc, fd)) \wedge (c \approx d \supset g(u, x, y) \approx g(v, fa, fb)))$}
\end{prooftree}
$$

Figure 3: An **LK**$^=$-derivation using sequents

$$
\begin{prooftree}
\AxiomC{}
\RightLabel{(refl)}
\UnaryInfC{$T\ a \approx b, F\ g(fa, fc, fd) \approx g(fa, fc, fd)$}
\RightLabel{$(\approx)$}
\UnaryInfC{$T\ a \approx b, F\ g(fa, fc, fd) \approx g(fb, fc, fd)$}
\RightLabel{(α)}
\UnaryInfC{$F\ a \approx b \supset g(fa, fc, fd) \approx g(fb, fc, fd)$}
\AxiomC{}
\RightLabel{(refl)}
\UnaryInfC{$T\ c \approx d, F\ g(fc, fa, fb) \approx g(fc, fa, fb)$}
\RightLabel{$(\approx)$}
\UnaryInfC{$T\ c \approx d, F\ g(fc, fa, fb) \approx g(fd, fa, fb)$}
\RightLabel{(α)}
\UnaryInfC{$F\ c \approx d \supset g(fc, fa, fb) \approx g(fd, fa, fb)$}
\RightLabel{(β)}
\BinaryInfC{$F\ (a \approx b \supset g(fa, fc, fd) \approx g(fb, fc, fd)) \wedge (c \approx d \supset g(fc, fa, fb) \approx g(fd, fa, fb))$}
\RightLabel{(γ)}
\UnaryInfC{$F\ \exists v((a \approx b \supset g(fa, fc, v) \approx g(fb, fc, fd)) \wedge (c \approx d \supset g(fc, fa, fb) \approx g(v, fa, fb)))$}
\RightLabel{(γ)}
\UnaryInfC{$F\ \exists uv((a \approx b \supset g(fa, u, v) \approx g(fb, fc, fd)) \wedge (c \approx d \supset g(u, fa, fb) \approx g(v, fa, fb)))$}
\RightLabel{(γ)}
\UnaryInfC{$F\ \exists yuv((a \approx b \supset g(fa, u, v) \approx g(y, fc, fd)) \wedge (c \approx d \supset g(u, fa, y) \approx g(v, fa, fb)))$}
\RightLabel{(γ)}
\UnaryInfC{$F\ \exists xyuv((a \approx b \supset g(x, u, v) \approx g(y, fc, fd)) \wedge (c \approx d \supset g(u, x, y) \approx g(v, fa, fb)))$}
\end{prooftree}
$$

Figure 4: An **LK**$^=$-derivation using signed formulas

$$F\ \exists xyuv((a \approx b \supset g(x,u,v) \approx g(y,fc,fd)) \wedge (c \approx d \supset g(u,x,y) \approx g(v,fa,fb)))$$

$$F\ \exists yuv((a \approx b \supset g(fa,u,v) \approx g(y,fc,fd)) \wedge (c \approx d \supset g(u,fa,y) \approx g(v,fa,fb)))$$

$$F\ \exists uv((a \approx b \supset g(fa,u,v) \approx g(fb,fc,fd)) \wedge (c \approx d \supset g(u,fa,fb) \approx g(v,fa,fb)))$$

$$F\ \exists v((a \approx b \supset g(fa,fc,v) \approx g(fb,fc,fd)) \wedge (c \approx d \supset g(fc,fa,fb) \approx g(v,fa,fb)))$$

$$F\ (a \approx b \supset g(fa,fc,fd) \approx g(fb,fc,fd)) \wedge (c \approx d \supset g(fc,fa,fb) \approx g(fd,fa,fb))$$

$$\begin{array}{cc}
F\ a \approx b \supset g(fa,fc,fd) \approx g(fb,fc,fd) & F\ c \approx d \supset g(fc,fa,fb) \approx g(fd,fa,fb) \\
\\
T\ a \approx b & T\ c \approx d \\
F\ g(fa,fc,fd) \approx g(fb,fc,fd) & F\ g(fc,fa,fb) \approx g(fd,fa,fb) \\
\\
F\ g(fa,fc,fd) \approx g(fa,fc,fd) & F\ g(fc,fa,fb) \approx g(fc,fa,fb)
\end{array}$$

Figure 5: An $\mathbf{LK}^{=}$-derivation using tableaux

$$\frac{}{\Gamma, A, \neg A}\ (Ax) \qquad \frac{}{\Gamma, t \approx t}\ (\mathit{refl}) \qquad \frac{\Gamma[x \mapsto t], s \not\approx t}{\Gamma[x \mapsto s], s \not\approx t}\ (\approx)$$

$$\frac{\Gamma, \varphi, \psi}{\Gamma, \varphi \vee \psi}\ (\vee) \qquad \frac{\Gamma, \varphi \quad \Gamma, \psi}{\Gamma, \varphi \wedge \psi}\ (\wedge)$$

$$\frac{\Gamma, \varphi[x \mapsto t], \exists x \varphi}{\Gamma, \exists x \varphi}\ (\exists) \qquad \frac{\Gamma, \varphi[x \mapsto y]}{\Gamma, \forall x \varphi}\ (\forall)$$

In the rule (Ax), A is an atomic formula. The rule $(\forall)$ satisfies the eigenvariable condition.

Figure 6: Calculus $\mathbf{LK}^{=}$ for formulas in negation normal form

2 Translation of logic with equality into logic without equality

Instead of inventing methods of equality handling in sequent calculi, one can use "efficient" transformations of logic with equality into logic without equality. By "efficient" we mean that the transformations avoid adding equality axioms. There are two such methods, the first is proposed by Brand [33] and known as the modification method, or *Brand's transformation* and the second by Moser and Steinbach [119] and independently in a stronger form by Bachmair, Ganzinger and Voronkov [14], under the name of *constraint equality elimination*. Since these transformations are used to implement equality in several tableau-based theorem provers (Schumann [139], Bibel et.al. [30]), we briefly consider them in this section.

Both transformations are applied to a set of clauses. To apply them to a set of formulas, one should first transfer them to the clause form. Both transformations apply to the set of clauses a sequence of transformation steps, each step eliminating the need in some equality axioms. Throughout this section we assume that we deal with a set of clauses whose only predicate symbol is equality.

2.1 Modification method

The transformation used by the *modification method* comprises three parts: flattening, symmetry elimination and transitivity elimination.

Flattening. This transformation eliminates the need in function and predicate substitution axioms.

- ▶ A clause is *flat* if all occurrences of non-variables terms in it are arguments to the equality predicate.
- ▶ An *flat form of a clause* C is any clause that is flat and is equivalent to C.

For example, the clause $g(a, b) \approx b$ is not flat. Two possible flat forms of this clause are $b \not\approx x \vee g(a, x) \approx x$ and $b \not\approx x \vee b \not\approx y \vee g(a, x) \approx y$.

Every clause can be translated to an equivalent flat clause by the flattening transformation:

$$C[f(t)] \quad \Rightarrow \quad t \not\approx y \vee C[f(y)],$$

where y is a new variable. The essence of flattening is expressed by the following theorem of Brand [33].

Theorem 2.1 A set of flat clauses is satisfiable if and only if it has a model in which $\approx$ is interpreted as an equivalence relation.

In this section, when we speak about unsatisfiability of a set of clauses, we shall treat the equality predicate differently. One possibility is to define satisfiability as the existence of a model satisfying all clauses *and the equality axioms*. In this case we shall speak about *equational satisfiability*. Another possibility is to treat the equality predicate as satisfying either some equality axioms or none at all. Flattening eliminates the need in function and predicate substitution axioms, thus leaving us only with reflexivity, symmetry and transitivity. Thus, the above theorem says that a set of clauses is equationally unsatisfiable if and only if it has no model in which $\approx$ is an equivalence relation.

Thus, instead of adding all equality axioms to the set of flattened clauses we have to only add the axioms expressing that $\approx$ is an equivalence relation:

$$\begin{gathered} x \approx x, \\ x \not\approx y \vee y \approx x, \\ x \not\approx y \vee y \not\approx z \vee x \approx z. \end{gathered}$$

The next two steps of the transformation allow us to get rid of symmetry and transitivity.

Symmetry elimination.

- ▶ We call a *symmetric version* of C any clause obtained from C by replacing one or more positive equations $s \approx t$ by $t \approx s$.
- ▶ The *symmetry elimination* replaces every clause by all symmetric versions of this clause.

For example, one possible symmetric version of the clause $a \approx b \vee b \approx c \vee x \not\approx f(x)$ is $a \approx b \vee c \approx b \vee x \not\approx f(x)$. The clause $a \approx b \vee b \approx c \vee f(x) \not\approx x$ is *not* a symmetric version of this clause, since only the sides of positive equations can be swapped. Evidently, symmetry elimination produces 2^n clauses from a clause with n positive equations (unless the clause contains a literal $t \approx t$, in which case it is redundant).

One can prove that after symmetry elimination only transitivity and reflexivity axioms should be taken care of:

Theorem 2.2 Let N be a set of flat clauses and N' be obtained from N by symmetry elimination. Then N is equationally unsatisfiable if and only if N' has no model in which $\approx$ is interpreted as a reflexive and transitive relation.

Transitivity elimination. *Transitivity elimination* replaces every positive equation $s \approx t$ by the disjunction $t \not\approx y \vee s \approx y$ with a new auxiliary variable y.

For example, transitivity elimination replaces the clause

$$f(x) \approx a \vee b \approx y \vee x \not\approx y$$

by the clause

$$a \not\approx z \vee f(x) \approx z \vee y \not\approx u \vee b \approx u \vee x \not\approx y.$$

After transitivity elimination the only axiom that remains is reflexivity:

Theorem 2.3 Let N be a set of flat clauses and N' be obtained from N by symmetry and transitivity elimination. Then N is equationally unsatisfiable if and only if $N' \cup \{x \approx x\}$ is unsatisfiable.

Consider an example. Suppose that the modification method is applied to the following set of two clauses:

$$b \approx c \tag{4}$$

$$f(c, x) \not\approx x \tag{5}$$

Clause (4) is flat. Symmetry and transitivity elimination applied to this clause yield clauses (6) and (7) below. Clause (5) is not flat. Flattening yields clause (8) below. Symmetry and transitivity elimination do not change the clause. Thus, the modification method gives the following four clauses.

$$b \not\approx y \vee c \approx y \tag{6}$$
$$c \not\approx y \vee b \approx y \tag{7}$$
$$c \not\approx y \vee f(y,x) \not\approx x \tag{8}$$
$$x \approx x \tag{9}$$

Brand's transformation and basic paramodulation. It was noted by several researchers that there exist close connections between the modification method and basic paramodulation. Basic paramodulation inferences on the set of original clauses can be simulated by resolution on the set of transformed clauses. Consider, for example, the following paramodulation inference from clauses (4) and (5):

$$\frac{b \approx c \quad f(c,x) \not\approx x}{f(b,x) \not\approx x}\,.$$

If we transform, using the modification method, the conclusion of this inference, we obtain the clause $b \not\approx y \vee f(y,x) \not\approx x$. The same clause can be obtained by the resolution inference from modified clauses (6) and (8)

$$\frac{b \not\approx y \vee c \approx y \quad c \not\approx y \vee f(y,x) \not\approx x}{b \not\approx y \vee f(y,x) \not\approx x}\,.$$

This technique of simulating derivations in the paramodulation-based refutation system by resolution-based derivations on the transformed set of clauses can be used to prove completeness of the modification method. Brand used pure model-theoretic technique to prove completeness.

One can also note that some resolution inferences on the modified set of clauses correspond to redundant inferences on the original set of clauses. For example, paramodulation into a variable is known to be redundant. The picture below shows such a redundant inference and the corresponding resolution inference:

$$\frac{a \approx x \quad b \approx c}{a \approx b} \qquad \text{and} \qquad \frac{c \approx y \vee b \not\approx y \quad x \not\approx y \vee a \approx y}{b \not\approx y \vee a \approx y}\,.$$

Some resolution inferences correspond to unordered applications of paramodulation. For example, if we consider any reduction ordering in which $b \succ c$, then no ordered paramodulation is possible from clauses (4) and (5), but we have seen that the modified set of clauses allows for some resolution inferences. For quite some time people who worked on theorem proving using tableau and related methods tried to optimize the modification method so that less resolution inferences on the transformed set will correspond to redundant superposition inferences.

2.2 Constrained equality elimination

Such an optimized translation has been proposed by Bachmair, Ganzinger and Voronkov [14] under the name of *constraint equality elimination*, or simply, *CEE-transformation*. This transformation comprises two of three parts. *Flattening* and *symmetry elimination* are the same as in Brand's transformation. The third part (transitivity elimination) is different from Brand's transformation in several aspects. To precise CEE-transformation we need some additional notions.

Constrained clauses.

- By an *ordering constraint*, or simply *constraint* we mean a conjunction of expressions which can be of two kinds:

  - *equality constraint* $s = t$, or
  - *inequality constraint* $s \succ t$ or $s \succeq t$,

  where s, t are terms.

- A substitution θ is called a *solution* to an equality constraint $s = t$ (respectively, inequality constraint $s \succ t$ or $s \succeq t$) if θ is grounding for $vars(s) \cup vars(t)$ and $s\theta = t\theta$ (respectively, $s\theta \succ t\theta$ or $s\theta \succeq t\theta$), where $\succ$ is interpreted as a reduction ordering. A substitution θ is a solution to a conjunction $\mathcal{C}$ of constraints if θ is a solution to every equality or inequality constraint in $\mathcal{C}$.

- A constraint $\mathcal{C}$ is *satisfiable* if it has a solution.

- Constraints $\mathcal{C}_1$ and $\mathcal{C}_2$ are called *equivalent* if they have the same sets of solutions.

Note that we use a different equality symbol in equality constraints, in order to emphasize that we mean the syntactic equality.

There are efficient methods for solving ordering constraints for lexicographic path orderings described e.g. in Nieuwenhuis [120] and Nieuwenhuis and Rubio [123].

- ▶ We call a *constrained clause* a pair consisting of a clause C and a constraint $\mathcal{C}$. Such a constrained clause is be denoted by $C \cdot \mathcal{C}$. We identify a constrained clause $C \cdot \top$, where $\top$ is the empty conjunction, with the unconstrained clause C.

- ▶ A *ground instance* of a constrained clause $C \cdot \mathcal{C}$ is any ground clause $C\theta$ such that the substitution θ is a solution to the constraint $\mathcal{C}$.

- ▶ We call two constrained clauses $C_1 \cdot \mathcal{C}_1$ and $C_2 \cdot \mathcal{C}_2$ *equivalent*, if they have the same ground instances. A set N of constrained clauses is *satisfiable* if the set of all its ground instances is satisfiable.

Transitivity elimination. One of the ideas of transitivity elimination in constraint equality elimination is to introduce so-called *link constraints* into transitivity elimination. For example, the equation $s \approx t$ is transformed into $(t \not\approx x \vee s \approx x) \cdot (t \succeq x \wedge s \succ x)$. The names comes from the intuitive idea that the variable x is the *link variable* used to simulate transitivity.

The rationale behind introducing link constraints is that the sequence of equational replacements

$$s \approx s_0 \approx s_1 \approx \ldots \approx s_n \approx t$$

(using equations $s_i \approx s_{i+1}$) can be simulated by a sequence of resolution inferences using $s_i \approx x_i \leftrightarrow s_{i+1} \approx x_i$, plus a final resolution step with the reflexivity axiom $x \approx x$ that instantiates the link variables. The ordering constraints

1. ensure that the variables x_i can only be *instantiated by minimal terms* among the s_i;

2. *block the search for alternative equational proofs* that apply the same equations but differ in the instantiation of the link variables.

- Formally, *transitivity elimination* is the following transformation:

$$\begin{array}{rcl} s \approx t & \Rightarrow & (t \not\approx x \vee s \approx x) \cdot (t \succeq x \wedge s \succ x) \\ s \not\approx t & \Rightarrow & (t \not\approx x \vee s \not\approx x) \cdot (t \succeq x \wedge s \succeq x) \\ s \approx x & \Rightarrow & (s \approx x) \cdot (s \succ x) \\ s \not\approx x & \Rightarrow & (s \not\approx x) \cdot (s \succeq x) \end{array}$$

  where t is not a variable. This transformation replaces every literal in the clause by the corresponding literals shown after $\Rightarrow$, and adds to the clause the corresponding constraints.

  - The variables x introduced by the transformation are called the *link variables*.
  - Each constraint introduced by the transformation (for example $s \succeq x$ is called the *link constraint*.

Consider again clauses (4) and (5). We have already considered how Brand's transformation behaves on these clauses. Now we apply CEE-transformation to these clauses, using any ordering in which c is the least element. With such ordering, no ordered paramodulation is possible. CEE-transformation yields four clauses:

$$\begin{array}{l} (b \not\approx y \vee c \approx y) \cdot (b \succeq y \wedge c \succ y) \\ (c \not\approx y \vee b \approx y) \cdot (c \succeq y \wedge b \succ y) \\ (c \not\approx y \vee f(y,x) \not\approx x) \cdot (c \succeq y \wedge f(y,x) \succeq x) \\ x \approx x \end{array}$$

Note that the constraint $c \succ y$ is unsatisfiable and the constraint $c \succeq y$ can only be satisfied when $c = y$. The constraint $f(y,x) \succeq x$ is valid. Thus, we can simplify the set to

$$\begin{array}{l} c \not\approx c \vee b \approx c \\ c \not\approx c \vee f(c,x) \not\approx x \\ x \approx x \end{array}$$

Since the set contains $x \approx x$, both occurrences of $c \not\approx c$ can be removed, yielding

$$\begin{array}{l} b \approx c \\ f(c,x) \not\approx x \\ x \approx x \end{array}$$

No derivation by either resolution or model elimination is possible! This example shows the power of CEE-transformation compared to Brand's transformation.

The main property of CEE-transformation is preservation of satisfiability proved in [14]:

Theorem 2.4 A set N of unconstrained equational clauses is equationally unsatisfiable if and only if the transformed set is unsatisfiable.

CEE-transformation can be used by any non-equational prover by adding constraints. The transformation without constraints is also sound and complete. The *least constant optimization* that removed the clause containing $c \succ y$ and simplified the clauses containing $c \succeq y$ in the example, can be also be used by provers that do not handle constraints. In fact, this transformation without constraints was described by Moser and Steinbach [119] and used in some versions of the theorem prover SETHEO (Ibens and Letz [89]).

Compared to Brand's transformation, CEE-transformation has the following advantages:

1. link constraints are added;

2. positive equations $t \approx x$ are not split into $x \not\approx y \vee t \approx y$.

The disadvantage is that negative equations $t \not\approx s$ are split giving longer clauses. Preliminary experiments with CEE-transformation [14] with the Protein prover (Baumgartner and Furbach [16]) on certain simple problems in group theory have shown that constraints may eliminate 99% percent of model elimination derivations even for comparatively simple examples.

Following Bachmair, Ganzinger and Voronkov [14], we give examples that show Brand's transformation *cannot be improved* by either link constraints or non-splitting of positive literals $t \approx x$.

1. Suppose that Brand's transitivity elimination is equipped by link constraints. Consider an equationally unsatisfiable set of unit clauses $a \approx b$, $a \approx c$ and $b \not\approx c$. We obtain the following set of constrained clauses

$$
\begin{array}{l}
(b \not\approx x \vee a \approx x) \cdot (b \succeq x \wedge a \succ x) \\
(a \not\approx x \vee b \approx x) \cdot (a \succeq x \wedge b \succ x) \\
(c \not\approx x \vee a \approx x) \cdot (c \succeq x \wedge a \succ x) \\
(a \not\approx x \vee c \approx x) \cdot (a \succeq x \wedge c \succ x) \\
b \not\approx c \\
x \approx x
\end{array}
$$

We show that this of set of constraint clauses is satisfiable given an ordering in which $c \succ b \succ a$. Indeed, the first and third clause contain the unsatisfiable constraint $a \succ x$ and hence can be removed. The remaining clauses are satisfiable even without the constraints. In short, ordering constraints are not compatible with Brand's original transformations.

2. The set of three unit clauses $f(x) \approx x$, $g(x) \approx x$ and $f(x) \not\approx g(x)$ is equationally unsatisfiable. Suppose that we use Brand's transformation, but do not split positive equations $t \approx x$. Then we obtain a satisfiable set of clauses

$$
\begin{array}{ll}
f(x) \approx x & f(x) \not\approx y \vee x \approx y \\
g(x) \approx x & g(x) \not\approx y \vee x \approx y \\
f(x) \not\approx g(x) & x \approx x
\end{array}
$$

In short, non-splitting of positive equations is not compatible with Brand's original transformations. Equally, we can say that constraint equality elimination cannot be improved by non-splitting negative literals $s \not\approx t$, even if we drop constraints.

Although the modification method has been proposed in the resolution framework, it has not been widespread used. The principal influence of this method on the equality reasoning was the proof of completeness of paramodulation without functional reflexivity axioms (conjectured by Robinson and Wos [134]). The main tool for handling equality in resolution-based systems is paramodulation augmented with ordering restrictions and redundancy deletion. At the same time, most existing implementations of sequent-based methods with equality are based on the Brand's translation because the known ways of handling equality in resolution has not been generalized to sequent-based methods until very recently.

The distinction between CEE-transformation and Brand's transformation reflects the progress in equality handling made for more than twenty years (from 1975).

3 Free variable systems

The system $\mathbf{LK}^=$, while having some nice proof-theoretical properties, is not very suitable for automatic proof-search. The main problem lies in the γ-rules, for example

$$\frac{\Gamma, F\ \varphi[x \mapsto t], F\ \exists x \varphi}{\Gamma, F\ \exists x \varphi}\ (\gamma)$$

where t can be *any term.* Application of such rules from conclusions to premises create too high nondeterminism. To make $\mathbf{LK}^=$ suitable for automatic proof-search, we have to introduce its *free variable* version, where instead of t we substitute a variable, which during the proof-search can be instantiated to some particular term t. In this section we consider two possible ways of making $\mathbf{LK}^=$ a free variable system. One gives rise to the semantic tableaux and the other to the inverse method. We also briefly consider the method of matings.

3.1 Free variable tableaux

For the formalization of free variable tableaux it is not enough to give sequent-style inference rules, like for $\mathbf{LK}^=$. Free variables are global for the whole tableau that represents a derivation. Thus, we need to introduce a calculus dealing with derivations. There are several ways of defining such calculi. One way is to define a calculus on trees, like in Fitting [64]. Another possibility is to only deal with the multiset of branches of such trees, which gives a more succinct formalization of inference rules (e.g. Moser, Lynch and Steinbach [118], Plaisted [129] or Degtyarev and Voronkov [54]). In this section we shall present such a formalization for free variable tableaux. *Since we restrict ourselves to skolemized formulas, we do not need to consider δ-rules.*

- ▶ A *branch* is any finite multiset of signed formulas.
- ▶ A *tableau* is any finite multiset $\{\Gamma_1, \ldots, \Gamma_n\}$ of branches, denoted by $\Gamma_1 \mid \ldots \mid \Gamma_n$.
- ▶ The tableau with $n = 0$ is called the *empty tableau* and denoted by $\#$.

If Γ is a branch and $X\ \varphi$ is a signed formula, where $X \in \{T, F\}$, then $\Gamma, X\ \varphi$ denotes the branch $\Gamma \cup \{X\ \varphi\}$, and similar for $X\ \varphi, \Gamma$.

Let Γ be a branch in a tableau.

- ▶ The *multiset of literals on* Γ, denoted $lit(\Gamma)$, is defined by

$$lit(\Gamma) \rightleftharpoons \{A \mid A \text{ is an atom and } T\,A \in \Gamma\} \cup \{\neg A \mid A \text{ is an atom and } F\,A \in \Gamma\}.$$

- ▶ A branch Γ is called *closed* if $lit(\Gamma)$ is inconsistent.

One can use any standard definition of inconsistency for first-order classical logic. In terms of the calculus $\mathbf{LK}^=$ the inconsistency means that the sequent $lit(\Gamma) \rightarrow$ is derivable in $\mathbf{LK}^=$. It is obvious that in a derivation of such a sequent one can only use the rules $(\neg \rightarrow)$, (Ax), (refl) and $(\approx)$. For logic without equality, the inconsistency is equivalent to the condition that $lit(\Gamma)$ contains a pair of complementary literals A and $\neg A$. For logic with equality, a criterion for inconsistency is given in Theorem 3.4 below.

- ▶ The *tableau calculus* **TK** is given in Figure 7 (compare it with $\mathbf{LK}^=$ in the form using signed formulas);
- ▶ α-, β-, γ-, and $\neg$-rules are called the *tableau expansion rules*;
- ▶ the rule (abc) is called the *atomic branch closure rule*.

Compairing the calculus **TK** introduced above with the calculus $\mathbf{LK}^=$ for signed formulas, we note the following. First, **TK** operates multisets of multisets of formulas (trees) and not multisets of formulas (branches) as $\mathbf{LK}^=$. So derivations in **TK** are always linear and not tree-like as in $\mathbf{LK}^=$. Second, applications of **TK**-inference rules correspond to counterapplications (from the conclusion to the premises) of $\mathbf{LK}^=$-inference rules.

The following version of Herbrand theorem forms the theoretical basis for the tableau method:

Theorem 3.1 Let φ be a closed formula. It is provable in classical logic if and only if there is a tableau $\mathcal{T}$ obtained from $F\,\varphi$ by applications of tableau expansion rules and a substitution θ such that any branch in $\mathcal{T}\theta$ is closed.

Theorem 3.1 suggests the following way of proving φ: starting with $F\,\varphi$, we construct a tableau by some applications of the tableau expansion rules. Then, we try to find the substitution θ that makes all branches in the tableau closed. In the case without equality, this theorem and standard lifting yield the following theorem:

$$\frac{\Gamma_1, T\,\varphi \wedge \psi \mid \ldots \mid \Gamma_n}{\Gamma_1, T\,\varphi, T\,\psi \mid \ldots \mid \Gamma_n}\ (\alpha) \qquad \frac{\Gamma_1, F\,\varphi \wedge \psi \mid \ldots \mid \Gamma_n}{\Gamma_1, F\,\varphi \mid \Gamma_1, F\,\psi \mid \ldots \mid \Gamma_n}\ (\beta)$$

$$\frac{\Gamma_1, T\,\varphi \vee \psi \mid \ldots \mid \Gamma_n}{\Gamma_1, T\,\varphi \mid \Gamma_1, T\,\psi \mid \ldots \mid \Gamma_n}\ (\beta) \qquad \frac{\Gamma_1, F\,\varphi \vee \psi \mid \ldots \mid \Gamma_n}{\Gamma_1, F\,\varphi, F\,\psi \mid \ldots \mid \Gamma_n}\ (\alpha)$$

$$\frac{\Gamma_1, T\,\varphi \supset \psi \mid \ldots \mid \Gamma_n}{\Gamma_1, F\,\varphi \mid \Gamma_1, T\,\psi \mid \ldots \mid \Gamma_n}\ (\beta) \qquad \frac{\Gamma_1, F\,\varphi \supset \psi \mid \ldots \mid \Gamma_n}{\Gamma_1, T\,\varphi, F\,\psi \mid \ldots \mid \Gamma_n}\ (\alpha)$$

$$\frac{\Gamma_1, T\,\neg\varphi \mid \ldots \mid \Gamma_n}{\Gamma_1, F\,\varphi \mid \ldots \mid \Gamma_n}\ (\neg) \qquad \frac{\Gamma_1, F\,\neg\varphi \mid \ldots \mid \Gamma_n}{\Gamma_1, T\,\varphi \mid \ldots \mid \Gamma_n}\ (\neg)$$

$$\frac{T\,\forall x\varphi \mid \ldots \mid \Gamma_n}{\Gamma_1, T\,\varphi[x \mapsto z], \Gamma_1, T\,\forall x\varphi, \mid \ldots \mid \Gamma_n}\ (\gamma)$$

$$\frac{F\,\exists x\varphi \mid \ldots \mid \Gamma_n}{\Gamma_1, F\,\varphi[x \mapsto z], \Gamma_1, F\,\exists x\varphi, \mid \ldots \mid \Gamma_n}\ (\gamma)$$

$$\frac{\Gamma_1, T\,A, F\,B \mid \Gamma_2 \mid \ldots \mid \Gamma_n}{(\Gamma_2 \mid \ldots \mid \Gamma_n)\mathrm{mgu}(A, B)}\ (abc)$$

In both rules (γ) the variable z does not occur in the premise. In the rule (abc) the formulas A and B are atomic formulas whose predicate symbol is different from $\approx$.

Figure 7: Calculus **TK**,

Theorem 3.2 Let ξ be a closed formula without equality. It is provable in classical logic if and only if there is a derivation in **TK** of the empty tableau # from the tableau $F\,\xi$.

For logic with equality the situation is more complicated because there is no simple criterion of inconsistency of a branch.

Theorem 3.1 suggests the following notion (coming back to Chang and Lee [95]).

- ► A branch Γ is called *substitutively inconsistent* if there is a substitution σ such that $lit(\Gamma)\sigma$ is inconsistent.

- ▶ A tableau $\Gamma_1 \mid \dots \mid \Gamma_n$ is called *substitutively inconsistent* if there is a substitution σ such that for every $i \in \{1, \dots, n\}$ the multiset $lit(\Gamma_i)\sigma$ is inconsistent.

In order to illustrate Theorem 3.1 for logic with equality, we come back to Example 1.3.

Now, we represent derivation in **TK** as trees with signed formulas (i.e. as semantic tableau in the common meaning).

Example 3.3 Consider again the formula of Example 1.3. After some application of tableau expansion rules we obtain the free variable tableau shown in Figure 8. This tableau has two branches. The multisets of literals on the two branches are, respectively

$$\begin{array}{l} \{a \approx b, g(x,u,v) \not\approx g(y,fc,fd)\} \quad \text{and} \\ \{c \approx d, g(u,x,y) \not\approx g(v,fa,fb)\}. \end{array} \tag{10}$$

The tableau is substitutively inconsistent: the substitution $[x \mapsto fa, y \mapsto fb, u \mapsto fc, v \mapsto fd]$ makes both branches closed.

To demonstrate that a branch is closed, we have to be able to check a multiset of literals $\mathcal{L}$ for inconsistency. To give a characterization of inconsistency for first-order logic with equality, we introduce some definitions. Let E be a multiset of equations and L_1, L_2 be terms or literals.

- ▶ We write $L_1 \leftrightarrow_E L_2$ if there exists an equation $(s \approx t) \in E$ such that L_1 is $L[s]$ and L_2 is $L[t]$. We denote by $\leftrightarrow_E^*$ the reflexive and transitive closure of $\leftrightarrow_E$.

The following statement characterizes inconsistency of a multiset of literals:

Theorem 3.4 Let $\mathcal{L}$ be a multiset of literals and $\mathcal{E}(\mathcal{L})$ be the multiset of equations in $\mathcal{L}$. Then the following conditions are equivalent:

1. $\mathcal{L}$ is inconsistent;
2. either there is a literal $(s \not\approx t) \in \mathcal{L}$ such that $s \leftrightarrow_{\mathcal{E}(\mathcal{L})}^* t$, or there are non-equality literals $A, \neg B \in \mathcal{L}$ such that $A \leftrightarrow_{\mathcal{E}(\mathcal{L})}^* B$.

$$F\ \exists xyuv((a \approx b \supset g(x,u,v) \approx g(y,fc,fd)) \wedge (c \approx d \supset g(u,x,y) \approx g(v,fa,fb)))$$

$$F\ \exists yuv((a \approx b \supset g(x,u,v) \approx g(y,fc,fd)) \wedge (c \approx d \supset g(u,x,y) \approx g(v,fa,fb)))$$

$$F\ \exists uv((a \approx b \supset g(x,u,v) \approx g(y,fc,fd)) \wedge (c \approx d \supset g(u,x,y) \approx g(v,fa,fb)))$$

$$F\ \exists v((a \approx b \supset g(x,u,v) \approx g(y,fc,fd)) \wedge (c \approx d \supset g(u,x,y) \approx g(v,fa,fb)))$$

$$F\ (a \approx b \supset g(x,u,v) \approx g(y,fc,fd)) \wedge (c \approx d \supset g(u,x,y) \approx g(v,fa,fb))$$

$$F\ a \approx b \supset g(x,u,v) \approx g(y,fc,fd)) \qquad\qquad F\ c \approx d \supset g(u,x,y) \approx g(v,fa,fb)$$

$$\begin{array}{c} T\ a \approx b \\ F\ g(x,u,v) \approx g(y,fc,fd)) \end{array} \qquad\qquad \begin{array}{c} T\ c \approx d \\ F\ g(u,x,y) \approx g(v,fa,fb) \end{array}$$

Figure 8: A free variable tableau derivation

The inconsistency can be checked using any congruence closure algorithm (e.g. Shostak [142]). However, the real question is how to close tableau branches for a given free variable tableau. This questions leads to simultaneous rigid E-unification considered in Section 5.

3.2 Matings

In this section we briefly discuss the method of matings which can be considered as another formalization of proof-search in a free-variable sequent calculus.

Let us come back to the formula of Example 1.3. The tableau constructed in Example 3.3 can be represented as a matrix in the method of matings as shown in Figure 9. For a better illustration, we also show the construction of the matrix. After translation to the negation normal form, the negation of the input formula becomes $\forall xyuv((a \approx b \wedge g(x,u,v) \not\approx g(y,fc,fd)) \vee (c \approx d \wedge g(u,x,y) \not\approx g(v,fa,fb)))$. From this formula, we can perform the sequence of steps shown in Figure 9. The matrix in the figure has two vertical paths which are precisely the multisets in (10).

In general, for every matrix M consisting only of literals we can construct a tableau $\mathcal{T}$ such that the multiset of vertical paths in M is the multiset of multisets of literals on the branches of $\mathcal{T}$. Methods and algorithms described in this paper work on tableaux represented as multisets of branches. To modify them for the connection format, one has to imagine matrices as multisets of branches[3].

A detailed explanation of equational matings can be found in Gallier, Raatz, and Snyder [73], Gallier et.al. [70] or Gallier [68].

3.3 The inverse method

Derivations in the free variable tableau system correspond to the *backward* (i.e. from conclusions to premises) proof-search in $\mathbf{LK}^=$. There is an orthogonal method of *forward* proof-search in $\mathbf{LK}^=$ which gives rise to the inverse method. Forward proof-search starts with axioms and tries to derive the goal formula. The whole idea of the forward direction of proof-search seems at the first sight strange. But in the case of sequent calculi, the possibility

[3]Some rules formulated below, for example tableau paramodulation or superposition, have a convenient presentation on tableaux but have no straightforward analogue for matrices.

of such proof-search direction is based on some proof-theoretic properties of $\mathbf{LK}^{=}$, most notably

- the *subformula property*: for every derivation Π of a formula ξ, all sequents occurring in Π consist of formulas of the form $\varphi\theta$, where φ is a subformula of ξ.

This property together with the use of free variables gives rise to a proof-search method originally described by Maslov [106, 108, 110]. Since the inverse method is much less known than the tableau method, we describe in this section a version of the inverse method for logic without equality.

Our formalization of the inverse method will mainly follow Voronkov [155]. Some other recent formalizations may be found in Lifschitz [98], Tammet [147]. In this section, we only formalize the inverse method for logic without equality. The inverse method for logic with equality is considered in Section 8.3. Given a closed formula ξ as the goal, the inverse method constructs a logical system $\mathbf{IK}_\xi$ which is a specialization of $\mathbf{LK}^{=}$ intended for proving ξ. This calculus works on sequents that consist of formulas $\varphi\theta$ such that φ is a subformula of ξ.

For simplicity, we assume that ξ is in the skolemized negation normal form and give a calculus consisting of the rules $(\vee)$, $(\wedge)$ and $(\exists)$, like the calculus of Figure 6. Before defining $\mathbf{IK}_\xi$, we define some operations related to substitutions.

Let s, t be terms.

- A *weak most general unifier* of s, t denoted $\mathrm{wmgu}(s, t)$ is a pair of substitutions $(\rho\sigma, \sigma)$ such that ρ is a renaming substitution, $s\rho$ and t have no common variables, and σ is a most general unifier of $s\rho$ and t.

- A *most general unifier* of substitutions σ_1, σ_2, denoted $\mathrm{mgu}(\sigma_1, \sigma_2)$, is a substitution θ defined as follows. Let $\{x_1, \ldots, x_n\} = dom(\sigma_1) \cup dom(\sigma_2)$. Then θ is any simultaneous most general unifier of tuples $\langle x_1\sigma_1, \ldots, x_n\sigma_1\rangle$ and $\langle x_1\sigma_2, \ldots, x_n\sigma_2\rangle$.

- We denote by σ_{-x} the substitution defined as follows. For every variable y,

$$y\sigma_{-x} \rightleftharpoons \begin{cases} x, & \text{if } x = y; \\ y\sigma, & \text{otherwise} \end{cases}$$

Using these notions, we define the calculus $\mathbf{IK}_\xi$ formalizing the inverse method in the following way.

- *Sequents in the calculus* $\mathbf{IK}_\xi$ consist of formulas $\varphi\theta$ such that φ is a subformula of ξ and all variables in $dom(\theta)$ are free variables of φ.

- The *calculus* $\mathbf{IK}_\xi$ is given in Figure 10.

Consider an example on an inverse method proof of the formula

$$\exists x \exists y \forall z (P(x) \vee P(y) \supset P(z)).$$

After skolemization and translation into the negation normal form we obtain the formula $\xi = \exists x \exists y ((\neg P(x) \wedge \neg P(y)) \vee P(f(x, y)))$. An $\mathbf{IK}_\xi$-derivation of ξ is given in Figure 11.

As related to proof-search, there are several terms to distinguish the tableau method from the inverse method:

Tableau method	Inverse method	
bottom-up	top-down	(proof-theoretic terminology)
top-down	bottom-up	(search-related terminology)
backward	forward	
direct	indirect	
goal-oriented	inverse	
analytic	non-analytic	
nonlocal	local	

The inverse method for classical logic can be simulated by hyperresolution using a structure-preserving clause-form translation that has been described in various forms in Maslov [109], Eder [61], Plaisted and Greenbaum [130] and Boy de la Tour [32]. We consider an example in Section 8.3.

$$\begin{array}{lr}\forall xyuv((a \approx b \wedge g(x,u,v) \not\approx g(y,fc,fd)) \vee & \\ \quad (c \approx d \wedge g(u,x,y) \not\approx g(v,fa,fb))) & \Rightarrow\end{array}$$

$$\begin{array}{lr}\forall yuv((a \approx b \wedge g(x,u,v) \not\approx g(y,fc,fd)) \vee & \\ \quad (c \approx d \wedge g(u,x,y) \not\approx g(v,fa,fb))) & \Rightarrow\end{array}$$

$$\begin{array}{lr}\forall uv((a \approx b \wedge g(x,u,v) \not\approx g(y,fc,fd)) \vee & \\ \quad (c \approx d \wedge g(u,x,y) \not\approx g(v,fa,fb))) & \Rightarrow\end{array}$$

$$\begin{array}{lr}\forall v((a \approx b \wedge g(x,u,v) \not\approx g(y,fc,fd)) \vee & \\ \quad (c \approx d \wedge g(u,x,y) \not\approx g(v,fa,fb))) & \Rightarrow\end{array}$$

$$\begin{array}{lr}(a \approx b \wedge g(x,u,v) \not\approx g(y,fc,fd)) \vee & \\ (c \approx d \wedge g(u,x,y) \not\approx g(v,fa,fb)) & \Rightarrow\end{array}$$

$$\left[\ a \approx b \wedge g(x,u,v) \not\approx g(y,fc,fd) \quad c \approx d \wedge g(u,x,y) \not\approx g(v,fa,fb)\ \right] \quad \Rightarrow$$

$$\left[\ \left[\begin{array}{c} a \approx b \\ g(x,u,v) \not\approx g(y,fc,fd) \end{array}\right] \quad c \approx d \wedge g(u,x,y) \not\approx g(v,fa,fb)\ \right] \quad \Rightarrow$$

$$\left[\ \left[\begin{array}{c} a \approx b \\ g(x,u,v) \not\approx g(y,fc,fd) \end{array}\right] \quad \left[\begin{array}{c} c \approx d \\ g(u,x,y) \not\approx g(v,fa,fb) \end{array}\right]\ \right]$$

Figure 9: Matings

$$\frac{}{A\sigma_1, \neg B\sigma_2}\ (Ax) \qquad \frac{\Gamma, \varphi\sigma}{\Gamma, (\varphi \vee \psi)\sigma}\ (\vee) \qquad \frac{\Gamma, \psi\sigma}{\Gamma, (\varphi \vee \psi)\sigma}\ (\vee)$$

$$\frac{\Gamma, \varphi\sigma_1 \quad \Delta, \psi\sigma_2}{(\Gamma, \Delta, \varphi\sigma_1 \wedge \psi\sigma_2)\mathrm{mgu}(\sigma_1, \sigma_2)}\ (\wedge) \qquad \frac{\Gamma, \varphi\sigma_1, \varphi\sigma_2}{(\Gamma, \varphi\sigma_1)\mathrm{mgu}(\sigma_1, \sigma_2)}\ (fac) \qquad \frac{\Gamma, \varphi\sigma}{\Gamma, (\exists x\varphi)\sigma_{-x}}\ (\exists)$$

In the rule (Ax), A and $\neg B$ are subformulas of ξ and $(\sigma_1, \sigma_2) = \mathrm{wmgu} A, B$. In the other rules, the formulas $\varphi \wedge \psi$, $\varphi \vee \psi$ and $\exists x\varphi$ are subformulas of ξ.

Figure 10: Calculus $\mathbf{IK}_\xi$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\neg P(x)[x \mapsto f(x,y)], P(f(x,y))}\ (Ax) \quad \dfrac{}{\neg P(y)[y \mapsto f(x,y)], P(f(x,y))}\ (Ax)}{(\neg P(x)[x \mapsto f(x,y)] \wedge \neg P(y)[y \mapsto f(x,y)], P(f(x,y)), P(f(x,y)))\varepsilon}\ (\wedge)}{((\neg P(x) \wedge \neg P(y)) \vee P(f(x,y)))[x \mapsto f(x,y), y \mapsto f(x,y)], P(f(x,y)), P(f(x,y))}\ (\vee)}{\exists x \exists y((\neg P(x) \wedge \neg P(y)) \vee P(f(x,y))), P(f(x,y)), P(f(x,y))}\ (\exists)}{\exists x \exists y((\neg P(x) \wedge \neg P(y)) \vee P(f(x,y))), P(f(x,y))}\ (fac)}{\exists x \exists y((\neg P(x) \wedge \neg P(y)) \vee P(f(x,y))), ((\neg P(x) \wedge \neg P(y)) \vee P(f(x,y)))\varepsilon}\ (\vee)}{\exists x \exists y((\neg P(x) \wedge \neg P(y)) \vee P(f(x,y))), \exists x \exists y((\neg P(x) \wedge \neg P(y)) \vee P(f(x,y)))}\ (\exists)}{\exists x \exists y((\neg P(x) \wedge \neg P(y)) \vee P(f(x,y)))}\ (fac)$$

Figure 11: An $\mathbf{IK}_\xi$-derivation

4 Early history

Cut-free sequent calculi for first-order logic introduced by Gentzen [76] turned out to be an important tool for investigating basic proof-theoretic problems (e.g. Gentzen [77] and Girard [79]). It has also been realized that sequent systems give a convenient tool for designing proof-search algorithms.

One of the first sequent-based proof procedures has been proposed by Kanger [90] independently of the more known papers of Beth [27, 26] (the monograph [90] is based on lectures made in Stockholm University in 1955). The distinctive features of the calculi used in Kanger [90, 91] are the absence of structure rules and invertibility of all inference rules (calculi with similar properties have been studied in Matulis [113, 114] and Curry [35]). Antecedents and succedents of sequents in Kanger's calculi are multisets of formulas. Invertibility of rules allowed Kanger to prove completeness "by means of arguments which are new in some respect and which involve a new turn to the notion of validity". This means that if a sequent $\mathcal{S}$ is unprovable, then *any* derivation tree for $\mathcal{S}$ has a branch containing a countermodel for $\mathcal{S}$.

Search for a countermodel is the basis for the tableau method developed by Beth [27, 26]. In [26] Beth noted that his tableau calculus F can equivalently be described as a sequent calculus similar to the Kleene's $G3$ [92]. In fact, the tableau calculus of Beth was even more similar to the calculus of Kanger [90].

As far as we know, the first sequent-based proof system for logic with equality has been proposed and *implemented* by Wang [162]. Equality has been axiomatized by one axiom

$$\Gamma \to \Delta, t \approx t$$

and one inference rule

$$\frac{\Gamma, t \approx s \to \Delta[x \mapsto s]}{\Gamma, t \approx s \to \Delta[x \mapsto t]}$$

The first extensive analysis of sequent calculi for equational logic have been made in Kanger [91] and in the papers of the Leningrad group of mathematical logic (Rogava [137], Lifschitz [96, 97], Orevkov [125] and Pliuškevičiene [131]). The main rule for handling equality used by Kanger is based on the simultaneous replacement of all occurrences of the same subterm:

$$\frac{s \approx t, \Gamma_t^s \to \Delta_t^s}{s \approx t, \Gamma \to \Delta}$$

where Γ_t^s denotes the result of the simultaneous replacement of all occurrences of the term s by t in Γ. This rule is close to *simultaneous paramodulation* of Benanav [25]. It is notable that the system with simultaneous paramodulation has the lifting property.

Kanger [91] also introduced some restrictions on derivations that are interesting from the viewpoint of the current knowledge of the area. First, he only allowed *nonincreasing* applications of the above equality rule, i.e. applications in which the depth of t is not greater than the depth of s. Second, he restricted himself to *regular derivations* only.

Lifschitz [96, 97] and Orevkov [125] considered some extravagant ways of orientation of equations. For example, Orevkov [125] oriented equations using *any* asymmetric binary relation R. The replacement of s by t using $s \approx t$ has only been allowed when it is not true that sRt.

In addition to the brilliant[4] formalization of equality, the free variable system of Kanger used a new strategy for instantiating variables in the applications of γ-rules (i.e. the rules $(\to \exists)$ and $(\forall \to)$). His strategy of instantiating variables is based on two ideas: the use of free variables and instantiation of free variables by terms already occurring in the derivation.

Free variables used by Kanger have been originally introduced by Prawitz [132] for logic without equality. The idea is to introduce a new kind of variables (called *"dummies"* in both Prawitz [132] and Kanger [91]), and to delay instantiation of these variables until necessary information for it is obtained. Comparing this approach to an earlier work of Beth [26], Prawitz [132] noted: "the solution proposed here is quite different but well-suited for mechanical use". This method was independently proposed in Russia by N. Shanin in 1962 (see [106]) and has been characterized as the *"metavariable method"* in Maslov, Mints and Orevkov [112]. Information for instantiation is provided by constructing an uninstantiated proof, and checking from time to time whether one can find values for dummies which make it a valid proof. In Kanger [91] this check is reduced to verifying that the top sequents are "directly demonstrable", i.e. can be obtained from axioms by applications of equality rules. The possibility of applying all equality rules before all other

[4]Brilliant, since Kanger has anticipated many tendencies used in the modern methods of handling equality in automated deduction.

rules has also been demonstrated in Orevkov [125] and used in Maslov [107].

Dummies or metavariables have later been called *"free variables"* in Fitting [64]. For instantiation of dummies, or free variables, Kanger [91] proposed an approach later called *minus-normalization* in Maslov and Mints [111]. According to this approach, instantiation of variables in the applications of γ-rules from the conclusion to the premise is made only by ground terms explicitly occurring in the conclusion. This method is complete for first-order logic. However, for a language with function symbols minus-normalization is interesting mostly theoretically. Even in simplest cases, minus-normalization requires a huge number of instantiations. For example, in the tableau of Example 1.3, we have to consider 8^4 possible instantiations of variables x, y, u, v by terms in the set $\{a, b, c, d, fa, fb, fc, fd\}$. Moreover, the use of minus-normalization can lead to considerable growth of derivations. Some results on minus-normalization are proved in Norgela [124].

In the modern terminology, we can say that minus-normalization gives an incomplete (but terminating) algorithm for simultaneous rigid E-unification discussed in Section 5.

For free variable tableaux, more practical way of instantiating variables is the introduction of unification in inference rules, for example branch closure rules considered in subsequent sections. Fitting [65] writes

> It occurred independently to several people that free variables could be used instead, in a way that gave an important role to unification.

Results considered in this section provided the proof-theoretic basis for further research in sequent calculi with equality. Further research introduced methods based on simultaneous rigid E-unification and variants of rigid paramodulation considered in Sections 6 and 7.

5 Rigid E-unification

In Section 3 we considered the free variable version **TK** of semantic tableaux (in fact, without equality). In this section we consider the problem of instantiating free variable semantic tableaux with equality which gives rise to *simultaneous rigid E-unification* introduced by Gallier, Raatz and Snyder [72]. In Section 5.1 we define simultaneous rigid E-unification and show how it arises in semantic tableaux with equality. In Section 5.2 we consider some properties of simultaneous rigid E-unification.

5.1 Simultaneous rigid E-unification

The actual problem in the tableau-based theorem proving with equality, as established by Theorem 3.1, is the problem of *finding a substitution* that makes all tableau branches closed. The problem of finding such a substitution, together with Theorem 3.4, leads to *simultaneous rigid E-unification.* Simultaneous rigid E-unification arises after we fixed, according to Theorem 3.4, a literal $s \not\approx t$ or two literals $A, \neg B$ on every branch of a tableau.

Let us give formal definitions.

- A *rigid equation* is an expression of the form $E \vdash_\forall s \approx t$, where E is a finite set of equations and s, t are terms. The set E is called *the left-hand side* of this rigid equation, and $s \approx t$ is called its *right-hand side*.

- A *solution* to such a rigid equation is any substitution θ such that $E\theta \vdash s\theta \approx t\theta$. A rigid equation is *solvable* if it has a solution.

- The *rigid E-unification* problem is the problem of determining whether a given rigid equation is solvable.

In other words, θ is a solution to a rigid equation $E \vdash_\forall s \approx t$ if and only if the set $E\theta \cup \{s\theta \not\approx t\theta\}$ is inconsistent.

- A *system of rigid equation* is any finite set of rigid equations;

- A *solution* to a system R of rigid equations is any substitution which is a solution to every rigid equation in R;

▶ The *simultaneous rigid E-unification* problem is the problem of determining whether a given system of rigid equations possesses a solution.

In Example 3.3, to close all branches in the tableau, we had to find a substitution θ such that the following two multisets of literals are inconsistent:

$$\begin{array}{l} \{a \approx b, g(x,u,v)\theta \not\approx g(y,fc,fd)\theta\}, \\ \{c \approx d, g(u,x,y)\theta \not\approx g(v,fa,fb)\theta\}. \end{array} \tag{11}$$

This (simultaneous) inconsistency problem can be expressed through the system of two rigid equations:

$$\begin{array}{l} a \approx b \vdash_{\forall} g(x,u,v) \approx g(y,fc,fd), \\ c \approx d \vdash_{\forall} g(u,x,y) \approx g(v,fa,fb). \end{array} \tag{12}$$

This system of rigid equations has one solution $[x \mapsto fa, y \mapsto fb, u \mapsto fc, v \mapsto fd]$. In order to check it, we should verify

$$\begin{array}{l} a \approx b \vdash g(fa,fc,fd) \approx g(fb,fc,fd), \\ c \approx d \vdash g(fc,fa,fb) \approx g(fd,fa,fb). \end{array}$$

In general, there can be more than one system of rigid equations, each of whom expressing that all branches in a tableau are closed. For example, for the tableau

$$T\ a \approx b, F\ b \approx c, F\ a \approx x \mid T\ P(x), F\ P(b)$$

we can construct two systems of rigid equations expressing its inconsistency:

$$\begin{array}{l} a \approx b \vdash_{\forall} b \approx c \\ \vdash_{\forall} x \approx b \end{array} \qquad \text{and} \qquad \begin{array}{l} a \approx b \vdash_{\forall} a \approx x \\ \vdash_{\forall} x \approx b \end{array}$$

Simultaneous rigid E-unification was introduced in Gallier, Raatz and Snyder [72] who hoped that the problem is decidable. The terminology "simultaneous rigid E-unification" can be explained as follows. The word "rigid" is introduced to distinguish rigid E-unification from *E-unification*. The latter problem can be formulated as follows. Given a (finite) set of equations $E = \{s_1 \approx t_1, \ldots, s_n \approx t_n\}$ and the equation $s \approx t$, find a substitution θ such that $\forall(s_1 \approx t_1), \ldots, \forall(s_n \approx t_n) \vdash s\theta \approx t\theta$. For rigid E-unification, we try to find a substitution θ such that $\vdash \forall(s_1\theta \approx t_1\theta \wedge \ldots \wedge s_n\theta \approx t_n\theta \supset s\theta \approx t\theta)$.

In E-unification, all variables in the equations $s_i \approx t_i$ are treated as universal, while all variables in rigid equations are *rigid*: if we substitute a term for a variable in any part of a rigid equation, we must substitute it in the whole rigid equation. The word "simultaneous" means that we have to find a simultaneous solution to several rigid equations.

Consider an example rigid equation $x \cdot y \approx y \cdot x \vdash_\forall a \cdot (b \cdot a) \approx (a \cdot b) \cdot a$. It has no solutions. However, we have $\forall x \forall y(x \cdot y \approx y \cdot x) \vdash a \cdot (b \cdot a) \approx (a \cdot b) \cdot a$, and hence the terms $a \cdot (b \cdot a)$ and $(a \cdot b) \cdot a$ are E-unifiable with respect to the equality theory $E = \{x \cdot y \approx y \cdot x\}$.

Surprisingly, rigid variables in the context of resolution theorem proving have been used much earlier by Chang [34] and Chang and Lee [95] in their *V-resolution* and *V-paramodulation* rules. Chang and Lee tried to use rigid variables in order to capture Prawitz's procedure by resolution and to formalize the idea of reasoning with bounded resources. Later, V-paramodulation have been described as *rigid paramodulation* by Plaisted [129]. We consider resolution-based theorem proving with rigid variables in Section 7.3.

5.2 Undecidability and other properties of simultaneous rigid E-unification

Gallier, Raatz and Snyder [72] introduced simultaneous rigid E-unification and asked whether this problem is decidable. Several papers published in 1988–1992 [71, 69, 70, 68, 82] described faulty proofs of the decidability. Finally, in 1995 the problem was proved to be undecidable in Degtyarev and Voronkov [52] by reduction of *monadic semi-unification* whose undecidability was proved by Baaz [5]. More comprehensive proofs of the undecidability appeared in Degtyarev and Voronkov [51, 57] by reduction of second-order unification whose undecidability was proved by Goldfarb [80] and in Degtyarev and Voronkov [56] by reduction of tenth Hilbert's problem. We shall briefly explain the ideas of the undecidability proof of Degtyarev and Voronkov [51, 57] and consider further results on simultaneous rigid E-unification below.

This undecidability proof is also interesting because it gives an encoding of second-order unification by simultaneous rigid E-unification. We shall present second-order unification here informally, for more detail see Goldfarb

[80] or Degtyarev and Voronkov [51, 57]. *Second-order unification* can be considered as a formalization of application of substitutions to terms.

Second-order unification differs from ordinary unification by the use of second-order variables. A second-order variable x may occur in terms in the form $x(t_1, \dots, t_n)$. Second-order substitutions substitute for such variables λ-terms of the form $\lambda w_1 \dots w_n.t$, where t is any first-order term. The application of the substitution $[x \mapsto \lambda w_1 \dots w_n.t]$ to the term $x(t_1, \dots, t_n)$ yields the term $t[w_1 \mapsto t_1, \dots, w_n \mapsto t_n]$. Ordinary first-order variables can be considered as second-order variables of 0 arguments.

An example of a second-order unification problem is $f(z(y(a))) \approx z(y(b))$. By introduction of new variables, any such problem can be reduced to a set of equations of the form $x(t_1, \dots, t_n) \approx t$, where $n \geq 0$ and $t, t_1, \dots, t_n$ are first-order terms. For example, the second-order equation $f(z(y(a))) \approx z(y(b))$ can be reduced to the following set of equations:

$$\{y(a) \approx u_1, z(u_1) \approx u_2, y(b) \approx u_3, z(u_3) \approx f(u_2)\}.$$

Consider the following second-order equation

$$x(y, g(y)) \approx f(g(z), a, y). \tag{13}$$

We will be interested in its ground solutions, i.e. in solutions θ such that $x\theta, y\theta, z\theta$ are ground terms. We also assume that we are looking for its solutions in the signature $\{f, g, a, b\}$. Among ground solutions to this equation are the following:

$$[x \mapsto \lambda w_1 w_2.f(g(b), a, b), y \mapsto b, z \mapsto b], \tag{14}$$
$$[x \mapsto \lambda w_1 w_2.f(w_2, a, w_1), y \mapsto b, z \mapsto b], \tag{15}$$
$$[x \mapsto \lambda w_1 w_2.f(w_2, w_1, a), y \mapsto a, z \mapsto a], \tag{16}$$
$$[x \mapsto \lambda w_1 w_2.f(w_1, a, g(b)), y \mapsto g(b), z \mapsto b]. \tag{17}$$

In order to encode second-order unification, one can use the following properties of simultaneous rigid E-unification (Degtyarev and Voronkov [51, 57]). First, one can express the property of being a ground term of a given signature $\mathcal{G}$ having at least one constant c. Let t be any term. Introduce the following rigid equation:

$$Gr(t, \mathcal{G}) \rightleftharpoons \{f(c, \ldots, c) \approx c \mid f \in \mathcal{G}\} \vdash_\forall t \approx c.$$

We have

Lemma 5.1 A substitution θ is a solution to $Gr(t, \mathcal{G})$ if and only if $t\theta$ is a ground term of the signature $\mathcal{G}$.

In a similar way, one can represent all regular sets (Goubault [82], Plaisted [129] and Veanes [148]).

Second, simultaneous rigid E-unification can represent application of substitutions. We shall temporarily use the substitution notation $[c_1 \mapsto t_1, \ldots, c_n \mapsto t_n]$, where c_i are constants, to denote the operation of the simultaneous replacement of *all* occurrences of c_i by t_i. Representation of substitutions is based on the following property of equational logic:

Lemma 5.2 Let $c_1, \ldots, c_n$ be pairwise different constants and $t_1, \ldots, t_n$ be first order terms such that c_i does not occur in t_j for all $i, j \in \{1, \ldots, n\}$. Then $c_1 \approx t_1, \ldots, c_n \approx t_n \vdash s_1 \approx s_2$ if and only if $s_1[c_1 \mapsto t_1, \ldots, c_n \mapsto t_n] = s_2[c_1 \mapsto t_1, \ldots, c_n \mapsto t_n]$.

This lemma and Lemma 5.1 are enough to represent second-order unification. Indeed, consider any second-order equation of the form $x(t_1, \ldots, t_n) \approx t$ in a signature $\mathcal{G}$. Let $c_1, \ldots, c_n$ be new constants. Consider be the following system R of rigid equations:

$$\begin{array}{l}
Gr(t_1, \mathcal{G}), \\
\ldots \\
Gr(t_n, \mathcal{G}), \\
Gr(t, \mathcal{G}), \\
Gr(x, \mathcal{G} \cup \{c_1, \ldots, c_n\}), \\
c_1 \approx t_1, \ldots, c_n \approx t_n \vdash_\forall x \approx t.
\end{array}$$

Let θ be any substitution. Consider when θ solves this system of rigid equations. By Lemma 5.1, all terms $t_1\theta, \ldots, t_n\theta, t\theta$ are ground terms in the signature $\mathcal{G}$, and $x\theta$ is a ground term in $\mathcal{G} \cup \{c_1, \ldots, c_n\}$. Since $c_1, \ldots, c_n$ do not occur in $t_1\theta, \ldots, t_n\theta, t\theta$, we can apply Lemma 5.2 to the last rigid equation in R, obtaining

$$x\theta[c_1 \mapsto t_1\theta, \ldots, c_n \mapsto t_n\theta] = t\theta.$$

Let $\theta = [x_1 \mapsto s_1, \dots , x_m \mapsto s_m, x \mapsto s]$. It is easy to see that θ is a solution to R if and only if the substitution $[x_1 \mapsto s_1, \dots , x_m \mapsto s_m, x \mapsto \lambda c_1 \dots c_n.s]$ is a solution to $x(t_1, \dots , t_n) \approx t$. Thus, we can encode second-order unification in simultaneous rigid E-unification in a natural way.

If we apply this encoding to second order equation (13), we obtain the following set of rigid equations:

$$
\begin{array}{l}
f(a,a,a) \approx a, g(a) \approx a, b \approx a \vdash_\forall y \approx a, \\
f(a,a,a) \approx a, g(a) \approx a, b \approx a \vdash_\forall z \approx a, \\
f(a,a,a) \approx a, g(a) \approx a, b \approx a, c_1 \approx a, c_2 \approx a \vdash_\forall x \approx a, \\
c_1 \approx y, c_2 \approx g(y) \vdash_\forall x \approx f(g(z), a, y).
\end{array}
$$

Some solutions to this system of rigid equations are

$$
\begin{array}{l}
[x \mapsto f(g(b), a, b), y \mapsto b, z \mapsto b]; \\
[x \mapsto f(c_2, a, c_1), y \mapsto b, z \mapsto b]; \\
[x \mapsto f(c_2, c_1, a), y \mapsto a, z \mapsto a]; \\
[x \mapsto f(c_1, a, g(b)), y \mapsto g(b), z \mapsto b].
\end{array}
$$

The reader can compare them with solutions (14)–(17) of the original second-order equation. Other relations between second-order unification and simultaneous rigid E-unification are discussed in Veanes [151].

Although the undecidability result for simultaneous rigid E-unification seems to diminish the value of this notion, it has some other applications. In [69] Gallier et.al. noted that "rigid E-unification and Girard's linear logic [78] share the same spirit". The further development of results concerning simultaneous rigid E-unification has shown its close connections to a remarkable number of problems in logic with equality. Let us briefly review some of these results.

The undecidability of simultaneous rigid E-unification implies the undecidability of the following problems:

1. $\exists^*$*-fragment of intuitionistic logic with equality*, and also *the prenex fragment of intuitionistic logic with equality* (Degtyarev and Voronkov [56]). This result is interesting because both these fragments for intuitionistic logic without equality are decidable and PSPACE-complete (Degtyarev and Voronkov [53], Voronkov [157]).

2. the *Herbrand skeleton problem* (in the terminology of Voda and Komara [152]) is the following series of decision problems, for every natural

number n. Given a formula $\exists\bar{x}\varphi(\bar{x})$, do there exist sequences of terms $\bar{t}_1, \ldots, \bar{t}_n$ such that the formula $\varphi(\bar{t}_1) \vee \ldots \vee \varphi(\bar{t}_n)$ is provable. This problem naturally arises from the Herbrand theorem: a formula $\exists\bar{x}\varphi(\bar{x})$ is provable if and only if there exists a natural number n and sequences of terms $\bar{t}_1, \ldots, \bar{t}_n$ such that the formula $\varphi(\bar{t}_1) \vee \ldots \vee \varphi(\bar{t}_n)$ is provable.

3. *skeleton instantiation problem* (also considered in Section 9): given a formula and a derivation skeleton, is there a derivation of this formula with this skeleton.

and some similar problems (see Degtyarev, Gurevich and Voronkov [43], Voronkov [161].

The undecidability of simultaneous rigid E-unification posed a problem of its replacement by other techniques. These techniques will be considered in Sections 6 and 7. However, in some cases automated reasoning with simultaneous rigid E-unification is still possible, because of the decidability of some its fragments. In fact, the (un)decidability of simultaneous rigid E-unification depends on the signature $\mathcal{F}$. We formulate some results related to special cases of simultaneous rigid E-unification:

1. Gallier et.al. [71] proved the decidability and NP-completeness of rigid E-unification. Although it is an interesting result by itself, it does not contribute much to automated reasoning problems because practical problems usually give rise to the simultaneous case.

2. Degtyarev, Matiyasevich and Voronkov [45] note that the function-free fragment of simultaneous rigid E-unification is NP-complete.

3. Degtyarev, Matiyasevich and Voronkov [45] prove the decidability of simultaneous rigid E-unification in the language with one unary function symbol and any number of constants (which implies the decidability of all problems mentioned above for this language). The complexity of this fragment is unknown since the proof uses a reduction to the Diophantine problem for addition and divisibility (Beltyukov [24], Mart'janov [105] and Lipshitz [99]) whose complexity is now known (Lipshitz [100]).

4. The decidability of simultaneous rigid E-unification in the signature $\mathcal{F}$ with only unary function symbols is an open problem. It is known (Degtyarev, Matiyasevich and Voronkov [45]) that it is decidable in

any such signature if and only if it is decidable in the signature with two unary function symbols and any number of constants. In addition, the word equation problem (Makanin [104]) has a simple reduction to simultaneous rigid E-unification in such signatures. (It is easy to show such a reduction by modifying the encoding of second-order unification considered above to the case of unary function symbols.) Gurevich and Voronkov [83, 84] proved that simultaneous rigid E-unification in the case of unary function symbols is equivalent to an extension of word equations.

5. Simultaneous rigid E-unification with ground left-hand sides is undecidable (Plaisted [129]).

6. Veanes [149, 150] improved this results by showing that the simultaneous rigid E-unification is undecidable even in the case of one binary function symbol, one constant, two variables and ground left-hand sides.

7. Degtyarev et.al. [41, 42] proved that simultaneous rigid E-unification with one variable is decidable.

We do not give details and techniques used in these results because many of them are quite nontrivial. For a more thorough discussion see Voronkov [161]. An analysis of first-order theorem proving based on rigid variables may be found in Voronkov [160].

5.3 Other works

The first procedure for rigid E-unification described in Gallier et.al. [71] has not been intended as an efficient procedure for rigid E-unification, but rather as a proof of its NP-completeness. Later, several papers described other algorithms for rigid E-unification (Goubault [81], Beckert [18], Becher and Petermann [17] and De Kogel [37]). These algorithms have been presented as either more efficient or more simple than the original algorithm of Gallier et.al. The interest in papers on (non-simultaneous) rigid E-unification has been motivated by two reasons.

First, such algorithms were used in implementations of tableau provers with equality (for example, Beckert [18], Petermann [127] and Beckert [21]).

Unlike Gallier et.al.'s procedure, procedures of these papers use infinite sets of solutions or are non-terminating.

Second, for a long time there have been a hope to construct a decision procedure for simultaneous rigid E-unification by using finite complete sets of minimal solutions to rigid equations, maybe by changing the notion of a minimal solution. The undecidability result for simultaneous rigid E-unification has shown that this is not possible.

Moreover, it is hard to expect that there are decidable fragments of simultaneous rigid E-unification with a reasonably low complexity bound, except for the function-free fragment. Hence, such decision procedures are hardly interesting for theorem proving in classical logic. However, such procedures and a semi-decision procedure for simultaneous rigid E-unification are of a definite interest in connection with theorem proving in nonclassical predicate logics with equality, as explained in Section 9.

There are papers which define and study so-called *mixed E-unification* where some variables are treated as rigid and some as universal (Beckert [18, 20, 21]. It is proposed to apply it in the situation when there are "formulas universal on a branch with respect to a variable".

Two main sources of references to results on rigid E-unification are Veanes [149] and Voronkov [161].

6 Incomplete procedures for rigid E-unification

In this chapter we describe an incomplete procedure for simultaneous rigid E-unification which is complete for theorem proving in first-order logic. This procedure has been originally introduced in Degtyarev and Voronkov [58]. *In this chapter all formulas are assumed to be skolemized, i.e., contain no positive occurrences of* $\forall$ *or negative occurrences of* $\exists$.

6.1 Gallier et.al.'s procedure: merits and problems

Gallier et.al. [71, 69, 70, 68] have shown how to use rigid E-unification in conjunction with equational matings. A corresponding formal description of a procedure is given, for example in Gallier et.al. [70]. This procedure is based on the following premises:

1. For every rigid equation there exists a finite *complete set of minimal solutions.* This set is finite and can be computed by some terminating algorithm.
2. For every solvable system S of rigid equations, some of its solutions can be found incrementally (rigid-equation-wise) by a consecutive combination of finite complete sets of solutions for components of S.

Unfortunately, the second premise happened to be false, as noted in Becher and Petermann [17]. To illustrate Gallier et.al.'s procedure, we introduce one definition. Let Γ be a tableau branch.

- ▶ The *set of rigid equations on* Γ is defined in the following way. A rigid equation $E \vdash_{\forall} s \approx t$ is *on* Γ if $E = \{s' \approx t' \mid (T\, s' \approx t') \in \Gamma\}$ and $(F\, s \approx t) \in \Gamma$. (This definition is equivalent to the one given by Beckert and Häehnle [22].)

Consider again Example 3.3. Gallier et.al.'s procedure would select one of the branches, for example the left one, and one rigid equation on that branch. In our case, there are two such rigid equations: $\vdash_{\forall} g(x,u,v) \approx g(y,fc,fd)$ and $a \approx b \vdash_{\forall} g(x,u,v) \approx g(y,fc,fd)$. These rigid equations have the same complete set of minimal solutions consisting of one substitution $[x \mapsto y, u \mapsto fc, v \mapsto fd]$. According to the procedure, any substitution of the finite complete set of solutions is applied to the whole tableau, and the procedure is continued with the rest of the branches. It was asserted in the above mentioned papers that such a procedure gives a decision procedure for simultaneous rigid E-unification. In other terms, if a tableau is substitutively inconsistent, the procedure can make all branches closed without any applications of tableau expansion rules. But in our case, after the application of the substitution $[x \mapsto y, u \mapsto fc, v \mapsto fd]$ to the tableau we obtain the tableau which is not substitutively inconsistent any more. Thus, it cannot now be closed without expansions. It means that the second premise of Gallier et.al.' procedure is incorrect.

To close the tableau of this example, we need to consider a nonminimal solution to the second equation $a \approx b \vdash_{\forall} g(x, u, v) \approx g(y, fc, fd)$. If we choose the solution $[x \mapsto fa, y \mapsto fb, u \mapsto fc, v \mapsto fd]$ instead of a minimal one $[x \mapsto y, u \mapsto fc, v \mapsto fd]$, we would obtain a closed tableau. However, when we do not restrict ourselves by a complete set of minimal solutions, we lose termination.

We can note that *any* procedure that terminates for a given tableau cannot check whether the tableau is substitutively inconsistent, due to the undecidability of simultaneous rigid E-unification.

All these considerations do not imply that the procedure is incomplete for first-order logic with equality. Indeed, it is possible that after some additional applications of tableau expansion rules the procedure will find a solution. Using the language of matings, the solution can probably be found for some (not necessarily minimal) amplification.

Thus, Gallier et.al.'s procedure is an example of a tableau-based procedure for establishing provability in first-order logic with equality, which uses an incomplete but terminating algorithm for simultaneous rigid E-unification. This procedure generalizes tableau-based algorithms for logic with equality. Its main advantage is that substitutive inconsistency for the whole tableau can be found by an incremental branch closure, where branches are closed one by one. However, it is still unknown whether the Gallier et.al.'s procedure is complete for first-order logic with equality.

The use of an incomplete procedure for (simultaneous) rigid E-unification in tableau-based methods has been circulated in the literature (see e.g. Petermann [127] or Beckert [19]), especially after the first proof of the undecidability of simultaneous rigid E-unification (Degtyarev and Voronkov [52]) that appeared in May 1995. The question of the existence of such a strategy was open for some time. In the next section we describe a calculus that gives such a procedure.

6.2 An incomplete calculus for rigid E-unification

We shall use the term "rigid basic superposition" to denote a "rigid" version of basic superposition. We formalize rigid basic superposition using constraints that is close to the presentation of Nieuwenhuis and Rubio [123]. We shall use ordering constraints as defined in Section 2.2.

Below we shall introduce an inference system **BSE** (Basic Superposition with Equality Solution) for solving rigid equations. The provable objects of **BSE** are *constrained rigid equations.* In this section we display constraints as sets of atomic equality and inequality constraints, instead of a conjunction.

- A *constrained rigid equation* is a pair consisting of a rigid equation R and a constraint $\mathcal{C}$. Such a constrained rigid equation will be denoted $R \cdot \mathcal{C}$.

- The calculus **BSE** consists of the following inference rules:

  - *Rigid basic (right and left) superposition rules* are the following inference rules:

$$\frac{E \cup \{s \approx t\} \vdash_\forall u[s'] \approx v \cdot \mathcal{C}}{E \cup \{s \approx t\} \vdash_\forall u[t] \approx v \cdot \mathcal{C} \cup \{s \succ t, u[s'] \succ v, s = s'\}}\ (rbrs)$$

and

$$\frac{E \cup \{s \approx t, u[s'] \approx v\} \vdash_\forall e \cdot \mathcal{C}}{E \cup \{s \approx t, u[t] \approx v\} \vdash_\forall e \cdot \mathcal{C} \cup \{s \succ t, u[s'] \succ v, s = s'\}}\ (rbls),$$

respectively.

  - *Rigid equality solution* is the following inference rule:

$$\frac{E \vdash_\forall s \approx t \cdot \mathcal{C}}{E \vdash_\forall s \approx s \cdot \mathcal{C} \cup \{s = t\}}\ (reqs)$$

Application of all the rules is restricted to the following conditions:

1. The rigid equation at the conclusion of the rule is satisfiable;

2. In (*reqs*), $s \neq t$;
3. In the rigid basic superposition rules, the term s' is not a variable;
4. In (*rbls*), $u[t] \neq v$.

▶ The *calculus* **BSE** is the logical calculus whose inference rules are (*rbls*), (*rbrs*) and (*reqs*).

The basic restriction in **BSE** is formalized by representing most general unifiers through equality constraints. Condition 1 has two purposes. The satisfiability of equations in constraints is needed to preserve correctness of the method. The satisfiability of inequality constraints is needed to ensure termination (Theorem 6.3 below). Condition 2 is needed to forbid infinite chains of repeated applications of (*reqs*). Conditions 3 and 4 are not essential, they are added as a standard optimization used in paramodulation-based methods.

▶ We denote by $R \cdot \mathcal{C} \rightsquigarrow R' \cdot \mathcal{C}'$ the fact that $R' \cdot \mathcal{C}'$ is obtained from $R \cdot \mathcal{C}$ by an application of an inference rule of **BSE**. The symbol $\rightsquigarrow^*$ denotes the reflexive and transitive closure of $\rightsquigarrow$.

Consider an example derivation in **BSE**.

Example 6.1 Consider the rigid equation $hx \approx a, ha \approx a, hb \approx fy \vdash_\forall y \approx gfy$. The ordering $\succ$ is based on the precedence relation $f \succ_\mathcal{F} h \succ_\mathcal{F} b \succ_\mathcal{F} a$. Under this ordering we have $hb \succ a$ and $fy \succ hb$. The following is a **BSE**-derivation for this rigid equation:

$$
\cfrac{\cfrac{\cfrac{hx \approx a, ha \approx a, hb \approx fy \vdash_\forall y \approx gfy \cdot \emptyset}{hx \approx a, ha \approx a, hb \approx fy \vdash_\forall y \approx ghb \cdot \{fy \succ hb, gfy \succ y, fy = fy\}}\,(rbrs)}{\begin{array}{l} hx \approx a, ha \approx a, hb \approx fy \vdash_\forall y \approx ga \\ \quad \cdot \{fy \succ hb, gfy \succ y, fy = fy, hx \succ a, ghb \succ y, hx = hb\}\end{array}}\,(rbrs)}{\begin{array}{l} hx \approx a, ha \approx a, hb \approx fy \vdash_\forall y \approx y \\ \quad \cdot \{fy \succ hb, gfy \succ y, fy = fy, hx \succ a, ghb \succ y, hx = hb, y = ga\}\end{array}}\,(reqs)
$$

By using *constraint simplification*, i.e. replacement of constraints by equivalent "simpler" constraints we can rewrite this derivation as

$$\cfrac{\cfrac{hx \approx a, ha \approx a, hb \approx fy \vdash_\forall y \approx gfy \cdot \emptyset}{hx \approx a, ha \approx a, hb \approx fy \vdash_\forall y \approx ghb \cdot \emptyset}\ (rbrs)}{\cfrac{hx \approx a, ha \approx a, hb \approx fy \vdash_\forall y \approx ga \cdot \{ghb \succ y, x = b\}}{hx \approx a, ha \approx a, hb \approx fy \vdash_\forall y \approx y \cdot \{x = b, y = ga\}}\ (reqs)}\ (rbrs)$$

Soundness of **BSE** can be formulated in the following way:

Theorem 6.2 (Soundness of BSE) Let $R \cdot \emptyset \rightsquigarrow^* E \vdash_\forall s \approx t \cdot \mathcal{C}$. Then every substitution satisfying $\mathcal{C}$ is a solution to R. In particular, R is solvable.

This theorem leads to the following definition.

- ► A constraint $\mathcal{C}$ is called an *answer constraint* for a rigid equation R if for some rigid equation $E \vdash_\forall s \approx s$ we have $R \cdot \emptyset \rightsquigarrow^* E \vdash_\forall s \approx s \cdot \mathcal{C}$.

We note that **BSE** is an incomplete calculus for solving rigid equations. It means that there are solvable rigid equations R that have no answer constraint. For instance, consider the rigid equation[5] $x \approx a \vdash_\forall gx \approx x$. It has one solution $[x \mapsto ga]$. However, no inference rule of **BSE** is applicable to this rigid equation.

Although the calculus **BSE** is incomplete, it has some pleasant termination properties:

Theorem 6.3 (Termination of BSE) For any constrained rigid equation $R \cdot \mathcal{C}$, there exists a finite number of derivations from $R \cdot \mathcal{C}$.

This property is due to the constraints: after some finite number of steps we shall not be able to apply any inference rule because the constraint in the conclusion of any rule would be unsatisfiable. Inequality constraints are not needed for soundness or completeness of **BSE**. The pragmatics behind inequality constraints is to ensure that the search for solutions to a rigid equation is finite. In addition, the use of ordering constraints prunes the search space.

In [58] left rigid basic superposition has been formulated incorrectly in the following way:

$$\frac{E \cup \{s \approx t, u[s'] \approx v\} \vdash_\forall e \cdot \mathcal{C}}{E \cup \{s \approx t, u[s'] \approx v, u[t] \approx v\} \vdash_\forall e \cdot \mathcal{C} \cup \{s \succ t, u[s'] \succ v, s = s'\}}\ (rbls)$$

[5] Suggested by G.Becher (private communication).

With this formulation, termination is not guaranteed as have been shown in [59].

To illustrate this theorem, consider again Example 6.1. The rigid equation of this example has an infinite number of solutions including $[x \mapsto a, y \mapsto gh^n a]$, for every natural number n. However, all possible **BSE**-derivations starting with $hx \approx a, ha \approx a, hb \approx fy \vdash_\forall y \approx gfy \cdot \emptyset$ give only two answer constraints, one is

$$\{fy \succ hb, gfy \succ y, fy = fy, hx \succ a, ghb \succ y, hx = hb, y = ga\}$$

shown in Example 6.1, another is $\{fy \succ hb, gfy \succ y, fy = fy, y = ghb\}$ obtained from the following derivation:

$$\cfrac{\cfrac{hx \approx a, ha \approx a, hb \approx fy \vdash_\forall y \approx gfy \cdot \emptyset}{hx \approx a, ha \approx a, hb \approx fy \vdash_\forall y \approx ghb \cdot \{fy \succ hb, gfy \succ y, fy = fy\}}\ (rbrs)}{\begin{array}{c} hx \approx a, ha \approx a, hb \approx fy \vdash_\forall y \approx y \cdot \\ \{fy \succ hb, gfy \succ y, fy = fy, y = ghb\} \end{array}}\ (reqs)$$

This answer constraint can be simplified to $\{y = ghb\}$. There are some other derivations with the same answer constraints, but only a finite number.

Theorem 6.3 yields

Theorem 6.4 Any rigid equation has a finite number of answer constraints. There is an algorithm giving by any rigid equation R the set of all answer constraints for R.

6.3 Answer constraints and the tableau method

In this section we consider how to use the system **BSE** for theorem proving by the tableau method. Since we only consider skolemized formulas, we have no δ-rules in tableau calculi.

We extend the notion of answer constraints to tableau branches. Let Γ be such a branch.

- ▶ A constraint $\mathcal{C}$ is a *answer constraint for* Γ if $\mathcal{C}$ is an answer constraint for some rigid equation on Γ.

By Theorem 6.4, we obtain

Theorem 6.5 Any tableau branch has a finite number of answer constraints. There is an algorithm giving by any tableau branch Γ the set of all answer constraints for Γ.

Note that every substitution satisfying an answer constraint for Γ is a substitution closing Γ.

The following theorem asserts soundness and completeness of the tableau method with answer constraints (Degtyarev and Voronkov [58]):

Theorem 6.6 Let φ be a sentence. Then φ is provable in first-order logic with equality if and only if there is a tableau $\mathcal{T}$ obtained from $F\varphi$ by tableau expansion rules with the following property. Let $\Gamma_1, \ldots, \Gamma_n$ be all branches of $\mathcal{T}$. Then there exist answer constraints $\mathcal{C}_1, \ldots, \mathcal{C}_n$ for $\Gamma_1, \ldots, \Gamma_n$, respectively, such that $\mathcal{C}_1 \cup \ldots \cup \mathcal{C}_n$ is satisfiable.

Note that if the constraint $\mathcal{C}_1 \cup \ldots \cup \mathcal{C}_n$ is satisfiable, then there exists a substitution θ satisfying this constraint. Since θ satisfies each $\mathcal{C}_i$, it closes all branches of the tableau.

Example 6.7 To illustrate this theorem, consider again the tableau of Example 3.3. The ordering $\succ$ is based on the precedence $g \succ_{\mathcal{F}} f \succ_{\mathcal{F}} a \succ_{\mathcal{F}} b \succ_{\mathcal{F}} c \succ_{\mathcal{F}} d$.

There is one rigid equation on each branch of the tableau:

$$a \approx b \vdash_{\forall} g(x, u, v) \approx g(y, fc, fd); \tag{18}$$
$$c \approx d \vdash_{\forall} g(u, x, y) \approx g(v, fa, fb). \tag{19}$$

Rigid basic superposition is applicable to none of these rigid equations. Rigid equation (18) has one answer constraint $\{g(x, u, v) = g(y, fc, fd)\}$ obtained by the following application of the rigid equality solution rule:

$$\frac{a \approx b \vdash_{\forall} g(x, u, v) \approx g(y, fc, fd) \cdot \emptyset}{a \approx b \vdash_{\forall} g(x, u, v) \approx g(x, u, v) \cdot \{g(x, u, v) = g(y, fc, fd)\}}\ (\mathit{reqs})$$

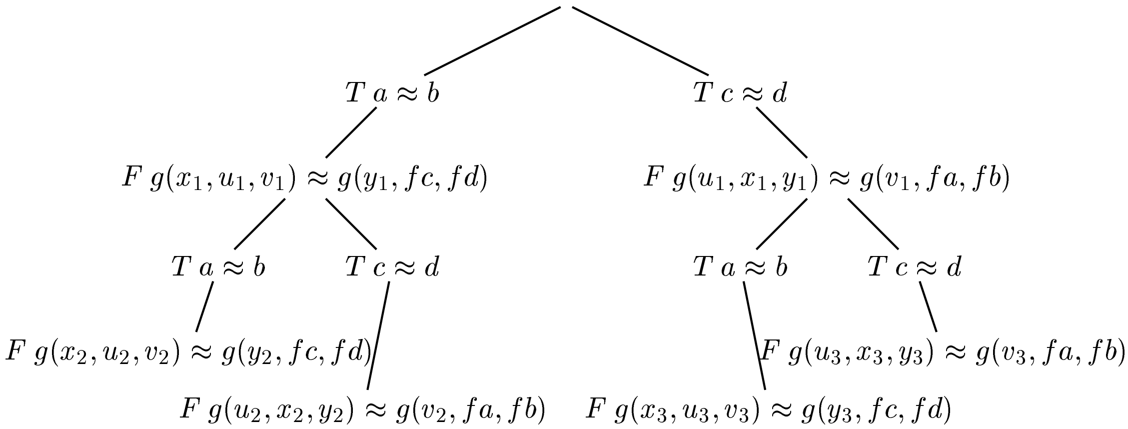
Similarly, rigid equation (19) has one answer constraint

$$\{g(u, x, y) = g(v, fa, fb)\}.$$

The union of these constraints is

$$\{g(x,u,v)=g(y,fc,fd),\, g(u,x,y)=g(v,fa,fb)\},$$

and it is inconsistent. Thus, our method find no solution for this substitutively inconsistent tableau. After more tableau expansion steps we obtain a tableau whose atomic part is the following:



It has four branches. The multisets of literals on the branches are

$$\begin{aligned}
S_1 &= \{a \approx b, g(x_1,u_1,v_1) \not\approx g(y_1,fc,fd), g(x_2,u_2,v_2) \not\approx g(y_2,fc,fd)\};\\
S_2 &= \{a \approx b, c \approx d, g(x_1,u_1,v_1) \not\approx g(y_1,fc,fd), g(u_2,x_2,y_2) \not\approx g(v_2,fa,fb)\};\\
S_3 &= \{a \approx b, c \approx d, g(u_1,x_1,y_1) \not\approx g(v_1,fa,fb), g(x_3,u_3,v_3) \not\approx g(y_3,fc,fd)\};\\
S_4 &= \{c \approx d, g(u_1,x_1,y_1) \not\approx g(v_1,fa,fb), g(u_3,x_3,y_3) \not\approx g(v_3,fa,fb)\}.
\end{aligned}$$

Consider the following rigid equations R_1–R_4 on the branches S_1–S_4, respectively:

$$\begin{aligned}
R_1 &= a \approx b \vdash_\forall g(x_2,u_2,v_2) \approx g(y_2,fc,fd);\\
R_2 &= a \approx b, c \approx d \vdash_\forall g(x_1,u_1,v_1) \approx g(y_1,fc,fd);\\
R_3 &= a \approx b, c \approx d \vdash_\forall g(u_1,x_1,y_1) \approx g(v_1,fa,fb);\\
R_4 &= c \approx d \vdash_\forall g(u_3,x_3,y_3) \approx g(v_3,fa,fb).
\end{aligned}$$

Apply the following **BSE**-derivations to R_1–R_4:

$$\frac{a \approx b \vdash_\forall g(x_2,u_2,v_2) \approx g(y_2,fc,fd) \cdot \emptyset}{a \approx b \vdash_\forall g(x_2,u_2,v_2) \approx g(x_2,u_2,v_2) \cdot \{g(x_2,u_2,v_2) = g(y_2,fc,fd)\}}\ (reqs)$$

$$\dfrac{\dfrac{a \approx b, c \approx d \vdash_\forall g(x_1, u_1, v_1) \approx g(y_1, fc, fd) \cdot \emptyset}{\begin{array}{l} a \approx b, c \approx d \vdash_\forall g(x_1, u_1, v_1) \approx g(y_1, fd, fd) \\ \quad \cdot \{c \succ d, g(y_1, fc, fd) \succ g(x_1, u_1, v_1), c = c\} \end{array}} \; (rbrs)}{\begin{array}{l} a \approx b, c \approx d \vdash_\forall g(x_1, u_1, v_1) \approx g(x_1, u_1, v_1) \\ \quad \cdot \{c \succ d, g(y_1, fc, fd) \succ g(x_1, u_1, v_1), \\ \quad\; c = c, g(x_1, u_1, v_1) = g(y_1, fd, fd)\} \end{array}} \; (reqs)$$

$$\dfrac{\dfrac{a \approx b, c \approx d \vdash_\forall g(u_1, x_1, y_1) \approx g(v_1, fa, fb) \cdot \emptyset}{\begin{array}{l} a \approx b, c \approx d \vdash_\forall g(u_1, x_1, y_1) \approx g(v_1, fb, fb) \\ \quad \cdot \{a \succ b, g(v_1, fa, fb) \succ g(u_1, x_1, y_1), a = a\} \end{array}} \; (rbrs)}{\begin{array}{l} a \approx b, c \approx d \vdash_\forall g(u_1, x_1, y_1) \approx g(u_1, x_1, y_1) \\ \quad \cdot \{a \succ b, g(v_1, fa, fb) \succ g(u_1, x_1, y_1), \\ \quad\; a = a, g(u_1, x_1, y_1) = g(v_1, fb, fb)\} \end{array}} \; (reqs)$$

$$\dfrac{c \approx d \vdash_\forall g(u_3, x_3, y_3) \approx g(v_3, fa, fb) \cdot \emptyset}{c \approx d \vdash_\forall g(u_3, x_3, y_3) \approx g(u_3, x_3, y_3) \cdot \{g(u_3, x_3, y_3) = g(v_3, fa, fb)\}} \; (reqs)$$

The union of the answer constraints of these derivations is

$$\begin{array}{l} \{g(x_2, u_2, v_2) = g(y_2, fc, fd), \\ \; c \succ d, g(y_1, fc, fd) \succ g(x_1, u_1, v_1), c = c, g(x_1, u_1, v_1) = g(y_1, fd, fd), \\ \; a \succ b, a = a, g(v_1, fa, fb) \succ g(u_1, x_1, y_1), g(u_1, x_1, y_1) = g(v_1, fb, fb), \\ \; g(u_3, x_3, y_3) = g(v_3, fa, fb)\}. \end{array}$$

It is easy to check that the following substitution makes all branches closed:

$$\begin{array}{l} [x_1 \mapsto fb, y_1 \mapsto fb, u_1 \mapsto fd, v_1 \mapsto fd, x_2 \mapsto b, y_2 \mapsto b, \\ \; u_2 \mapsto fc, v_2 \mapsto fd, u_3 \mapsto d, v_3 \mapsto d, x_3 \mapsto fa, y_3 \mapsto fb]. \end{array}$$

Hence, this constraint is satisfiable.

7 Sequent-based calculi and paramodulation

In order to construct a tableau-based free-variable system with equality, in Section 5 we used the way which can be characterized as *total theory reasoning* (Stickel [146]). Indeed, branch closure (or closure of the whole tableau) has been represented as one derivation step. This step included the detection of a literal set which is inconsistent in the equality theory. In contrast with this approach, the ground version $\mathbf{LK}^=$ of tableaux with equality is, in the same terminology, an example of *partial theory reasoning*. In $\mathbf{LK}^=$, the search for inconsistency of a sequent in the equality theory is realized through a sequence of derivation steps, mainly the application of the rule $(\approx)$ of replacement of equals by equals. A sequence of applications of this rule is intended to give a syntactically inconsistent sequent, i.e. a sequent inconsistent in the empty theory.

Within the resolution framework, paramodulation has been defined as a generalization of replacement of equals by equals. For free variable tableau-based systems such rules have first been formulated by Loveland [101] in the context of model elimination and by Fitting [64] in the context of tableaux by the name of the *mgu tableau replacement rule*. Characterizing this rule Fitting writes:

> It was necessary to modify the tableau system with equality by introducing free variables in order to get a version suitable for implementation. We must do a similar thing with the resolution system. The result is a variation of what is known in the literature as *paramodulation* ...

In this chapter all formulas are assumed to be skolemized.

7.1 Fitting's tableau paramodulation

Since paramodulation has appeared before and is a generally accepted term, we shall call the mgu tableau replacement rule simply *tableau paramodulation.*

▶ *Tableau paramodulation* is the following inference rule:

$$\frac{\Gamma_1, X\ A[s'], T\ s \approx t \mid \ldots \mid \Gamma_n}{(\Gamma_1, X\ A[s'], T\ s \approx t, X\ A[t] \mid \ldots \mid \Gamma_n)\mathrm{mgu}(s, s')}\ (tpar)$$

where A is an atomic formula.

In the tableau paramodulation rule, the substitution $\mathrm{mgu}(s, s')$ is applied to the whole tableau, i.e. it is "global". Free variables in tableaux are treated *rigidly,* as placeholders for terms, or unknown parameters, but not as universally quantified variables.

Besides the tableau paramodulation rule, the system of Fitting for equational logic includes the following two inference rules.

- *Free variable tableau reflexivity* of Fitting [64] is the following inference rule:

$$\frac{\Gamma_1 \mid \ldots \mid \Gamma_n}{\Gamma_1, T\ x \approx x \mid \ldots \mid \Gamma_n}$$

- *Tableau function reflexivity* of Fitting [64] is the following inference rule:

$$\frac{\Gamma_1 \mid \ldots \mid \Gamma_n}{\Gamma_1, T\ f(x_1, \ldots, x_n) \approx f(x_1, \ldots, x_n) \mid \ldots \mid \Gamma_n}$$

It is easy to see that instead of the last two rules one can change the initial tableau for proving a formula ξ from $F\ \xi$ to

$$\{T\ \forall\bar{x}(f(\bar{x}) \approx f(\bar{x})) \mid f \in \mathcal{F}\}, T\ \forall x(x \approx x), F\ \xi$$

Below we shall consider some difficulties in removing the function reflexivity rule from Fitting's system.

- Fitting's *free variable tableau system* $\mathbf{TK}^=$ for logic with equality consists of the rules of $\mathbf{TK}$ plus the tableau paramodulation rule, free variable tableau reflexivity rule and the tableau function reflexivity rule.[6]

This system is complete:

Theorem 7.1 Let ξ be a closed formula. The following conditions are equivalent:

[6]An inessential difference between our formulation of Fitting's system and Fitting's formulation of [64], as well as of [66] is that we remove closed branches from the tableau while in the system of [64] one can close the same branch several times. Also, Fitting does not give a name to his system, we call it here $\mathbf{TK}^=$ for convenience.

1. ξ is provable in first-order logic with equality;

2. there is a derivation of the empty tableau from the tableau $F\xi$ in $\mathbf{TK}^{=}$.

The system $\mathbf{TK}^{=}$ has all disadvantages of the early paramodulation rule of Robinson and Wos [134]:

1. Function reflexivity rule is used;

2. Paramodulation into variables is allowed;

3. Increasing applications of paramodulation are allowed (for example, x can be rewritten to fx using an equation $x \approx fx$.

As a consequence, for a given tableau expansion there may be an infinite sequence of paramodulations, either due to the use of function reflexivity or due to the use of increasing applications of paramodulation.

Fitting uses partial theory reasoning for adding equality to free-variable tableaux. He extended the tableau calculus **TK** by simple rules for equality. However, his scheme of proof-search is, in fact, based on finding all solutions for simultaneous rigid E-unification. Fitting's completeness theorem is formulated in the way similar to Theorem 3.1 and leads to a proof-search strategy consisting of two parts. In the first part a tableau is constructed using tableau expansion rules. In the second part, equality rules and branch closure rules are applied with the purpose of closing the tableau. The system $\mathbf{TK}^{=}$ has the following very strong property (a reformulation of Lifting Lemma of [64], page 220).

Theorem 7.2 If a tableau $\mathcal{T}$ is substitutively inconsistent, then there is derivation of the empty tableau from $\mathcal{T}$ in $\mathbf{TK}^{=}$ not using tableau expansion rules.

Unlike **TK**, the system $\mathbf{TK}^{=}$ may have *infinite derivations without tableau expansion rules,* i.e. the proof-search for a given tableau expansion may be nonterminating. It is not possible to restrict paramodulation in $\mathbf{TK}^{=}$ to obtain a system $\mathcal{S}$ which satisfies Theorem 7.2 and has the termination property for the subsystem of $\mathcal{S}$ without tableau expansion rules: any such system can be used as a decision procedure for simultaneous rigid E-unification. Moreover, if we drop function reflexivity, the resulting system would not satisfy Theorem 7.2:

Example 7.3 Consider the tableau of Example 3.3. Its atomic part has the form

$$T\ a \approx b, F\ g(x,u,v) \approx g(y, fc, fd) \mid T\ c \approx d, F\ g(u,x,y) \approx g(v, fa, fb).$$

This tableau is substitutively inconsistent. However, we cannot derive the empty tableau from it without the use of tableau function reflexivity.

Plaisted [129] described a tableau system which is complete, does not use function reflexivity and has the termination property for the subsystem without tableau expansion rules but uses a new tableau factoring rule. Degtyarev and Voronkov [58] described a complete system obtained from the $\mathbf{TK}^{=}$ by dropping tableau function reflexivity and imposing several restrictions on paramodulation. The restrictions ensure the termination property. In particular, results of Degtyarev and Voronkov [58] imply that $\mathbf{TK}^{=}$ is complete without function reflexivity. These results are described below.

An attempt to improve Fitting's tableau paramodulation was made in Beckert and Hähnle [22]. In that paper a restricted form of paramodulation is introduced in which "it is not necessary to apply equalities to other equalities" and function reflexivity is not used. This method is nonterminating for a given tableau expansion. In addition, despite the claim of the completeness, the method is incomplete. For example, as noted in Degtyarev and Voronkov [58], the method cannot prove the formula $\exists x(a \approx b \wedge g(fa, fb) \approx h(fa, fb) \supset g(x,x) \approx h(x,x))$.

7.2 Tableau basic superposition

Here we give a calculus introduced in Degtyarev and Voronkov [58]. This calculus imposes the following restrictions on paramodulation:

1. no function reflexivity is needed;
2. paramodulation into variables is not allowed;
3. orderings are used so that there are no increasing applications of paramodulation;
4. basic restriction on paramodulation allows one to prohibit paramodulation into nonvariable terms introduced by unification.

We shall introduce the logical calculus **TBSE** (Tableau Basic Superposition with Equality Solution) which is an adaptation of the calculus **BSE** of Section 6.2 to tableaux. The provable objects of **BSE** are *constrained tableaux*.

- ▶ A *constrained tableau* is a pair consisting of a tableau $\mathcal{T}$ and a constraint $\mathcal{C}$. Such a constrained tableau will be denoted $\mathcal{T} \cdot \mathcal{C}$.

In this section we display constraints as sets of atomic equality and inequality constraints, instead of a conjunction.

In this section we assume that all rules of **TK** are modified for constraint tableaux such that the tableau expansion rules do not change the constraint. For example, one of the β-rules is changed to

$$\frac{\Gamma_1, T\ \varphi \vee \psi \mid \ldots \mid \Gamma_n \cdot \mathcal{C}}{\Gamma_1, T\ \varphi \mid \Gamma_1, T\ \psi \mid \ldots \mid \Gamma_n \cdot \mathcal{C}}\ (\beta),$$

and the rule (abc) is changed to

$$\frac{\Gamma_1, T\ A, F\ B \mid \Gamma_2 \mid \ldots \mid \Gamma_n \cdot \mathcal{C}}{\Gamma_2 \mid \ldots \mid \Gamma_n \cdot \mathcal{C} \cup \{A = B\}}\ (abc),$$

The calculus **TBSE** consists of the following inference rules.

- ▶ *Tableau basic superposition* is the following inference rule:

$$\frac{\Gamma_1, T\ s \approx t, X\ u[s'] \approx v \mid \ldots \mid \Gamma_n \cdot \mathcal{C}}{\Gamma_1, T\ s \approx t, X\ u[t] \approx v \mid \ldots \mid \Gamma_n \cdot \mathcal{C} \cup \{s \succ t, u[s'] \succ v, s = s'\}}\ (tbs),$$

  where

  1. The constraint at the conclusion of the rule is satisfiable;
  2. The signed formula $X\ u[t] \approx v$ does not occur in Γ_1;
  3. The term s' is not a variable.

- ▶ *Tableau equality solution* is the following inference rule:

$$\frac{\Gamma_1, F\ s \approx t \mid \ldots \mid \Gamma_n \cdot \mathcal{C}}{\Gamma_2 \mid \ldots \mid \Gamma_n \cdot \mathcal{C} \cup \{s = t\}}\ (teqs)$$

  where $\mathcal{C} \cup \{s = t\}$ is satisfiable.

▶ The *calculus* **TBSE** is the logical calculus whose inference rules are (tbs), ($teqs$) and all rules of **TK**.

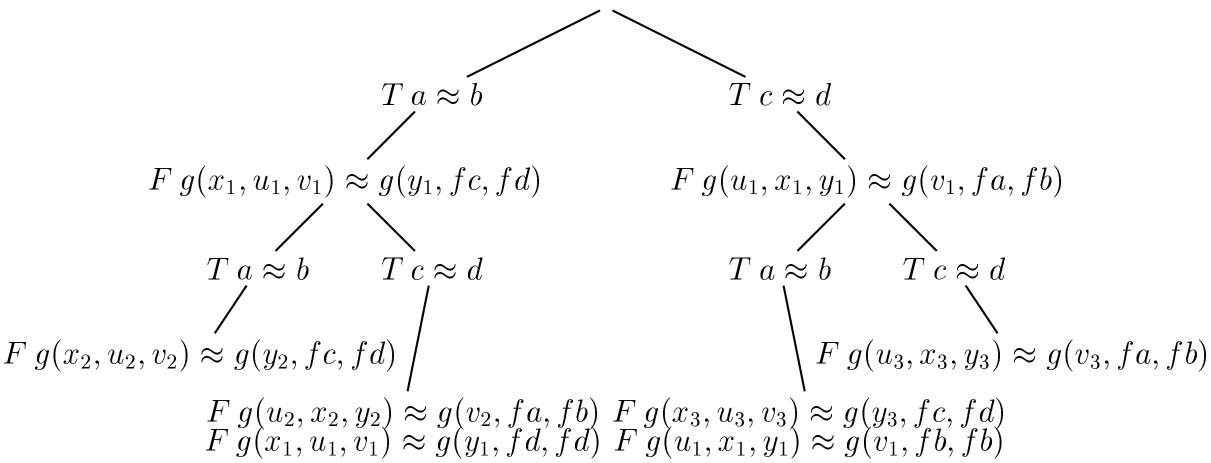
Theorem 7.4 (Soundness and completeness of TBSE) Let ξ be a formula. It is provable in first-order logic with equality if and only if there is a derivation from the constrained tableau $F\,\xi \cdot \emptyset$ of a constraint tableau of the form $\# \cdot \mathcal{C}$.

This logical system has a pleasant termination property:

Theorem 7.5 (Termination of TBSE) For every constrained tableau $\mathcal{T} \cdot \mathcal{C}$ there is only a finite number of derivations from $\mathcal{T} \cdot \mathcal{C}$ not using tableau expansion rules.

This means that for a given amplification we cannot have infinite search. Infinite search without expansion steps is possible in Fitting's system.

To illustrate the connection between the tableau basic superposition rules and rules of **TBSE**, we reconsider the tableau of Example 3.3. On the branch containing the literal $g(x_1, u_1, v_1) \not\approx g(y_1, fc, fd)$ and the equation $c \approx d$, we can apply rigid basic superposition that adds $g(x_1, u_1, v_1) \not\approx g(y_1, fd, fd)$ to the branch. Similarly, we can apply rigid basic superposition to the branch containing $g(u_1, x_1, y_1) \not\approx g(v_1, fa, fb)$ and $a \approx b$, obtaining $g(u_1, x_1, y_1) \not\approx g(v_1, fb, fb)$. This results in the following tableau displayed in the tree-like form (we only show the atomic formulas in the tableau):



After four application of the tableau equality solution rules all branches of this tableau become closed. The resulting constraint of this derivation is the same as the union of the answer constraints shown at the end of Section 6.3.

7.3 Rigid variables in resolution theorem proving

In Section 3.1 we have defined the notion of substitutive inconsistency for branches and tableaux. Originally, this notion has been introduced by Chang [34] and Chang and Lee [95] for sets of clauses.

- ▶ A finite set or a multiset of clauses S is called *substitutively inconsistent* if there is a substitution σ such that the formula $\bigwedge_{C \in S} C\sigma$ is inconsistent.

Later, Plaisted [129] called substitutive inconsistency rigid Eq-unsatisfiability. For establishing substitutive inconsistency, Chang and Lee used so-called V-resolution [34] with V-paramodulation [95], where V stands for "variable-constrained". Following Becher and Petermann [17], instead of "variable-constrained" we shall use more accepted words "rigid resolution" and "rigid paramodulation" for V-resolution and V-paramodulation.

Rigid resolution (respectively, paramodulation) is a "rigid" variant of resolution (respectively, paramodulation). In these inference rules, all variables are treated "rigidly", as unknown parameters while in the ordinary resolution and paramodulation rules variables are treated as universally quantified. Hence, similar to tableaux, we shall formulate these inference rules as working on multisets of clauses. As usual, for a set or a multiset of clauses S and a clause C we shall write S, C instead of $S \cup \{C\}$.

- ▶ The *calculus* **RRP** consists of the following four rules:
  - ▶ *rigid resolution rule*:

$$\frac{S, A \vee C_1, \neg A' \vee C_2}{(S, A \vee C_1, \neg A' \vee C_2, C_1 \vee C_2)\mathrm{mgu}(A, A')} \ (\mathit{rres});$$

  - ▶ *rigid factoring rule*:

$$\frac{S, L \vee L' \vee C}{(S, L \vee C)\mathrm{mgu}(L, L')} \ (\mathit{rfac});$$

▶ *rigid reflexivity rule*:

$$\frac{S, s \not\approx t \vee C}{(S, C)\mathrm{mgu}(s,t)}\ (\mathit{rrefl});$$

▶ *rigid paramodulation rule*:

$$\frac{S, L[s'] \vee C_1, s \approx t \vee C_2}{(S, L[s'] \vee C_1, s \approx t \vee C_2, L[t] \vee C_1 \vee C_2)\mathrm{mgu}(s,s')}\ (\mathit{rpar}).$$

The goal is to derive a multiset of clauses containing the empty clause $\Box$. The calculus **RRP** is complete for checking substitutive inconsistency in logic without equality.[7] For logic with equality, completeness of **RRP** for checking substitutive inconsistency can be proved with the additional function reflexivity rule, similar to the (*tfr*) rule for tableaux:

$$\frac{S}{S, f(x_1, \ldots, x_n) \approx f(x_1, \ldots, x_n)}\ (\mathit{rfr}),$$

where f is a function symbol occurring in S of arity $n > 0$ and $x_1, \ldots, x_n$ are new variables. Instead of this rule we can use

$$\frac{S}{S[x \mapsto f(x_1, \ldots, x_n)]}.$$

We have

Theorem 7.6 Let S be a finite multiset of clauses. Then the following conditions are equivalent:

1. S is substitutively inconsistent;
2. there is a multiset of clauses S' such that S' is derivable from S in **RRP** + (*rfr*) and $\Box \in S'$.

Following [58] we demonstrate that **RRP** is incomplete for checking substitutive inconsistency. This example is a translation of Example 7.3 to

[7] Chang and Lee [95] assert without proof a theorem which implies completeness of **RRP** for logic with equality which is not true as we show later.

clausal form. In fact, it encodes substitutive inconsistency of the tableau from that example via a multiset of clauses.

$$
\begin{array}{l}
a \approx b \vee c \approx d \\
a \approx b \vee g(u, x, y) \not\approx g(v, fa, fb) \\
c \approx d \vee g(x, u, v) \not\approx g(y, fc, fd) \\
g(x, u, v) \not\approx g(y, fc, fd) \vee g(u, x, y) \not\approx g(v, fa, fb)
\end{array}
$$

This multiset of clauses is substitutively inconsistent. The only substitution making this multiset inconsistent is $\theta \approx [x \mapsto fa, y \mapsto fb, u \mapsto fc, v \mapsto fd]$. The reader can check that any application of an inference rule of **RRP** to this multiset of clauses gives a substitution incompatible with θ.

In the calculus **RRP** the rigid paramodulation rule is unrestricted: there are no ordering restrictions and paramodulation into variables is allowed[8].

7.4 From resolution to tableaux and matings

Several papers compared resolution with connections (matings), for example Bibel [28], Eder [62, 63], Mints [116], Baumgartner and Furbach [15] and Avron [4]. In particular, Eder [62] has shown "how a resolution refutation can be transformed to a complementary connected matrix, thus yielding a connection derivation of a given matrix". Under this translation, the connection calculus had to be augmented with a factoring (factorization) rule: "In cases where resolution steps are combined with factorization steps we have to introduce factorization links in addition to the connections".

Plaisted [129] uses a similar technique to translate "rigid resolution and paramodulation" into the connection proofs using the following idea. Let S be a multiset of clauses. Plaisted calls an *amplification* of S a multiset T of one or more variants of clauses in S with variables renamed so that each variable appears in at most one clause in T. (The same notion has been used in V-resolution of Chang and Lee [95] by the name of "alleged substitutively unsatisfiable set".) For large enough T, substitutive inconsistency of T is equivalent to the inconsistency of S. Then the obtained rigid refutation is translated into so-called *path paramodulation* over the multiset of paths through T. This technique is, in fact, identical to the technique used by

[8]Plaisted [129] shows incompleteness of ordered rigid paramodulation and notes that this incompleteness result may be not true for unrestricted rigid paramodulation. Thus, this example improves the result of Plaisted [129].

Eder [63]. For example, Plaisted's "path correspondence" was used by Eder as "shortening of a path". Final results of Plaisted are also similar to the results of Eder: the tableau system of Plaisted contains in addition to the standard tableau rules a factoring rule.

As a consequence, this method does not allow one to close the tableau "branch-wise" as it is done in the procedure of Gallier et.al. or in the calculus **BSE** of Section 6. In other words, path paramodulation of Plaisted does not allow branch-wise computation of solutions to rigid E-unification for separate branches. Nevertheless, this technique gives some substitute for tableau basic superposition described here, because path paramodulation terminates for a given tableau expansion. Unlike tableau basic superposition, the termination of path paramodulation is due to the fact that all literals involved in paramodulation are deleted from their path.

8 Equality elimination

In all sequent-based free-variable methods, proof-search comprises two kinds of operations. Operations (or inferences rules) of the first kind construct a tree of sequents, matrix or tableau using suitable *expansion rules*. The second kind of operations (not always formalized as inference rules) tries to close leaf sequents in the tree, paths in the matrix or branches in the tableau using suitable substitutions.

Despite differences among these methods, they have a common disadvantage: "globality" of the second kind of operations. Maslov, Mints and Orevkov [112] characterized such methods (tableau, matings and model elimination) as *global*, and methods like resolution or the inverse method as *local*. We quote [112]:

> ... Following the idea it is not the proof we get at first but some intermediate object — prededuction; after that we verify the possibility of turning the sufficiently complicated predeductions into correct deductions (and this amount of work is generally speaking useless for the further steps if the given prededuction cannot still be turned into a correct deduction and have to be built over). This is connected first of all with the "globality" of work with the predeductions. That is why more perspective are apparently the methods enabling us to combine the unpreciseness of the values of variables and "local independence" of work (i.e. the using of a relatively small part of deduction at the given stage of work and independence of this work from the remaining parts of the deduction).

Consider several examples. In the MGU replacement rule of Fitting [64] the substitution produced by unification is applied to the whole tableau. In equational matings (Gallier et.al. [73, 69]) the substitution is global for all paths in a matrix. In tableau basic superposition of Section 7.2 the constraint is global for the whole tableau and every application of rigid basic superposition on any branch changes the constraint and thus influences all other branches.

In this chapter we describe the *equality elimination method* which allows us to localize equality reasoning so that we obtain a partially local sequent-based method (for tableaux or matings) or a completely local sequent-based method (for the inverse method). This method is based on extending sequent-based provers by a bottom-up equation solver using basic superposition. Solutions to equations are generated by this solver and used to close branches of a tableau, paths of a matrix or to generate axioms for the inverse method. The equation solution is even more restricted by the use of orderings, basic simplification and subsumption.

The equality elimination method was originally introduced in Degtyarev and Voronkov [50] as a method of handling equality in logic programs. Later, it has been applied to the inverse method in Degtyarev and Voronkov [48] and to matings and semantic tableaux in Degtyarev and Voronkov [49, 54]. Applications of equality elimination to the tableau method and to the inverse method are based on a common characterization of provability in terms of solution clauses described in various forms in the rest of this section. This characterization can be considered as a computationally improved version of the Herbrand theorem. However, the resulting calculi for the inverse method and for the tableau method are quite different because the inverse method is local while the tableau method is global.

8.1 Theoretical basis for equality elimination

For the rest of this section we assume that ξ denotes a closed formula in the skolem negation normal form to be proved (the "goal"). We assume that all different occurrences of quantifiers in ξ bind different variables. For example, ξ cannot have the form $\exists x A \vee \exists x B$. We shall identify subformulas of ξ and their superformulas with their *occurrences* in ξ. For example, in the formula ξ of the form $A \wedge (A \vee B)$ the second occurrence of A is considered a subformula of the occurrence of $(A \vee B)$, but the first occurrence of A is not.

In order to define a more efficient sequent-based calculus, we shall only deal with some subformulas of ξ called conjunctive subformulas.

- ▶ An (occurrence of a) subformula φ of ξ is called *conjunctive* if it is an occurrence in a subformula $\varphi \wedge \psi$ or in $\psi \wedge \varphi$.

In fact, conjunctive subformulas of ξ are equivalent to β-subformulas.

- ▶ We call a formula φ a *superformula* of a formula ψ if ψ is a subformula of φ.

- ▶ Let φ be a subformula of ξ. A *conjunctive superformula* of φ in ξ is any superformula ψ of φ that is a conjunctive subformula of ξ.

- ▶ The *least conjunctive superformula* of φ in ξ is the conjunctive superformula ψ of φ in ξ such that any other conjunctive superformula of φ is a superformula of ψ.

Note that any formula having a conjunctive superformula has the unique least conjunctive superformula. We also note that if φ is a formula and ψ is its least conjunctive superformula, then $\varphi \vdash \psi$. This partially explains the need for introducing least conjunctive superformulas. There are deterministic chains of inferences in sequent systems consisting of α- and γ-rules. For example,

$$\cfrac{\cfrac{\Gamma, F\,\varphi}{\Gamma, F\,\varphi \vee \psi}\,(\alpha)}{\Gamma, F\,\exists x(\varphi \vee \psi)}\,(\gamma)$$

By restricting ourselves to conjunctive superformulas only, we eliminate these deterministic chains, making them in one step[9].

Since we fix the goal formula ξ, we shall speak about the least conjunctive superformula of φ meaning the least conjunctive superformula of φ in ξ. Least conjunctive superformulas are illustrated in Table 1 (ignore for a moment the third column of the table).

We can enumerate all conjunctive subformulas $\xi_1, \ldots, \xi_n$ of ξ, for example in the order of their occurrences in ξ. Thus we can unambiguously use "the kth conjunctive (sub)formula" ξ_k of ξ.

Let $\mathcal{A}_1, \ldots, \mathcal{A}_n$ be predicate symbols not occurring in ξ.

- ▶ We say that the atomic formula $\mathcal{A}_k(x_1, \ldots, x_m)$ is the *ξ-name of a subformula* φ of ξ if

  1. the least conjunctive superformula of φ is ξ_k;

[9]This simple but powerful idea of restricting to conjunctive superformulas (see Voronkov [155]) has been described in the form of a sequent calculus in Voronkov [154] and implemented in the theorem prover described in Voronkov [153] and in a theorem prover for intuitionistic logic implemented by Tammet [147]. In the framework of tableau theorem proving, several papers used permutabilities of inference rules in sequent calculi (see e.g. Shankar [141]). In fact, the least conjunctive superformula of φ is a superformula ψ of φ provable from φ and such that all inference rules applied in the proof of ψ from φ are permutable with all other rules. The use of conjunctive superformulas allows us to get rid of non-conjunctive subformulas before the proof-search, unlike the dynamic use of permutabilities as in Shankar [141]. The use of conjunctive superformulas also covers all applications of *universal formulas* in the tableau frameworks proposed in Beckert and R. Hähnle [22].

Subformula	least conjunctive superformula	set of ξ-names
$(\exists x(F(x) \wedge (B(x) \vee \exists y C(x,y))) \wedge \exists z D(z)) \vee E$	no	$\emptyset$
$\exists x(F(x) \wedge (B(x) \vee \exists y C(x,y))) \wedge \exists z D(z)$	no	$\emptyset$
$\exists x(F(x) \wedge (B(x) \vee \exists y C(x,y)))$	$\exists x(F(x) \wedge (B(x) \vee \exists y C(x,y)))$	$\{\mathcal{A}_1\}$
$F(x) \wedge (B(x) \vee \exists y C(x,y))$	$\exists x(F(x) \wedge (B(x) \vee \exists y C(x,y)))$	$\{\mathcal{A}_1\}$
$F(x)$	$F(x)$	$\{\mathcal{A}_2(x)\}$
$B(x) \vee \exists y C(x,y)$	$B(x) \vee \exists y C(x,y)$	$\{\mathcal{A}_3(x)\}$
$B(x)$	$B(x) \vee \exists y C(x,y)$	$\{\mathcal{A}_3(x)\}$
$\exists y C(x,y)$	$B(x) \vee \exists y C(x,y)$	$\{\mathcal{A}_3(x)\}$
$C(x,y)$	$B(x) \vee \exists y C(x,y)$	$\{\mathcal{A}_3(x)\}$
$\exists z D(z)$	$\exists z D(z)$	$\{\mathcal{A}_4\}$
$D(z)$	$\exists z D(z)$	$\{\mathcal{A}_4\}$
E	no	$\emptyset$

Table 1: Least conjunctive superformulas and sets of ξ-names of subformulas of the formula $\xi = (\exists x(F(x) \wedge (B(x) \vee \exists y C(x,y))) \wedge \exists z D(z)) \vee E$

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

If a ξ-name of a formula φ exists, then it is unique. Note that different formulas may have the same ξ-names. Also note that some subformulas of ξ do not have ξ-names (those who do not have a conjunctive superformula). We can use the *set of ξ-names* of a subformula. The set of ξ-names of a formula φ is either $\emptyset$ or a singleton $\{\mathcal{A}_k(x_1, \ldots, x_m)\}$. Table 1 illustrates sets of ξ-names.

Consider another example giving us the idea of an algorithm for assigning ξ-names. We shall use this example for the illustration of all methods of this section.

Example 8.1 (ξ-names) Consider the formula of Example 1.3

$$\exists xyuv((a \approx b \supset g(x,u,v) \approx g(y,fc,fd)) \wedge \\ (c \approx d \supset g(u,x,y) \approx g(v,fa,fb))).$$

Its negation normal form is

$$\xi = \exists xyuv((a \not\approx b \vee g(x,u,v) \approx g(y,fc,fd)) \wedge \\ (c \not\approx d \vee g(u,x,y) \approx g(v,fa,fb))).$$

The only conjunctive subformulas of ξ are the subformulas $a \not\approx b \vee g(x,u,v) \approx g(y,fc,fd)$ and $c \not\approx d \vee g(u,x,y) \approx g(v,fa,fb)$. Thus, we can introduce the following ξ-names:

1. $\mathcal{A}_1(x,u,v,y)$ for all subformulas of $a \not\approx b \vee g(x,u,v) \approx g(y,fc,fd)$;
2. $\mathcal{A}_2(u,x,y,v)$ for all subformulas of $c \not\approx d \vee g(u,x,y) \approx g(v,fa,fb)$.

All other formulas have no ξ-names since they have no least conjunctive superformula.

In Section 7 we formalized basic superposition using ordering constraints. Here we shall formalize it via *closures* used e.g. in Bachmair et.al. [13].

► A *closure* is a pair $C \cdot \sigma$, where C is a clause and σ is a substitution.

Such a closure semantically corresponds to the clause $C\sigma$.

- ▶ Two closures $C_1 \cdot \sigma_1$ and $C_2 \cdot \sigma_2$ are called *variants* if C_1 is a variant of C_2 and $C_1\sigma_1$ is a variant of $C_2\sigma_2$.

We shall identify variants.

The equality elimination method is based on transformations of closures which try to eliminate equality from them (hence the name "equality elimination"). The equality elimination procedure begins with the so-called initial closures and uses a basic superposition calculus described below.

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

  1. whenever a literal $s \not\approx t$ occurs in ξ and C is the set of ξ-names of this occurrence of $s \not\approx t$, the closure $s \approx t, C \cdot \varepsilon$ is an initial closure;
  2. whenever the literal $s \approx t$ occurs in ξ and C is the set of ξ-names of this occurrence of $s \approx t$, the closure $s \not\approx t, C \cdot \varepsilon$ is an initial closure;
  3. let literals $P(s_1, \dots, s_n)$ and $\neg P(t_1, \dots, t_n)$ occur in ξ and C_1, C_2 are their sets of ξ-names. Let the substitution ρ rename variables such that variables of $C_1\rho$ and C_2 are disjoint. Then the closure $s_1\rho \not\approx t_1, \dots, s_n\rho \not\approx t_n, C_1\rho, C_2 \cdot \varepsilon$ is an initial closure.

Example 8.2 (Initial closures) Let us continue Example 8.1. The formula ξ of that example has no predicate symbols different from $\approx$. Thus, the last case of the definition of initial closures is not applicable. The first two cases show how to associate an initial closure with every equality literal in ξ. Applying them, we obtain the following four initial closures:

$$\begin{array}{ll}
1 & a \approx b, \mathcal{A}_1(x, u, v, y) \cdot \varepsilon \\
2 & g(x, u, v) \not\approx g(y, fc, fd), \mathcal{A}_1(x, u, v, y) \cdot \varepsilon \\
3 & c \approx d, \mathcal{A}_2(u, x, y, v) \cdot \varepsilon \\
4 & g(u, x, y) \not\approx g(v, fa, fb), \mathcal{A}_2(u, x, y, v) \cdot \varepsilon
\end{array}$$

We introduce several inference rules on closures defining a variant of basic superposition. We assume that premises of rules have disjoint variables which can be achieved by renaming variables.

- ▶ *Basic (right and left) superposition rules* are the following inference rules:

$$\frac{s \approx t, C \cdot \sigma_1 \quad u[s'] \approx v, D \cdot \sigma_2}{u[t] \approx v, C, D \cdot \sigma_1\sigma_2\theta}\ (brs)$$

$$\frac{s \approx t, C \cdot \sigma_1 \quad u[s'] \not\approx v, D \cdot \sigma_2}{u[t] \not\approx v, C, D \cdot \sigma_1\sigma_2\theta}\ (bls)$$

where

1. θ is a most general unifier of $s\sigma_1$ and $s'\sigma_2$;
2. $t\sigma_1\theta \not\succeq s\sigma_1\theta$ and $v\sigma_2\theta \not\succeq u[s']\sigma_2\theta$;
3. s' is not a variable;
4. (for left superposition only) $u[s'] \not\approx v$ is the leftmost disequation in the second premise[10].

- *Equality solution* is the following inference rule:

$$\frac{s \not\approx t, C \cdot \sigma}{C \cdot \sigma \mathrm{mgu}(s\sigma, t\sigma)}\ (eqs)$$

where $s \not\approx t$ is the leftmost disequation in the premise.

- $\mathbf{BS}_\xi$ is the inference system whose axioms are initial closures and whose inference rules are (bls), (brs) and (eqs).

- We call a *solution clause* is any clause $C\sigma$ such that

1. $C \cdot \sigma$ is derivable in $\mathbf{BS}_\xi$;
2. C does not contain equality.

Let $L_1, \dots, L_n$ be a solution clause. By replacing ξ-names to corresponding conjunctive subformulas, we obtain a multiset of formulas $\varphi_1, \dots, \varphi_n$ which we informally call *the formula image of the solution clause.*

Theorem 8.3 The following conditions are equivalent:

1. ξ is provable in first-order logic with equality;

[10] According to our definitions, a clause is a multiset of literals, so the use of the leftmost disequation is not quite correct, but this restriction can easily be formalized using the selection mechanism (Bachmair et.al. [13]).

2. There exist solution clauses $C_1, \ldots, C_m$ such that ξ is provable in the logical system whose axioms are the formula images of $C_1, \ldots, C_m$ and whose inference rules are rules of $\mathbf{IK}_\xi$ except for (Ax).

Thus, after obtaining a suitable set of solution clauses, we can reduce provability in logic with equality to provability in logic without equality and with axioms of a very special form. In subsequent sections, we shall present two reformulations of this theorem which will be directly applicable to the tableau method and to the inverse method.

We illustrate solution clauses by continuing Example 8.2.

Example 8.4 (Solution clauses) Let us continue Example 8.1. The function signature for this example is $\mathcal{F} = \{a, b, c, d, f, g\}$. Consider the lexicographic path ordering $\succ$ induced by the precedence relation $a \succ_{\mathcal{F}} b \succ_{\mathcal{F}} c \succ_{\mathcal{F}} d \succ_{\mathcal{F}} f \succ_{\mathcal{F}} g$. In this ordering, we have $a \succ b$ and $c \succ d$. Repeatedly applying all possible inference rules of $\mathbf{BS}_\xi$ to the closures of Example 8.2, we obtain the following closures. The derivations are presented in the linear format, instead of tree-like derivations. In each line, the right column provides the information about the inference rule applied to obtain the closure at that line and the premises of the application of the inference rule.

$$
\begin{array}{rll}
5. & \mathcal{A}_1(x, u, v, y) \cdot [y \mapsto x, u \mapsto fc, v \mapsto fd] & (eqs),\ 2 \\
6. & \mathcal{A}_2(u, x, y, v) \cdot [v \mapsto u, x \mapsto fa, y \mapsto fb] & (eqs),\ 4 \\
7. & \mathcal{A}_1(x_1, u_1, v_1, y_1), g(u_2, x_2, y_2) \not\approx g(v_2, fb, fb), & \\
 & \quad \mathcal{A}_2(u_2, x_2, y_2, v_2) \cdot \varepsilon & (bls),\ 1, 4 \\
8. & \mathcal{A}_2(u_1, x_1, y_1, v_1), g(x_2, u_2, v_2) \not\approx g(y_2, fd, fd), & \\
 & \quad \mathcal{A}_1(x_2, u_2, v_2, y_2) \cdot \varepsilon & (bls),\ 3, 2 \\
9. & \mathcal{A}_1(x_1, u_1, v_1, y_1), \mathcal{A}_2(u_2, x_2, y_2, v_2) & \\
 & \quad \cdot [v_2 \mapsto u_2, x_2 \mapsto fb, y_2 \mapsto fb] & (eqs),\ 7 \\
10. & \mathcal{A}_2(u_1, x_1, y_1, v_1), \mathcal{A}_1(x_2, u_2, v_2, y_2) & \\
 & \quad \cdot [y_2 \mapsto x_2, u_2 \mapsto fd, v_2 \mapsto fd] & (eqs),\ 8
\end{array}
$$

No further applications of the inference rules of $\mathbf{BS}_\xi$ are possible. Thus, we obtain the following four solutions clauses obtained from closures 5,6,9, and 10, respectively:

$$\begin{array}{rl} 5' & \mathcal{A}_1(x, fc, fd, x) \\ 6' & \mathcal{A}_2(u, fa, fb, u) \\ 9' & \mathcal{A}_1(x_1, u_1, v_1, y_1), \mathcal{A}_2(u_2, fb, fb, u_2) \\ 10' & \mathcal{A}_2(u_1, x_1, y_1, v_1), \mathcal{A}_1(x_2, fd, fd, x_2) \end{array}$$

In general, there may be an infinite number of solution clauses.

8.2 Equality elimination for semantic tableaux

In the previous section we have introduced ξ-names in order to deal with occurrences of subformulas instead of subformulas themselves. In this section, we modify the free-variable tableau calculus to deal with ξ-names instead of formulas. The new tableaux will be called ξ-tableaux.

- ▶ A *ξ-branch* is any finite multiset of atoms of the form $\mathcal{A}_k(t_1, \dots, t_m)$ such that $\mathcal{A}_k(x_1, \dots, x_m)$ is a ξ-name of some subformula of ξ.
- ▶ A *ξ-tableau* is any finite multiset $\{C_1, \dots, C_n\}$ of ξ-branches, denoted by $C_1 \mid \dots \mid C_n$.

Our calculus for ξ-tableaux uses *subset unification*.

- ▶ A substitution σ is called a *subset unifier* of a clause C_1 against a clause C_2 if $C_1\sigma \subseteq C_2\sigma$.
- ▶ A subset unifier of C_1 against C_2 is called a *minimal* if it is minimal w.r.t. $\leq$ among all subset unifiers of C_1 against C_2.

Subset-unification has been earlier used in P-deduction of Demolombe [60] and in SLO-resolution by the name of *θ-subsumption* in Minker, Rajasekar and Lobo [115]. Subset-unifiability is NP-complete [47]. Some other properties of subset unification are discussed in Degtyarev and Voronkov [47].

On ξ-tableaux, we shall introduce a calculus called $\mathbf{T}_\xi$ which consists of two inference rules: tableau expansion and branch closure.

- ▶ Let C_1, C_2 and C be the sets of ξ-names of φ, ψ and $\varphi \wedge \psi$, where $\varphi \wedge \psi$ is a subformula of ξ. Then the following is a *ξ-tableau expansion rule*

$$\frac{D_1 \mid D_2 \mid \dots \mid D_m}{(C_1, D_1 \mid C_2, D_1 \mid D_2 \mid \dots \mid D_m)\sigma} \; (te_\xi)$$

where σ is a minimal subset unifier of C against D_1. We assume that the variables of the premise are disjoint from the variables of C, C_1, C_2 which can be achieved by renaming variables in the tableau.

This rule can be described in a more simple (but less compact) form, when we consider its two cases depending on the number of elements in C. There are two possibilities: C is either $\emptyset$ or a singleton multiset $\{\mathcal{A}_i(\bar{x})\}$. As for C_1 and C_2, they are nonempty since φ and ψ are conjunctive subformulas of $\varphi \wedge \psi$. Thus, $C_1 = \{\mathcal{A}_j(\bar{y})\}$ and $C_2 = \{\mathcal{A}_k(\bar{z})\}$.

1. If $C = \emptyset$, then the rule (te_ξ) becomes

$$\frac{D_1 \mid D_2 \mid \ldots \mid D_m}{D_1, \mathcal{A}_j(\bar{y}) \mid D_1, \mathcal{A}_k(\bar{z}) \mid D_2 \mid \ldots \mid D_m} \; (te_\xi)$$

2. If $C = \{\mathcal{A}_i(\bar{x})\}$, then the rule (te_ξ) becomes

$$\frac{D_1, \mathcal{A}_i(\bar{t}) \mid D_2 \mid \ldots \mid D_m}{D_1, \mathcal{A}_j(\bar{y})\sigma \mid D_1, \mathcal{A}_k(\bar{z})\sigma \mid D_2 \mid \ldots \mid D_m} \; (te_\xi)$$

   where $\sigma = [\bar{x} \mapsto \bar{t}]$.

This rule corresponds to the β-rule

$$\frac{\beta}{\beta_1 \mid \beta_2}$$

of Smullian [144] modified for ξ-branches.

- ▶ Let C be any solution clause. Then the following is a ξ-*branch closure rule*

$$\frac{C_1 \mid C_2 \mid \ldots \mid C_m}{(C_2 \mid \ldots \mid C_m)\sigma} \; (bc_\xi)$$

  where σ is a minimal subset-unifier of C against C_1. We assume that variables of the tableau are disjoint from the variables of C which can be achieved by renaming variables in C.

- ▶ The *calculus* $\mathbf{T}_\xi$ is the calculus whose axiom is the tableau consisting of one branch $\Box$ and whose inference rules are (te_ξ) and (bc_ξ).

Due to the use of least conjunctive superformulas, we do not need any analogues of α- or γ-rules in $\mathbf{T}_\xi$. The ξ-tableau expansion rule is an analogue of the β-rule. The calculus $\mathbf{T}_\xi$ was introduced in Degtyarev and Voronkov [54] in a slightly different form where both closures and ξ-tableaux were derivable objects. In fact, the calculus T_ξ of [54] is the union of $\mathbf{T}_\xi$ of this paper and $\mathbf{BS}_\xi$.

Theorem 8.5 (Equality elimination, ξ-tableau formulation) The following conditions are equivalent:

1. ξ is provable in first-order logic with equality;
2. the empty tableau # is derivable in $\mathbf{T}_\xi$.

This theorem can also be reformulated as a refinement of Theorem 3.1 (Herbrand's theorem for free variable tableaux):

Theorem 8.6 The following conditions are equivalent:

1. ξ is provable in first-order logic with equality;
2. there is a ξ-tableau $\mathcal{T}$ constructed from $\Box$ by applications of the ξ-tableau expansion rule and a substitution θ such that every branch in $\mathcal{T}\theta$ contains an instance of a solution clause.

Theorem 3.1 just asserted that we have to search for a substitution making all branches inconsistent, which leads to simultaneous rigid E-unification. Theorem 8.6 contains a more constructive statement: branches can be closed by instances of solution clauses.

Let us illustrate equality elimination for tableaux.

Example 8.7 Consider again Examples 8.1 and 8.4. For the formula ξ of these examples, we get the following ξ-tableau expansion rule:

$$\frac{D_1 \mid D_2 \mid \ldots \mid D_m}{\mathcal{A}_1(x,u,v,y), D_1 \mid \mathcal{A}_2(u,x,y,v), D_1 \mid D_2 \mid \ldots \mid D_m}\ (te_\xi)$$

where x, y, u, v are variables not occurring in the premise of this tableau.

For the branch closure rules, we shall need solution clauses $5', 6', 9'$ and $10'$ of Example 8.4. The derivation is as follows:

1. $\Box$
2. $\mathcal{A}_1(x_0, u_0, v_0, y_0) \mid \mathcal{A}_2(u_0, x_0, y_0, v_0)$ (te_ξ), 1
3. $\mathcal{A}_1(x_0, u_0, v_0, y_0), \mathcal{A}_1(x_1, u_1, v_1, y_1) \mid$ (te_ξ), 2
   $\mathcal{A}_1(x_0, u_0, v_0, y_0), \mathcal{A}_2(u_1, x_1, y_1, v_1) \mid \mathcal{A}_2(u_0, x_0, y_0, v_0)$
4. $\mathcal{A}_1(x_0, u_0, v_0, y_0), \mathcal{A}_2(fc, x_1, x_1, fd) \mid \mathcal{A}_2(u_0, x_0, y_0, v_0)$ (bc_ξ), 3
5. $\mathcal{A}_2(fd, u_0, u_0, fd)$ (bc_ξ), 4
6. $\mathcal{A}_2(fd, u_0, u_0, fd), \mathcal{A}_1(x_2, u_2, v_2, y_2) \mid$
   $\mathcal{A}_2(fd, u_0, u_0, fd), \mathcal{A}_2(u_2, x_2, y_2, v_2)$ (te_ξ), 5
7. $\mathcal{A}_2(fd, fb, fb, fd), \mathcal{A}_2(u_2, x_2, y_2, v_2)$ (bc_ξ), 6
8. # (bc_ξ), 7

Here the branch closure rules used to obtain tableaux $4, 5, 7, 8$ have been applied using solution clauses $5', 10', 9', 6'$, respectively.

A similar idea: combination of proof-search in tableaux and a bottom-up equality saturation of the original formula, is used in Moser, Lynch, and Steinbach [118] for constructing a model elimination tableau with refined paramodulation. The main difference between the two approaches is that the elimination of equality in the tableau part in [118] is partial but not total. This requires to introduce in the tableau part *relaxed paramodulation* of Snyder and Lynch [145] and even come back to *lazy paramodulation* (Gallier and Snyder [74]) dropping *top unification* from relaxed paramodulation. Comparing the two forms of paramodulation Snyder and Lynch [145] write: "Our original formulation of RPC (relaxed paramodulation calculus) used the form of lazy paramodulation from [74], but clearly the refinement to top unification is superior, since it restricts the number of inferences and conceptually it clarifies the dividing line between unification and completely lazy unification".

8.3 Equality elimination for the inverse method

Equality elimination for the inverse method is based on the same characterization of provability in terms of solution clauses as that for the tableau method (Theorem 8.6). However, the resulting calculi for the inverse method and for the tableau method are quite different because the inverse method is local while the tableau method is global. Among other inference rules, the inverse method uses the *factoring rule* that is absent in the tableau method. In a certain sense, equality elimination for tableaux and the inverse method are dual to each other which reflects the duality between bottom-up and top-down reasoning.

We shall formulate the inverse method in the form of a logical calculi $\mathbf{I}_\xi$ over *ξ-clauses.*

- ► A *ξ-clause* is any finite multiset of atoms of the form $\mathcal{A}_k(t_1, \ldots, t_m)$ such that $\mathcal{A}_k(x_1, \ldots, x_m)$ is a ξ-name of some subformula of ξ.

The definition of ξ-clauses is the same as that of ξ-branches. We use ξ-clauses to stress, first, that they do not belong to particular tableaux and second, that the calculus of ξ-clauses is similar to the resolution calculus. The calculus $\mathbf{I}_\xi$ consists of two inference rules: the conjunction rule and the factoring rule.

- ► Let C_1, C_2 and C be the sets of ξ-names of φ, ψ and $\varphi \wedge \psi$, where $\varphi \wedge \psi$ is a subformula of ξ and D_1, D_2 be clauses. Let $\bar{x}$ be all variables of $\varphi \wedge \psi$. Then the following is a *ξ-conjunction rule*

$$\frac{D_1, C_1[\bar{x} \mapsto \bar{s}] \quad D_2, C_2[\bar{x} \mapsto \bar{t}]}{(D_1, D_2, C[\bar{x} \mapsto \bar{s}])\mathrm{mgu}(\bar{s}, \bar{t})}\ (\wedge_\xi)$$

- ► The following is a *factoring rule*

$$\frac{C, A, B}{(C, A)\mathrm{mgu}(A, B)}\ (fac)$$

- ► The *calculus* $\mathbf{I}_\xi$ is the calculus whose axioms are solution clauses and whose inference rules are $(\wedge_\xi)$ and (fac).

Theorem 8.8 The following conditions are equivalent:

1. ξ is provable in first-order logic with equality;
2. the empty clause $\Box$ is derivable in $\mathbf{I}_\xi$.

Example 8.9 Consider again Examples 8.1 and 8.4. For the formula ξ of these examples, we get the following conjunction rule:

$$\frac{D_1, \mathcal{A}_1(s_x, s_u, s_v, s_y) \quad D_2, \mathcal{A}_2(t_u, t_x, t_y, t_v)}{(D_1, D_2)\mathrm{mgu}(\langle s_x, s_y, s_u, s_v\rangle, \langle t_x, t_y, t_u, t_v\rangle)}\ (\wedge_\xi)$$

We start with solution clauses $5', 6', 9'$ and $10'$ of Example 8.4. The derivation is as follows:

$$\begin{array}{rll}
5'. & \mathcal{A}_1(x, fc, fd, x) & \\
6'. & \mathcal{A}_2(u, fa, fb, u) & \\
9'. & \mathcal{A}_1(x_1, u_1, v_1, y_1), \mathcal{A}_2(u_2, fb, fb, u_2) & \\
10'. & \mathcal{A}_2(u_3, x_3, y_3, v_3), \mathcal{A}_1(x_4, fd, fd, x_4) & \\
1. & \mathcal{A}_1(x_5, u_5, v_5, y_5), \mathcal{A}_2(u_6, x_6, y_6, v_6) & (\wedge_\xi),\ 9', 10' \\
2. & \mathcal{A}_1(x_7, u_7, v_7, y_7) & (\wedge_\xi),\ 5', 1 \\
3. & \Box & (\wedge_\xi),\ 6', 2
\end{array}$$

Note that the same derivation is obtained by positive hyperresolution, when we replace the ξ-conjunction rule of this example by the clause

$$\neg\mathcal{A}_1(x, u, v, y), \neg\mathcal{A}_2(u, x, y, v).$$

Deduction in the calculus $\mathbf{BS}_\xi$ can be improved by redundancy deletion. Besides tautology deletion, we can use such deletion strategies as *basic subsumption*, *basic simplification*, and *basic blocking* (Bachmair et.al. [12, 13]).

Application of arbitrary simplification in the basic setting may yield incompleteness, as we can see from the following example taken from Degtyarev [40]. Let S consists of 3 closures

$$\begin{array}{l}
1.\ f_1x \approx f_2x \cdot \varepsilon \\
2.\ f_3c \approx c \cdot \varepsilon \\
3.\ f_1f_3y \not\approx f_2c \cdot \varepsilon.
\end{array}$$

We use any reduction ordering such that $f_1x \succ f_2x$. This set of closures is inconsistent, and there is a derivation of the empty closure from 1–3 by basic superposition. We can simplify closure 3 by 1 obtaining

$$4.\ f_2x \not\approx f_2c \cdot [x \mapsto f_3y].$$

It is easy to see that it is impossible to derive the empty closure from $1, 2, 4$ by basic supeprosition.

If we use basic simplification instead, the simplification would give the closure

$$4.\ f_2f_3y \not\approx f_2c \cdot \varepsilon.$$

The applications of equality elimination for logic programs (Degtyarev and Voronkov [50]), to the inverse method (Degtyarev and Voronkov [48]) and to semantic tableaux (Degtyarev and Voronkov [49, 54]) can be considered as a transformation of a set of clauses with equality into a set of clauses without equality. However, all these applications share the same problem: the transformation itself and the resulting set of clauses without equality can, in general, be infinite (since there may be an infinite number of solution clauses). In order to cope with problem, in Degtyarev, Koval and Voronkov [44] and Degtyarev and Voronkov [55] we introduce a new inference rule: the so-called *basic folding* for logic programs with equality. By introduction of new predicate symbols encoding partially solved equations we ensure termination of the equality elimination procedure, thus allowing one to obtain elegant finite programs without equality from equational logic programs. Moreover, the resulting program without equality can sometimes give a unique solution from an infinite number of solutions to the original program. The basic folding is a promising approach in the area of machine learning because it gives us a possibility to automatically discover new notions and to reformulate a logical theory in terms of the new notions.

The essential difference between basic folding and the Brand's modification method [33] is that we try to *solve* all equations in the bodies by specialized applications of equality rules: basic superposition and equality solution. It allows us to automatically "subsume" the axioms of symmetry and transitivity, and also to essentially decrease the number of logical consequences due to the use of orderings. According to the Brand's method, these axioms must be explicitly applied to all positive occurrences of the predicate $\approx$. In [55] we show that basic folding may give a considerably simpler set of clauses: in an example of that paper the basic folding generates a set of clauses having a finite SLD-tree, while Brand's transformation generates a logic program with an infinite SLD-tree. However, the basic folding is currently only defined for sets of Horn clauses.

9 Equality reasoning in nonclassical logics

Equality handling in nonclassical logics is more problematic than in the classical one. Whereas problems in handling equality in tableaux for classical logic have been caused by the presence of rigid variables and do not appear in resolution-based theorem proving, the picture for nonclassical logics is entirely different. In this section we demonstrate that handling equality in intuitionistic logic necessarily requires the use of simultaneous rigid E-unification. We also demonstrate that the same is true for some formalizations of modal logics with equality.

9.1 Intuitionistic logic with equality

The material of this section is mainly based on Voronkov [159, 156].

For technical reasons, in this section we change the definition of a sequent.

► A *sequent* is an expression $\Gamma \to \Delta$, where Γ, Δ are multisets of formulas and Δ contains at most one formula. Γ (respectively Δ) is called the *antecedent* (respectively, the *succedent*) of this sequent.

The condition on the succedent to contain one formula is essential and leads to intuitionistic logic instead of classical.

Thus obtained *sequent calculus* $\mathbf{LJ}^=$ for intuitionistic logic with equality is shown in Figure 12. This calculus derives intuitionistic sequents. In fact, the rules of $\mathbf{LJ}^=$ are precisely the rules of $\mathbf{LK}^=$ adapted to meet the new definition of sequents.

This paper is not intended to serve as an introduction to intuitionistic logic, so we shall only assert some most essential properties of $\mathbf{LJ}^=$ needed to understand the problems arising in handling equality. *Regular derivations* in $\mathbf{LJ}^=$ are defined in the same way as for $\mathbf{LK}^=$ (page 17).

As in the case with $\mathbf{LK}^=$, regular derivations are enough:

Theorem 9.1 Any sequent derivable in $\mathbf{LJ}^=$ has a regular derivation.

Two examples of derivations in $\mathbf{LJ}^=$ of the formula

$$\forall x(x \approx a \vee x \approx b \supset \exists y(f(x,y) \approx b)) \supset \exists u \exists v \exists w(f(f(u,v),w) \approx b)$$

$$\frac{}{\Gamma, A, \Delta \rightarrow A}\ (Ax) \qquad \frac{}{\Gamma \rightarrow t{\approx}t}\ (\mathit{refl})$$

$$\frac{\Gamma[x \mapsto t], s{\approx}t \rightarrow \Delta[x \mapsto t]}{\Gamma[x \mapsto s], s{\approx}t \rightarrow \Delta[x \mapsto s]}\ (\approx)$$

$$\frac{\Gamma, \varphi, \psi, \Xi \rightarrow \Delta}{\Gamma, \varphi \wedge \psi, \Xi \rightarrow \Delta}\ (\wedge \rightarrow) \qquad \frac{\Gamma \rightarrow \varphi \quad \Gamma \rightarrow \psi}{\Gamma \rightarrow \varphi \wedge \psi}\ (\rightarrow \wedge)$$

$$\frac{\Gamma, \varphi, \Xi \rightarrow \Delta \quad \Gamma, \psi \rightarrow \Delta}{\Gamma, \varphi \vee \psi, \Xi \rightarrow \Delta}\ (\vee \rightarrow)$$

$$\frac{\Gamma \rightarrow \varphi}{\Gamma \rightarrow \varphi \vee \psi}\ (\rightarrow \vee_1) \qquad \frac{\Gamma \rightarrow \psi}{\Gamma \rightarrow \varphi \vee \psi}\ (\rightarrow \vee_2)$$

$$\frac{\Gamma, \psi, \Xi \rightarrow \Delta \quad \Gamma, \varphi \supset \psi, \Xi \rightarrow \varphi}{\Gamma, \varphi \supset \psi, \Xi \rightarrow \Delta}\ (\supset\rightarrow) \qquad \frac{\varphi, \Gamma \rightarrow \psi}{\Gamma \rightarrow \varphi \supset \psi}\ (\rightarrow\supset)$$

$$\frac{\Gamma, \neg\varphi \rightarrow \varphi}{\Gamma, \neg\varphi, \Xi \rightarrow}\ (\neg \rightarrow) \qquad \frac{\varphi, \Gamma \rightarrow}{\Gamma \rightarrow \neg\varphi}\ (\rightarrow \neg)$$

$$\frac{\Gamma, \varphi[x \mapsto t], \forall x\varphi, \Xi \rightarrow \Delta}{\Gamma, \forall x\varphi, \Xi \rightarrow \Delta}\ (\forall \rightarrow) \qquad \frac{\Gamma \rightarrow \varphi[x \mapsto y]}{\Gamma \rightarrow \forall x\varphi}\ (\rightarrow \forall)$$

$$\frac{\Gamma, \varphi[x \mapsto y] \rightarrow \Delta}{\Gamma, \exists x\varphi \rightarrow \Delta}\ (\exists \rightarrow) \qquad \frac{\Gamma \rightarrow \varphi[x \mapsto t]}{\Gamma \rightarrow \exists x\varphi}\ (\rightarrow \exists)$$

The rules $(\rightarrow \forall)$ and $(\exists \rightarrow)$ satisfy the standard eigenvariable conditions.

Figure 12: Calculus $\mathbf{LJ}^{=}$

In this example, we denote by φ the formula $\forall x(x\approx a \vee x\approx b \supset \exists y(f(x,y)\approx b))$. We also denote by $\cdots$ some irrelevant parts of the sequents.

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{f(a,y)\approx b, f(b,z)\approx b, \varphi \to f(b,z)\approx f(b,z)}\,(\mathit{refl})}{f(a,y)\approx b, f(b,z)\approx b, \varphi \to f(b,z)\approx b}\,(\approx)}{f(a,y)\approx b, f(b,z)\approx b, \varphi \to f(f(a,y),z)\approx b}\,(\approx)}{f(a,y)\approx b, f(b,z)\approx b, \varphi \to \exists w(f(f(a,y),w)\approx b)}\,(\to\exists)}{f(a,y)\approx b, \exists y(f(b,y)\approx b), \varphi \to \exists w(f(f(a,y),w)\approx b)}\,(\exists\to) \qquad \dfrac{\dfrac{}{\cdots \to b\approx b}\,(\mathit{refl})}{\cdots \to b\approx a \vee b\approx b}\,(\to\vee_2)}{\dfrac{f(a,y)\approx b, b\approx a \vee b\approx b \supset \exists y(f(b,y)\approx b), \varphi \to \exists w(f(f(a,y),w)\approx b)}{f(a,y)\approx b, \varphi \to \exists w(f(f(a,y),w)\approx b)}\,(\forall\to)}\,(\supset\to)
$$

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{f(a,y)\approx b, \varphi \to \exists v\exists w(f(f(a,v),w)\approx b)}\,(\to\exists)}{\exists y(f(a,y)\approx b), \varphi \to \exists v\exists w(f(f(a,v),w)\approx b)}\,(\exists\to) \qquad \dfrac{\dfrac{}{\cdots \to a\approx a}\,(\mathit{refl})}{\cdots \to a\approx a \vee a\approx b}\,(\to\vee_1)}{a\approx a \vee a\approx b \supset \exists y(f(a,y)\approx b), \varphi \to \exists v\exists w(f(f(a,v),w)\approx b)}\,(\supset\to)}{\dfrac{\varphi \to \exists v\exists w(f(f(a,v),w)\approx b)}{\dfrac{\varphi \to \exists u\exists v\exists w(f(f(u,v),w)\approx b)}{\to \varphi \supset \exists u\exists v\exists w(f(f(u,v),w)\approx b)}\,(\to\supset)}\,(\to\exists)}\,(\forall\to)
$$

Figure 13: A regular derivation in $\mathbf{LJ}^{=}$

In this example, we denote by φ the formula $\forall x(x{\approx}a \vee x{\approx}b \supset \exists y(f(x,y){\approx}b))$. We also denote by $\cdots$ some irrelevant parts of the sequents.

$$
\begin{prooftree}
\AxiomC{}
\RightLabel{(refl)}
\UnaryInfC{$f(a,y){\approx}b, f(b,z){\approx}b, \varphi \rightarrow b{\approx}b$}
\RightLabel{$(\approx)$}
\UnaryInfC{$f(a,y){\approx}b, f(b,z){\approx}b, \varphi \rightarrow f(b,z){\approx}b$}
\RightLabel{$(\rightarrow \exists)$}
\UnaryInfC{$f(a,y){\approx}b, f(b,z){\approx}b, \varphi \rightarrow \exists w(f(b,w){\approx}b)$}
\RightLabel{$(\exists \rightarrow)$}
\UnaryInfC{$f(a,y){\approx}b, \exists y(f(b,y){\approx}b), \varphi \rightarrow \exists w(f(b,w){\approx}b)$}
\AxiomC{}
\RightLabel{(refl)}
\UnaryInfC{$\cdots \rightarrow b{\approx}b$}
\RightLabel{$(\rightarrow \vee_2)$}
\UnaryInfC{$\cdots \rightarrow b{\approx}a \vee b{\approx}b$}
\RightLabel{$(\supset\rightarrow)$}
\BinaryInfC{$f(a,y){\approx}b, b{\approx}a \vee b{\approx}b \supset \exists y(f(b,y){\approx}b), \varphi \rightarrow \exists w(f(b,w){\approx}b)$}
\RightLabel{$(\forall \rightarrow)$}
\UnaryInfC{$f(a,y){\approx}b, \varphi \rightarrow \exists w(f(b,w){\approx}b)$}
\RightLabel{$(\approx)$}
\UnaryInfC{$f(a,y){\approx}b, \varphi \rightarrow \exists w(f(f(a,y),w){\approx}b)$}
\RightLabel{$(\rightarrow \exists)$}
\UnaryInfC{$f(a,y){\approx}b, \varphi \rightarrow \exists v \exists w(f(f(a,v),w){\approx}b)$}
\RightLabel{$(\exists \rightarrow)$}
\UnaryInfC{$\exists y(f(a,y){\approx}b), \varphi \rightarrow \exists v \exists w(f(f(a,v),w){\approx}b)$}
\AxiomC{}
\RightLabel{(refl)}
\UnaryInfC{$\cdots \rightarrow a{\approx}a$}
\RightLabel{$(\rightarrow \vee_1)$}
\UnaryInfC{$\cdots \rightarrow a{\approx}a \vee a{\approx}b$}
\RightLabel{$(\supset\rightarrow)$}
\BinaryInfC{$a{\approx}a \vee a{\approx}b \supset \exists y(f(a,y){\approx}b), \varphi \rightarrow \exists v \exists w(f(f(a,v),w){\approx}b)$}
\RightLabel{$(\forall \rightarrow)$}
\UnaryInfC{$\varphi \rightarrow \exists v \exists w(f(f(a,v),w){\approx}b)$}
\RightLabel{$(\rightarrow \exists)$}
\UnaryInfC{$\varphi \rightarrow \exists u \exists v \exists w(f(f(u,v),w){\approx}b)$}
\RightLabel{$(\rightarrow\supset)$}
\UnaryInfC{$\rightarrow \varphi \supset \exists u \exists v \exists w(f(f(u,v),w){\approx}b)$}
\end{prooftree}
$$

Figure 14: An irregular derivation in $\mathbf{LJ}^{=}$

are given in Figures 13 and 14 . The first derivation is regular while the second is not regular.

Unlike $\mathbf{LK}^=$, the calculus $\mathbf{LJ}^=$ has the *explicit definability property* which can be expressed by

Theorem 9.2 A sequent $\rightarrow \exists x\varphi$ is derivable in $\mathbf{LJ}^=$ if and only if there is a term t such that $\rightarrow \varphi[x \mapsto t]$ is derivable in $\mathbf{LJ}^=$.

Finally, in $\mathbf{LJ}^=$ we cannot get rid of δ-rules, since there are no analogues of skolemization.

9.2 Simultaneous rigid E-unification is inevitable

We introduce one technical notion.

- ► An inference rule is called *invertible* if the derivability of the conclusion of the rule implies the derivability of all premises of the rule.

All rules in $\mathbf{LK}^=$ are invertible, while $\mathbf{LJ}^=$ has some non-invertible rules.

All the methods considered in this paper which are complete for first-order classical logic with equality do not work for intuitionistic logic. The first thing we can note is that simultaneous rigid E-unification is representable in intuitionistic logic in the following way.

Consider any system R of rigid equations defined by

$$R \rightleftharpoons \begin{array}{ccccc} s_{11}\approx t_{11} & \dots & s_{1n_1}\approx t_{1n_1} & \vdash_\forall & s'_1\approx t'_1 \\ \vdots & & \vdots & & \vdots \\ s_{k1}\approx t_{k1} & \dots & s_{kn_k}\approx t_{kn_k} & \vdash_\forall & s'_k\approx t'_k \end{array} \tag{20}$$

Let $x_1, \dots, x_m$ be all variables occurring in R. Define the formulas ψ_R and φ_R as follows:

$$\begin{array}{rl} \psi_R \rightleftharpoons & (s_{11}\approx t_{11} \wedge \dots \wedge s_{1n_1}\approx t_{1n_1} \supset s'_1\approx t'_1) \quad \wedge \\ & \dots \quad \wedge \\ & (s_{k1}\approx t_{k1} \wedge \dots \wedge s_{kn_k}\approx t_{kn_k} \supset s'_k\approx t'_k) \end{array} \tag{21}$$

$$\varphi_R \rightleftharpoons \exists x_1 \dots \exists x_m \psi_R \tag{22}$$

By considering all possible derivations of φ_R in $\mathbf{LJ}^=$, or by using Theorem 9.2, it is easy to see that φ_R is derivable in $\mathbf{LJ}^=$ if and only if there is a substitution σ such that the formula $\psi_R\sigma$ is derivable in $\mathbf{LJ}^=$. Using the fact that the rules $(\rightarrow \wedge)$, $(\rightarrow\supset)$ and $(\wedge \rightarrow)$ are invertible we can prove that $\psi_R\sigma$ is derivable in $\mathbf{LJ}^=$ if and only if for all $j \in \{1, \ldots, k\}$ the sequent

$$s_{j1}\sigma \approx t_{j1}\sigma, \ldots, s_{jn_j}\sigma \approx t_{jn_j}\sigma \rightarrow s'_j\sigma \approx t'_j\sigma$$

is derivable in $\mathbf{LJ}^=$. Derivability of such sequents in $\mathbf{LJ}^=$ is equivalent to their derivability in $\mathbf{LK}^=$. Hence, ψ is provable if and only if σ is a solution to R. This consideration prove the following theorem (Voronkov [156, 159]):

Theorem 9.3 Simultaneous rigid E-unifiability is polynomial-time reducible to derivability in $\mathbf{LJ}^=$.

Thus, unlike classical logic, handling simultaneous rigid E-unification is inevitable in any automated reasoning system for intuitionistic logic with equality.

9.3 Automated reasoning modulo simultaneous rigid E-unification

We established that any complete system of automated reasoning for intuitionistic logic gives us a semi-decision algorithm for simultaneous rigid E-unification. In this section, we show how one can design semi-decision algorithms for $\mathbf{LJ}^=$ modulo simultaneous rigid E-unification. This means that we assume that a semi-decision procedure P for simultaneous rigid E-unification is given and we try to give a general semi-decision procedure for $\mathbf{LJ}^=$ which may call P from time to time. The idea of such a procedure can already be found in equational matings by Gallier et.al. [72, 71, 70]. However, for intuitionistic logic we have to elaborate several things due to the existence of non-invertible rules and to the presence of δ-rules. Due to non-invertible rules, we shall use *derivation skeletons* instead of tableaux or matrices. We shall also built-in simultaneous rigid E-unification directly into the calculus by means of *constraints.* (These constraints are different from the equaliyt and equality constraints we have considered so far.) The exposition here mainly follows [156].

► A *derivation skeleton* is a tree such that

1. its nodes are labelled by the names of inference rules in $\mathbf{LJ}^{=}$, except for $(\approx)$;
2. the number of parents of a node labelled by a name of an inference rule is equal to the number of the premises in such a rule in $\mathbf{LJ}^{=}$;
3. all nodes labelled by antecedent rules or (Ax) are additionally labelled by a natural number.

The natural number in an antecedent rule or in an axiom (Ax) represents the index of the main formula (counting from 0) of this rule in the antecedent of the conclusion of the rule. In other words, this number is the number of formulas in Γ in the corresponding rules of $\mathbf{LJ}^{=}$ (Figure 12). We always display the number as an index.

▶ The *skeleton* of a derivation Π is obtained from Π by removing all sequents and all applications of rules $(\approx)$, and by adding corresponding indices for antecedent rules and axioms (Ax).

We display derivation skeletons as derivations with omitted sequents, nodes are denoted by horizontal bars with labels displayed to the right of the bar. For example, both derivations shown in Figures 13 and 14 have the same skeleton

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\quad}(\mathit{refl})}{\quad}(\to\exists)}{\quad}(\exists\to)_1 \qquad \dfrac{\dfrac{}{\quad}(\mathit{refl})}{\quad}(\to\vee_2)}{\qquad}(\supset\to)_1}{\quad}(\forall\to)_1}{\quad}(\to\exists)}{\quad}(\exists\to)_0 \qquad \dfrac{\dfrac{}{\quad}(\mathit{refl})}{\quad}(\to\vee_1)}{\qquad}(\supset\to)_0}{\quad}(\forall\to)_0}{\quad}(\to\exists)}{\quad}(\to\supset)$$

Derivation skeletons are abstractions of derivations that encode information about the structures of derivations. They abstract from the following information:

1. instantiation of variables in quantifier rules;

2. the order of applications of the equality rules ($\approx$) and ($\approx$).

A problem similar to the existence of equational mating in terminology of Gallier [70] is the following

► *skeleton instantiation problem*: given a sequent $\mathcal{S}$ and a skeleton $\mathbf{S}$, does there exist a derivation of $\mathcal{S}$ in $\mathbf{LJ}^=$ with the skeleton $\mathbf{S}$.

For formulas without equality this problem is decidable in polynomial-time [158]. For formulas with equality the skeleton instantiation problem is undecidable. In fact, we have (Voronkov [156]):

Theorem 9.4 Simultaneous rigid E-unifiability is polynomial-time equivalent to the skeleton instantiation problem.

A more precise formulation of this statement with respect to signatures may be found in [161].

9.3.1 Constraints

As in the case with classical tableaux, we have to make a step from $\mathbf{LJ}^=$ to a free variable calculus. As it has been show above, handling simultaneous rigid E-unification is inevitable for free-variable versions of $\mathbf{LJ}^=$. What we shall do here is to introduce simultaneous rigid E-unification directly in the calculus in the form of *constraints.* This trick is similar to the introduction of constraints in the tableau calculus made in Section 7.2, but the notion of a constraint will be different. Here, constraints will encode the information on possible instantiations of skeletons to real derivations.

Based on the constraints, we define the constraint calculus $\mathbf{LJ}^=_c$. This calculus can be used for defining sequent style proof procedures for intuitionistic logic with equality. The proof procedures based on $\mathbf{LJ}^=_c$ consist of two parts: the construction of a derivation skeleton and the constraint satisfaction part. The constraint to be satisfied is computed from the skeleton of a derivation. The construction of a skeleton can also be considered as a sequence of applications of tableau expansion rules.

► *Constraints* are defined inductively as follows:

1. $\top$ is a constraint;
2. For any terms $t, t_1, \ldots, t_n$, $t \not\preceq \{t_1, \ldots, t_n\}$ is a constraint;
3. Any rigid equation $s_1 \approx t_1, \ldots, s_n \approx t_n \vdash_\forall s \approx t$ is a constraint;
4. If $\mathcal{C}_1, \mathcal{C}_2$ are constraints, then $\mathcal{C}_1 \land \mathcal{C}_2$ is a constraint;
5. If $\mathcal{C}$ is a constraint, x is variable, then $\exists x \mathcal{C}$ is a constraint.

Constraints can be regarded as first order formulas using the atomic formulas $\top$, $t \not\preceq \{t_1, \ldots, t_n\}$ and $E \vdash_\forall s \approx t$ such that $\ldots \not\preceq \{\ldots\}$ and $\ldots \vdash_\forall \ldots$ are considered as a family of predicate symbols of arbitrary arities. Then, we can speak about

- the set $\mathit{vars}(\mathcal{C})$ of *free variables* of a constraint $\mathcal{C}$.

Constraints describe the domain of substitutions. If a constraint $\mathcal{C}$ is true on a substitution σ we say that σ *satisfies* $\mathcal{C}$:

- The notion *a substitution σ satisfies the constraint* $\mathcal{C}$, denoted $\sigma \models \mathcal{C}$, is defined inductively as follows:

  1. $\sigma \models \top$ for every substitution σ;
  2. $\sigma \models E \vdash_\forall s \approx t$ if σ is a solution to the rigid equation $E \vdash_\forall s \approx t$;
  3. $\sigma \models t \not\preceq \{t_1, \ldots, t_n\}$ if $t\sigma$ is a variable and $t\sigma$ does not occur in $t_1\sigma, \ldots, t_n\sigma$;
  4. $\sigma \models \mathcal{C}_1 \land \mathcal{C}_2$ if $\sigma \models \mathcal{C}_1$ and $\sigma \models \mathcal{C}_2$;
  5. $\sigma \models \exists x \mathcal{C}$ if there is a substitution τ such that $\tau \models \mathcal{C}$ and $y\tau = y\sigma$ for every variable y different from x.

- Constraints $\mathcal{C}_1$ and $\mathcal{C}_2$ are *equivalent* if for every substitution σ we have $\sigma \models \mathcal{C}_1$ if and only if $\sigma \models \mathcal{C}_2$.

- A constraint $\mathcal{C}$ is *satisfiable* if there is a substitution σ satisfying $\mathcal{C}$.

9.3.2 Free variable calculus $\mathbf{LJ}_c^=$

In this section we introduce the calculus $\mathbf{LJ}_c^=$ of *constrained sequents.* Before giving its inference rules, we introduce the following notation.

- For a multiset of formulas Γ, denote by $\Gamma_\approx$ the multiset of formulas $\{s \approx t \mid s \approx t \in \Gamma\}$.

The calculus $\mathbf{LJ}_c^=$ deals with *constraint sequents:*

- a *constrained sequent* is an expression $\mathcal{S} \cdot \mathcal{C}$ where $\mathcal{S}$ is a sequent and $\mathcal{C}$ is a constraint;
- the *constraint calculus* $\mathbf{LJ}_c^=$ of constrained sequents is shown in Figure 15.

The notion of a skeleton for $\mathbf{LJ}_c^=$-derivations is similar to that of $\mathbf{LJ}^=$-derivations.

Consider two examples of derivations in $\mathbf{LJ}_c^=$. In both examples we derive the same formula as in Figures 13 and 14 , augmented with some constraint. In the first example, the constraint is unsatisfiable. As in Figures 13 and 14 , we denote by φ the formula $\forall x(x{\approx}a \vee x{\approx}b \supset \exists y(f(x,y){\approx}b))$.

$$
\begin{array}{c}
\overline{\varphi \to f(f(u,v),w){\approx}b \cdot \vdash_\forall f(f(u,v),w){\approx}b}\ (\approx) \\
\overline{\varphi \to \exists w(f(f(u,v),w){\approx}b) \cdot \exists w(\vdash_\forall f(f(u,v),w){\approx}b)}\ (\to \exists) \\
\overline{\varphi \to \exists v\exists w(f(f(u,v),w){\approx}b) \cdot \exists v\exists w(\vdash_\forall f(f(u,v),w){\approx}b)}\ (\to \exists) \\
\overline{\varphi \to \exists u\exists v\exists w(f(f(u,v),w){\approx}b) \cdot \exists u\exists v\exists w(\vdash_\forall f(f(u,v),w){\approx}b)}\ (\to \exists) \\
\to \varphi \supset \exists u\exists v\exists w(f(f(u,v),w){\approx}b) \cdot \exists u\exists v\exists w(\vdash_\forall f(f(u,v),w){\approx}b)\ (\to\supset)
\end{array}
$$

The resulting constraint $\exists u\exists v\exists w(\vdash_\forall f(f(u,v),w){\approx}b)$ of this derivation is evidently unsatisfiable.

Another derivation of this formula has the same skeleton as that of Figures 13 and 14 and is shown in Figure 16.
Here the constraints $\mathcal{C}_1$–$\mathcal{C}_{12}$ are as follows:

$$\frac{}{\Gamma, A, \Delta \to A \cdot \Gamma_{\approx} \cup \Delta_{\approx} \vdash_{\forall} s_1 \approx t_1 \wedge \ldots \wedge \Gamma_{\approx} \cup \Delta_{\approx} \vdash_{\forall} s_n \approx t_n} \ (Ax)$$

$$\frac{}{\Gamma \to t{\approx}t \cdot \Gamma_{\approx} \vdash_{\forall} s{\approx}t} \ (\mathit{refl})$$

$$\frac{\Gamma, \varphi, \psi, \Xi \to \Delta \cdot \mathcal{C}}{\Gamma, \varphi \wedge \psi, \Xi \to \Delta \cdot \mathcal{C}} \ (\wedge \to) \qquad \frac{\Gamma \to \varphi \cdot \mathcal{C}_1 \quad \Gamma \to \psi \cdot \mathcal{C}_2}{\Gamma \to \varphi \wedge \psi \cdot \mathcal{C}_1 \wedge \mathcal{C}_2} \ (\to \wedge)$$

$$\frac{\Gamma, \varphi, \Xi \to \Delta \cdot \mathcal{C}_1 \quad \Gamma, \psi \to \Delta \cdot \mathcal{C}_2}{\Gamma, \varphi \vee \psi, \Xi \to \Delta \cdot \mathcal{C}_1 \wedge \mathcal{C}_2} \ (\vee \to)$$

$$\frac{\Gamma \to \varphi \cdot \mathcal{C}}{\Gamma \to \varphi \vee \psi \cdot \mathcal{C}} \ (\to \vee_1) \qquad \frac{\Gamma \to \psi \cdot \mathcal{C}}{\Gamma \to \varphi \vee \psi \cdot \mathcal{C}} \ (\to \vee_2)$$

$$\frac{\Gamma, \psi, \Xi \to \Delta \cdot \mathcal{C}_1 \quad \Gamma, \varphi \supset \psi, \Xi \to \varphi \cdot \mathcal{C}_2}{\Gamma, \varphi \supset \psi, \Xi \to \Delta \cdot \mathcal{C}_1 \wedge \mathcal{C}_2} \ (\supset\to)$$

$$\frac{\varphi, \Gamma \to \psi \cdot \mathcal{C}}{\Gamma \to \varphi \supset \psi \cdot \mathcal{C}} \ (\to\supset)$$

$$\frac{\Gamma \to \varphi \cdot \mathcal{C}}{\Gamma, \neg\varphi, \Xi \to \cdot\mathcal{C}} \ (\neg \to) \qquad \frac{\varphi, \Gamma \to \cdot\mathcal{C}}{\Gamma \to \neg\varphi \cdot \mathcal{C}} \ (\to \neg)$$

$$\frac{\Gamma, \varphi[x \mapsto y], \forall x\varphi, \Xi \to \Delta \cdot \mathcal{C}}{\Gamma, \forall x\varphi, \Xi \to \Delta \cdot \exists y\mathcal{C}} \ (\forall \to)$$

$$\frac{\Gamma \to \varphi[x \mapsto y] \cdot \mathcal{C}}{\Gamma \to \forall x\varphi \cdot \exists y(y \not\approx \{v_1, \ldots, v_n\} \wedge \mathcal{C})} \ (\to \forall)$$

$$\frac{\Gamma, \varphi[x \mapsto y] \to \Delta \cdot \mathcal{C}}{\Gamma, \exists x\varphi \to \Delta \cdot \exists y(y \not\approx \{v_1, \ldots, v_n\} \wedge \mathcal{C})} \ (\exists \to)$$

$$\frac{\Gamma \to \varphi[x \mapsto y] \cdot \mathcal{C}}{\Gamma \to \exists x\varphi \cdot \exists y\mathcal{C}} \ (\to \exists)$$

In the rule (Ax), A is a predicate symbol different from equality. In the rules $(\forall \to)$–$(\to \exists)$ the variable y has no free occurrences in the sequent in the conclusion of the rules. In the rule $(\to \forall)$, $v_1, \ldots, v_n$ are all free variables of $\Gamma, \forall x\varphi$. In the rule $(\exists \to)$, $v_1, \ldots, v_n$ are all free variables of $\Gamma, \exists x\varphi, \Delta$.

Figure 15: Calculus $\mathbf{LJ}^{=}_{c}$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{f(x_1,y_1)\approx b, f(x_2,y_2)\approx b, \varphi \to f(f(u,v),w)\approx b \cdot \mathcal{C}_1}\,(\approx)}{f(x_1,y_1)\approx b, f(x_2,y_2)\approx b, \varphi \to \exists w(f(f(u,v),w)\approx b)\cdot\mathcal{C}_2}\,(\to \exists)}{f(x_1,y_1)\approx b, \exists y(f(x_2,y)\approx b), \varphi \to \exists w(f(f(u,v),w)\approx b)\cdot \mathcal{C}_4}\,(\exists \to) \quad \dfrac{\dfrac{}{\cdots \to x_2\approx b\cdot\mathcal{C}_3}\,(\approx)}{\cdots\to x_2\approx a\vee x_2\approx b\cdot\mathcal{C}_3}\,(\to \vee_2)}{f(x_1,y_1)\approx b, x_2\approx a\vee x_2\approx b\supset \exists y(f(x_2,y)\approx b),\varphi\to\exists w(f(f(u,v),w)\approx b)\cdot\mathcal{C}_5}\,(\supset\to)}{f(x_1,y_1)\approx b, \varphi \to \exists w(f(f(u,v),w)\approx b)\cdot\mathcal{C}_6}\,(\forall \to)$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{f(x_1,y_1)\approx b, \varphi \to \exists w(f(f(u,v),w)\approx b)\cdot\mathcal{C}_6}{f(x_1,y_1)\approx b, \varphi \to \exists v\exists w(f(f(u,v),w)\approx b)\cdot\mathcal{C}_7}\,(\to \exists)}{\exists y(f(x_1,y)\approx b), \varphi \to \exists v\exists w(f(f(u,v),w)\approx b)\cdot\mathcal{C}_9}\,(\exists \to) \quad \dfrac{\dfrac{}{\cdots \to x_1\approx a\cdot\mathcal{C}_8}\,(\approx)}{\cdots\to x_1\approx a\vee x_1\approx b\cdot\mathcal{C}_8}\,(\to \vee_1)}{x_1\approx a\vee x_1\approx b\supset \exists y(f(x_1,y)\approx b),\varphi\to\exists v\exists w(f(f(u,v),w)\approx b)\cdot\mathcal{C}_{10}}\,(\supset\to)}{\varphi \to \exists v\exists w(f(f(u,v),w)\approx b)\cdot\mathcal{C}_{11}}\,(\forall \to)}{\varphi \to \exists u\exists v\exists w(f(f(u,v),w)\approx b)\cdot\mathcal{C}_{12}}\,(\to \exists)}{\to \varphi \supset \exists u\exists v\exists w(f(f(u,v),w)\approx b)\cdot\mathcal{C}_{12}}\,(\to\supset)$$

Figure 16: A derivation in $\mathbf{LJ}_c^=$

$$
\begin{array}{lcl}
\mathcal{C}_1 & \rightleftharpoons & f(x_1,y_1)\approx b, f(x_2,y_2)\approx b \vdash_\forall f(f(u,v),w)\approx b \\
\mathcal{C}_2 & \rightleftharpoons & \exists w(f(x_1,y_1)\approx b, f(x_2,y_2)\approx b \vdash_\forall f(f(u,v),w)\approx b) \\
\mathcal{C}_3 & \rightleftharpoons & f(x_1,y_1)\approx b \vdash_\forall x_2\approx b \\
\mathcal{C}_4 & \rightleftharpoons & \exists y_2(y_2 \not\preceq \{x_1,y_1,x_2,u,v\} \wedge \\
 & & \quad \exists w(f(x_1,y_1)\approx b, f(x_2,y_2)\approx b \vdash_\forall f(f(u,v),w)\approx b)) \\
\mathcal{C}_5 & \rightleftharpoons & \mathcal{C}_4 \wedge \mathcal{C}_3 \\
\mathcal{C}_6 & \rightleftharpoons & \exists x_2(\mathcal{C}_4 \wedge \mathcal{C}_3) \\
\mathcal{C}_7 & \rightleftharpoons & \exists v \exists x_2(\mathcal{C}_4 \wedge \mathcal{C}_3) \\
\mathcal{C}_8 & \rightleftharpoons & \vdash_\forall x_1\approx a \\
\mathcal{C}_9 & \rightleftharpoons & \exists y_1(y_1 \not\preceq \{x_1,u\} \wedge \exists v\exists x_2(\mathcal{C}_4 \wedge \mathcal{C}_3)) \\
\mathcal{C}_{10} & \rightleftharpoons & \exists y_1(y_1 \not\preceq \{x_1,u\} \wedge \exists v\exists x_2(\mathcal{C}_4 \wedge \mathcal{C}_3)) \wedge \vdash_\forall x_1\approx a \\
\mathcal{C}_{11} & \rightleftharpoons & \exists x_1(\exists y_1(y_1 \not\preceq \{x_1,u\} \wedge \exists v\exists x_2(\mathcal{C}_4 \wedge \mathcal{C}_3)) \wedge \vdash_\forall x_1\approx a) \\
\mathcal{C}_{12} & \rightleftharpoons & \exists u\exists x_1(\exists y_1(y_1 \not\preceq \{x_1,u\} \wedge \exists v\exists x_2(\mathcal{C}_4 \wedge \mathcal{C}_3)) \wedge \vdash_\forall x_1\approx a)
\end{array}
$$

The resulting constraint $\mathcal{C}_{12}$ of this derivation is satisfiable. To establish this, we first note that $\mathcal{C}_{12}$ is satisfiable if and only if such is the following constraint $\mathcal{C}$ obtained from $\mathcal{C}_{12}$ be removing all quantifiers:

$$
\begin{array}{l}
f(x_1,y_1)\approx b, f(x_2,y_2)\approx b \vdash_\forall f(f(u,v),w)\approx b \wedge \\
f(x_1,y_1)\approx b \vdash_\forall x_2\approx b \wedge \\
y_2 \not\preceq \{x_1,y_1,x_2,u,v\} \wedge \\
\vdash_\forall x_1\approx a \wedge \\
y_1 \not\preceq \{x_1,u\}
\end{array}
$$

This quantifier-free constraint is satisfied by the substitution $[u \mapsto a, x_1 \mapsto a, v \mapsto y_1, x_2 \mapsto b, w \mapsto y_2]$. Note that this substitutions was used for variable instantiation in γ-rules in the derivation of Figures 13 and 14 .

The correspondence of this derivation in $\mathbf{LJ}_c^=$ to the derivation in $\mathbf{LJ}^=$ is not coincidental: derivations in $\mathbf{LJ}_c^=$ characterize all derivations in $\mathbf{LJ}^=$ with a given skeleton in the following way:

Theorem 9.5 Let $\Gamma \to \varphi$ be a sequent, θ be a substitution and $\mathbf{S}$ be a skeleton. Then the following conditions are equivalent:

1. there is a derivation of $\Gamma\theta \to \varphi\theta$ in $\mathbf{LJ}^=$ with the skeleton $\mathbf{S}$;

2. there is a constraint $\mathcal{C}$ and a derivation of $\Gamma \to \varphi \cdot \mathcal{C}$ in $\mathbf{LJ}_c^=$ with the skeleton $\mathbf{S}$ such that $\theta \models \mathcal{C}$.

This statement also gives us the completeness of $\mathbf{LJ}_c^=$:

Theorem 9.6 (completeness of $\mathbf{LJ}_c^=$) Let $\mathcal{S}$ be a closed sequent. Then $\mathcal{S}$ is derivable in $\mathbf{LJ}^=$ if and only if there is a satisfiable constraint $\mathcal{C}$ such that $\mathcal{S} \cdot \mathcal{C}$ is derivable in $\mathbf{LJ}_c^=$.

Note that the set of constraints $\mathcal{C}$ such that $\mathcal{S} \cdot \mathcal{C}$ is derivable in $\mathbf{LJ}_c^=$ with a skeleton $\mathbf{S}$ is *uniquely defined by the skeleton* $\mathbf{S}$ up to renaming of bound variables. Thus, the proof-search in intuitionistic logic can be considered as consisting of two parts: constructing a skeleton and the constraint satisfaction. This is similar to the matrix characterization of provability for classical logic.

The skeleton instantiation problem is undecidable which causes doubts about the efficiency of such procedures. In classical logic, there are many complete methods for reasoning with equality which do not use simultaneous rigid E-unification. However, for intuitionistic logic, because of simple reduction of simultaneous rigid E-unifiability to the derivability problem, handling of simultaneous rigid E-unification is inevitable for designing a complete procedure. In addition, some special cases of simultaneous rigid E-unification are decidable [45] (for example, the function-free case is NP-complete) which allows us to use decision procedures for these fragments.

9.4 Modal logics with equality

There is no uniform viewpoint on the use of quantification in modal logics. Neither is there a uniform viewpoint on the use of equality. The survey Garson [75] contains some material on this topic. Without going into many detail, we show that the above considerations about intuitionistic logic can as well be applied to some formalizations of various modal logics. Readers not familiar with modal logics can skip this section. We consider modal logics for which equality is axiomatized by the rules (*refl*) and ($\approx$), like for $\mathbf{LK}^=$ or $\mathbf{LJ}^=$. As an example, we take $\mathbf{S4}^=$.

We shall use the following notation. For any multiset of formulas Γ,

- ▶ $\Box\Gamma$ (respectively, $\Diamond\Gamma$) denotes the multiset of formulas $\{\Box\varphi \mid \varphi \in \Gamma\}$ (respectively, $\{\Diamond\varphi \mid \varphi \in \Gamma\}$).

- ▶ The *sequent calculus* $\mathbf{S4}^=$ for modal logic with equality is obtained from $\mathbf{LK}^=$ by adding the following inference rules:

$$
\frac{\Gamma, T\;\varphi}{\Gamma, T\;\Box\varphi}\;(\nu)
\qquad\qquad
\frac{T\;\Gamma, F\;\Delta, F\;\varphi}{\Xi, T\;\Box\Gamma, F\;\Diamond\Delta, F\;\Box\varphi}\;(\nu)
$$

$$
\frac{T\;\Gamma, F\;\Delta, T\;\varphi}{\Xi, T\;\Box\Gamma, F\;\Diamond\Delta, T\;\Diamond\varphi}\;(\nu)
\qquad\qquad
\frac{\Gamma, F\;\varphi}{\Gamma, F\;\Diamond\varphi}\;(\nu)
$$

Consider again any system R of rigid equations of the from (20) and the formula ψ_R of the form (21) (see page 96). Let $\varphi \rightleftharpoons \exists x_1 \ldots \exists x_m \Box\psi_R$. Any derivation of φ in $\mathbf{S4}^=$ has the following form, for a suitable substitution σ:

$$
\begin{array}{ll}
\left.\begin{array}{c}\vdots\\[1ex] F\;\psi_R\sigma\end{array}\right\} & \text{derivation of } \psi_R\sigma\\[2ex]
\left.\begin{array}{c}\overline{\Xi, F\;\Box\psi_R\sigma}\;(\nu)\\[1ex] \vdots\\[1ex] F\;\exists x_1\ldots\exists x_m\Box\psi_R\end{array}\right\} & \text{derivation using only } \gamma\text{-rules}
\end{array}
$$

Again, such a derivation exists if and only if σ is a solution to R. Thus, nearly every result asserted above for $\mathbf{LJ}^=$, can be reformulated in a suitable way for $\mathbf{S4}^=$.

The necessity of handling simultaneous rigid E-unification will arise in almost any logic whose semantics is based on possible worlds. The same trick that we used for modal logics above can, for instance, be made for some temporal logics.

References

[1] P.B. Andrews. Refutations by matings. *IEEE Trans. Comput.*, 25:801–807, 1976.

[2] P.B. Andrews. Theorem proving via general matings. *Journal of the Association for Computing Machinery*, 28(2):193–214, 1981.

[3] P.B. Andrews. *An introduction to type theory: to truth through proof.* Academic Press, 1986.

[4] A. Avron. Gentzen-type systems, resolution and tableaux. *Journal of Automated Reasoning*, 10:256–281, 1993.

[5] M. Baaz. Note on the existence of most general semi-unifiers. In *Arithmetic, Proof Theory and Computation Complexity*, volume 23 of *Oxford Logic Guides*, pages 20–29. Oxford University Press, 1993.

[6] M. Baaz and C.G. Fermüller. Non-elementary speedups between different versions of tableaux. In P. Baumgartner, R. Hähnle, and J. Posegga, editors, *Theorem Proving with Analytic Tableaux and Related Methods*, number 918 in Lecture Notes in Artificial Intelligence, pages 217–230, Schloß Rheinfels, St. Goar, Germany, May 1995.

[7] M. Baaz and A. Leitsch. On skolemization and proof complexity. *Fundamenta Informaticae*, 20(4), 1994.

[8] L. Bachmair and H. Ganzinger. On restriction of ordered paramodulation with simplication. In M.E. Stickel, editor, *Proc. 10th CADE*, volume 449 of *Lecture Notes in Artificial Intelligence*, pages 427–441, 1990.

[9] L. Bachmair and H. Ganzinger. Completion of first-order clauses with equality by strict superposition. In *Conditional and Typed Rewriting Systems*, volume 516 of *Lecture Notes in Computer Science*, pages 164–180, Montreal, 1991. Springer Verlag.

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

[11] L. Bachmair and H. Ganzinger. Strict basic superposition. In C. Kirchner and H. Kirchner, editors, *Automated Deduction — CADE-15. 15th International Conference on Automated Deduction*, volume 1421 of *Lecture Notes in Artificial Intelligence*, pages 160–174, Lindau, Germany, 1998. Springer Verlag.

[12] 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.

[13] L. Bachmair, H. Ganzinger, C. Lynch, and W. Snyder. Basic paramodulation. *Information and Computation*, 121:172–192, 1995.

[14] L. Bachmair, H. Ganzinger, and A. Voronkov. Elimination of equality via transformation with ordering constraints. In C. Kirchner and H. Kirchner, editors, *Automated Deduction — CADE-15. 15th International Conference on Automated Deduction*, volume 1421 of *Lecture Notes in Artificial Intelligence*, pages 175–190, Lindau, Germany, 1998. Springer Verlag.

[15] P. Baumgartner and U. Furbach. Consolution as a framework for comparing calculi. *Journal of Symbolic Computations*, 16:445–477, 1993.

[16] P. Baumgartner and U. Furbach. PROTEIN: A *PRO*ver with a *T*heory *E*xtension *IN*terface. In A. Bundy, editor, *Automated Deduction — CADE-12. 12th International Conference on Automated Deduction.*, volume 814 of *Lecture Notes in Artificial Intelligence*, pages 769–773, Nancy, France, June/July 1994.

[17] G. Becher and U. Petermann. Rigid unification by completion and rigid paramodulation. In B. Nebel and L. Dreschler-Fischer, editors, *KI-94: Advances in Artificial Intelligence. 18th German Annual Conference on Artificial Intelligence*, volume 861 of *Lecture Notes in Artificial Intelligence*, pages 319–330, Saarbrücken, Germany, September 1994. Springer Verlag.

[18] 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.

[19] B. Beckert. Are minimal solutions to simultaneous rigid E-unification sufficient for adding equality to semantic tableaux? Privately circulated manuscript, University of Karlsruhe, 1995.

[20] B. Beckert. Equality and other theory inferences. In M. D'Agostino, D. Gabbay, R. Hähnle, and J. Posegga, editors, *Handbook of Tableau Methods*. Kluwer, 1997.

[21] B. Beckert. Semantic tableaux with equality. *Journal of Logic and Computation*, 7(1):38–58, 1997.

[22] 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.

[23] B. Beckert, R. Hähnle, and P.H. Schmitt. The even more liberalized δ-rule in free variable semantic tableaux. In G. Gottlob, A. Leitsch, and D. Mundici, editors, *Computational Logic and Proof Theory. Proceedings of the Third Kurt Gödel Colloquium, KGC'93*, volume 713 of *Lecture Notes in Computer Science*, pages 108–119, Brno, August 1993.

[24] A.P. Beltyukov. Decidability of the universal theory of natural numbers with addition and divisibility (in Russian). *Zapiski Nauchnyh Seminarov LOMI*, 60:15–28, 1976. English translation in: Journal of Soviet Mathematics.

[25] D. Benanav. Simultaneous paramodulation. In M.E. Stickel, editor, *Proc. 10th Int. Conf. on Automated Deduction*, volume 449 of *Lecture Notes in Artificial Intelligence*, pages 442–455, 1990.

[26] E.W. Beth. *The Foundations of Mathematics*. North Holland, 1959.

[27] 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.

[28] W. Bibel. On matrices with connections. *Journal of the Association for Computing Machinery*, 28(4):633–645, 1981.

[29] W. Bibel. *Deduction. Automated Logic.* Academic Press, 1993.

[30] W. Bibel, S. Brüning, U. Egly, D. Korn, and T. Rath. Issues in theorem proving based on the connection method. In P. Baumgartner, R. Hähnle, and J. Posegga, editors, *Theorem Proving with Analytic Tableaux and Related Methods*, number 918 in Lecture Notes in Artificial Intelligence, pages 1–16, Schloß Rheinfels, St. Goar, Germany, May 1995.

[31] W. Bibel and J. Schreiber. Proof search in a Gentzen-like system of first order logic. In E. Gelenbe and D. Potier, editors, *Proc. Int. Computing Symp.*, pages 205–212. North Holland, 1975.

[32] 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.

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

[34] C.L. Chang. Theorem proving with variable-constrained resolution. *Information Sciences*, 4:217–231, 1972.

[35] H.B. Curry. *Foundations of Mathematical Logic.* New York, 1963.

[36] J.L. Darlington. Automatic theorem proving with equality substitutions and mathematical induction. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, volume 3, pages 113–127. American Elsevier, N.Y., 1968.

[37] E. De Kogel. Rigid E-unification simplified. In P. Baumgartner, R. Hähnle, and J. Posegga, editors, *Theorem Proving with Analytic Tableaux and Related Methods*, number 918 in Lecture Notes in Artificial Intelligence, pages 17–30, Schloß Rheinfels, St. Goar, Germany, May 1995.

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

[39] A. 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, Kiev, 1982.

[40] A. Degtyarev. Rewriting strategies for computational logic (in Russian). In *Proc. of the Soviet Conf. on Applied Logic*, pages 69–72, Novosibirsk, 1985.

[41] A. Degtyarev, Yu. Gurevich, P. Narendran, M. Veanes, and A. Voronkov. The decidability of simultaneous rigid E-unification with one variable. UPMAIL Technical Report 139, Uppsala University, Computing Science Department, March 1997. To appear in RTA'98.

[42] A. Degtyarev, Yu. Gurevich, P. Narendran, M. Veanes, and A. Voronkov. The decidability of simultaneous rigid E-unification with one variable. In T. Nipkow, editor, *Rewriting Techniques and Applications, RTA'98*, volume 1379 of *Lecture Notes in Computer Science*, pages 181–195. Springer Verlag, 1998.

[43] A. Degtyarev, Yu. Gurevich, and A. Voronkov. Herbrand's theorem and equational reasoning: Problems and solutions. In *Bulletin of the European Association for Theoretical Computer Science*, volume 60, pages 78–95. October 1996. The "Logic in Computer Science" column.

[44] A. Degtyarev, Yu. Koval, and A. Voronkov. Handling equality in logic programming via basic folding. UPMAIL Technical Report 101, Uppsala University, Computing Science Department, March 1995. Revised June 11, 1997.

[45] A. Degtyarev, Yu. Matiyasevich, and A. Voronkov. Simultaneous rigid E-unification and related algorithmic problems. In *Eleventh Annual IEEE Symposium on Logic in Computer Science (LICS'96)*, pages 494–502, New Brunswick, NJ, July 1996. IEEE Computer Society Press.

[46] A. Degtyarev and A. Voronkov. Equality control methods in machine theorem proving. *Cybernetics*, 22(3):298–307, 1986.

[47] A. Degtyarev and A. Voronkov. Equality elimination for semantic tableaux. UPMAIL Technical Report 90, Uppsala University, Computing Science Department, December 1994.

[48] A. Degtyarev and A. Voronkov. Equality elimination for the inverse method and extension procedures. In C.S. Mellish, editor, *Proc. International Joint Conference on Artificial Intelligence (IJCAI)*, volume 1, pages 342–347, Montréal, August 1995.

[49] A. Degtyarev and A. Voronkov. General connections via equality elimination. In M. De Glas and Z. Pawlak, editors, *Second World Conference on the Fundamentals of Artificial Intelligence (WOCFAI-95)*, pages 109–120, Paris, July 1995. Angkor.

[50] A. Degtyarev and A. Voronkov. A new procedural interpretation of Horn clauses with equality. In Leon Sterling, editor, *Proceedings of the Twelfth International Conference on Logic Programming*, pages 565–579. The MIT Press, 1995.

[51] A. Degtyarev and A. Voronkov. Reduction of second-order unification to simultaneous rigid E-unification. UPMAIL Technical Report 109, Uppsala University, Computing Science Department, June 1995.

[52] A. Degtyarev and A. Voronkov. Simultaneous rigid E-unification is undecidable. UPMAIL Technical Report 105, Uppsala University, Computing Science Department, May 1995.

[53] A. Degtyarev and A. Voronkov. Decidability problems for the prenex fragment of intuitionistic logic. In *Eleventh Annual IEEE Symposium on Logic in Computer Science (LICS'96)*, pages 503–512, New Brunswick, NJ, July 1996. IEEE Computer Society Press.

[54] A. Degtyarev and A. Voronkov. Equality elimination for the tableau method. In J. Calmet and C. Limongelli, editors, *Design and Implementation of Symbolic Computation Systems. International Symposium, DISCO'96*, volume 1128 of *Lecture Notes in Computer Science*, pages 46–60, Karlsruhe, Germany, September 1996.

[55] A. Degtyarev and A. Voronkov. Handling equality in logic programs via basic folding. In R. Dyckhoff, H. Herre, and P. Schroeder-Heister, editors, *Extensions of Logic Programming (5th International Workshop, ELP'96)*, volume 1050 of *Lecture Notes in Computer Science*, pages 119–136, Leipzig, Germany, March 1996.

[56] A. Degtyarev and A. Voronkov. Simultaneous rigid E-unification is undecidable. In H. Kleine Büning, editor, *Computer Science Logic. 9th International Workshop, CSL'95*, volume 1092 of *Lecture Notes in Computer Science*, pages 178–190, Paderborn, Germany, September 1995, 1996.

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

[58] A. Degtyarev and A. Voronkov. What you always wanted to know about rigid E-unification. In J.J. Alferes, L.M. Pereira, and E. Orlowska, editors, *Logics in Artificial Intelligence. European Workshop, JELIA'96*, volume 1126 of *Lecture Notes in Artificial Intelligence*, pages 50–69, Évora, Portugal, September/October 1996.

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

[60] R. Demolombe. An efficient strategy for non-Horn deductive databases. In G.X. Ritter, editor, *Information Processing 89*, pages 325–330. Elsevier Science, 1989.

[61] E. Eder. An implementation of a theorem prover based on the connection method. In W. Bibel and B. Petkoff, editors, *AIMSA'84, Artificial Intelligence — Methodology, Systems, Application*, pages 121–128. North Holland, September 1984.

[62] E. Eder. A comparison of the resolution calculus and the connection method, and a new calculus generalizing both methods. In E. Börger, G. Jäger, H. Kleine Büning, and M.M. Richter, editors, *CSL'88 (Proc. 2nd Workshop on Computer Science Logic)*, volume 385 of *Lecture Notes in Computer Science*, pages 80–98. Springer Verlag, 1988.

[63] E. Eder. Consolution and its relation with resolution. In *Proc. International Joint Conference on Artificial Intelligence (IJCAI)*, pages 132–136, 1991.

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

[65] M. Fitting. Tableaux for logic programming. *Journal of Automated Reasoning*, 13:175–188, 1994.

[66] M. Fitting. *First Order Logic and Automated Theorem Proving.* Springer Verlag, New York, 1996. Second edition.

[67] J. Gallier. *Logic for Computer Science: Foundations of Automatic Theorem Proving.* Harper and Row, New York, 1986.

[68] 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.

[69] J. Gallier, P. Narendran, D. Plaisted, and W. Snyder. Rigid E-unification: NP-completeness and applications to equational matings. *Information and Computation*, 87(1/2):129–195, 1990.

[70] 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):377–429, 1992.

[71] J.H. Gallier, P. Narendran, D. Plaisted, and W. Snyder. Rigid E-unification is NP-complete. In *Proc. IEEE Conference on Logic in Computer Science (LICS)*, pages 338–346. IEEE Computer Society Press, July 1988.

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

[73] J.H. Gallier, S. Raatz, and W. Snyder. Rigid E-unification and its applications to equational matings. In H. Aït Kaci and M. Nivat, editors, *Resolution of Equations in Algebraic Structures*, volume 1, pages 151–216. Academic Press, 1989.

[74] J.H. Gallier and W. Snyder. Complete sets of transformations for general E-unification. *Theoretical Computer Science*, 67:203–260, 1989.

[75] J.W. Garson. Quantification in modal logic. In D. Gabbay and F. Guenther, editors, *Handbook in Philosophical Logic*, volume II, chapter II.5, pages 249–307. D. Reidel Publishing Company, 1984.

[76] G. Gentzen. Untersuchungen über das logische Schließen. *Mathematical Zeitschrift*, 39:176–210, 405–431, 1934.

[77] G. Gentzen. Die Wiederspruchsfreiheit der reinen Zahlentheorie. *Mathematische Annalen*, 112:493–565, 1936.

[78] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–101, 1987.

[79] J.-Y. Girard. *Proof Theory and Logical Complexity.* Studies in Proof Theory. Bibliopolis, Napoly, 1987.

[80] W.D. Goldfarb. The undecidability of the second-order unification problem. *Theoretical Computer Science*, 13:225–230, 1981.

[81] J. Goubault. A rule-based algorithm for rigid E-unification. In Georg Gottlob, Alexander Leitsch, and Daniele Mundici, editors, *Computational Logic and Proof Theory. Proceedings of the Third Kurt Gödel Colloquium, KGC'93*, volume 713 of *Lecture Notes in Computer Science*, pages 202–210, Brno, August 1993.

[82] J. Goubault. Rigid $\vec{E}$-unifiability is DEXPTIME-complete. In *Proc. IEEE Conference on Logic in Computer Science (LICS).* IEEE Computer Society Press, 1994.

[83] Yu. Gurevich and A. Voronkov. Monadic simultaneous rigid E-unification and related problems. UPMAIL Technical Report 137, Uppsala University, Computing Science Department, February 1997. Revised June 20, 1997.

[84] Yu. Gurevich and A. Voronkov. Monadic simultaneous rigid E-unification and related problems. In P. Degano, R. Corrieri, and A. Marchetti-Spaccamella, editors, *Automata, Languages and Programming. 24th International Colloquium, ICALP'97*, volume 1256 of *Lecture Notes in Computer Science*, pages 154–165, Bologna, Italy, July 1997.

[85] R. Hähnle and P.H. Schmitt. The liberalized δ-rule in free variable semantic tableaux. *Journal of Automated Reasoning*, 13:211–221, 1994.

[86] J. Hsiang and M. Rusinowitch. A new method for establishing refutational completeness in theorem proving. In *Proc. 8th CADE*, volume 230 of *Lecture Notes in Computer Science*, pages 141–152, 1986.

[87] J. Hsiang and M. Rusinowitch. Proving refutational completeness of theorem proving strategies: the transfinite semantic tree method. *Journal of the Association for Computing Machinery*, 38(3):559–587, 1991.

[88] J.M. Hullot. Canonical forms and unification. In *5th CADE*, volume 87 of *Lecture Notes in Computer Science*, pages 318–334, 1980.

[89] O. Ibens and R. Letz. Subgoal alternation in model elimination. In D. Galmiche, editor, *Automated Reasoning with Analytic Tableaux and Related Methods*, volume 1227 of *Lecture Notes in Artificial Intelligence*, pages 201–215. Springer Verlag, 1997.

[90] S. Kanger. *Provability in Logic*, volume 1 of *Studies in Philosophy*. Almqvist and Wicksell, Stockholm, 1957.

[91] 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.

[92] S.C. Kleene. *Introduction to Metamathematics*. Van Nostrand P.C., Amsterdam, 1952.

[93] D. Knuth and P. Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon Press, Oxford, 1970.

[94] D.S. Lankford. Canonical inference. Technical report, Department of Mathematics, South-Western University, Georgetown, Texas, 1975.

[95] R.C.T. Lee and C.L. Chang. *Symbolic Logic and Mechanical Theorem Proving*. Academic Press, 1973.

[96] V. Lifschitz. A normal form of derivations in predicate calculus with equality and function symbols (in Russian). *Zapiski Nauchnyh Seminarov LOMI*, 4:58–64, 1967. English Translation in: Seminars in Mathematics: Steklov Math. Inst. 4, Consultants Bureau, NY-London, 1969.

[97] 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, RI, 1971.

[98] V. Lifschitz. What is the inverse method? *Journal of Automated Reasoning*, 5(1):1–23, 1989.

[99] L. Lipshitz. The Diophantine problem for addition and divisibility. *Transactions of the American Mathematical Society*, 235:271–283, January 1978.

[100] L. Lipshitz. Some remarks on the Diophantine problem for addition and divisibility. *Bull. Soc. Math. Belg. Sér. B*, 33(1):41–52, 1981.

[101] D.W. Loveland. *Automated Theorem Proving: a Logical Basis.* North Holland, 1978.

[102] E.L. Lusk. Controlling redundancy in large search spaces: Argonne-style theorem proving through the years. In A. Voronkov, editor, *Logic Programming and Automated Reasoning. International Conference LPAR'92.*, volume 624 of *Lecture Notes in Artificial Intelligence*, pages 96–106, St.Petersburg, Russia, July 1992.

[103] C. Lynch. Oriented equational logic is complete. *Journal of Symbolic Computations*, 23(1):23–45, 1997.

[104] G.S. Makanin. The problem of solvability of equations in free semigroups. *Mat. Sbornik (in Russian)*, 103(2):147–236, 1977. English Translation in American Mathematical Soc. Translations (2), vol. 117, 1981.

[105] V.I. Mart'janov. Universal extended theories of integers. *Algebra i Logika*, 16(5):588–602, 1977.

[106] S.Yu. Maslov. The inverse method of establishing deducibility in the classical predicate calculus. *Soviet Mathematical Doklady*, 5:1420–1424, 1964.

[107] S.Yu. Maslov. The generalization of the inverse method to predicate calculus with equality (in Russian). *Zapiski Nauchnyh Seminarov*

LOMI, 20:80–96, 1971. English translation in: Journal of Soviet Mathematics 1, no. 1.

[108] S.Yu. Maslov. An inverse method for establishing deducibility of nonprenex formulas of the predicate calculus. In J.Siekmann and G.Wrightson, editors, *Automation of Reasoning (Classical papers on Computational Logic)*, volume 2, pages 48–54. Springer Verlag, 1983.

[109] S.Yu. Maslov. Relationship between tactics of the inverse method and the resolution method. In J.Siekmann and G.Wrightson, editors, *Automation of Reasoning (Classical papers on Computational Logic)*, volume 2, pages 264–272. Springer Verlag, 1983.

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

[111] 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.

[112] S.Yu. Maslov, G.E. Mints, and V.P. Orevkov. Mechanical proof-search and the theory of logical deduction in the USSR. In J.Siekmann and G.Wrightson, editors, *Automation of Reasoning (Classical papers on Computational Logic)*, volume 1, pages 29–38. Springer Verlag, 1983.

[113] V.A. Matulis. Two variants of classical predicate calculus without structure rules (in Russian). *Soviet Mathematical Doklady*, 147(5):1029–1031, 1962.

[114] V.A. Matulis. On variants of classical predicate calculus with the unique deduction tree (in Russian). *Soviet Mathematical Doklady*, 148:768–770, 1963.

[115] 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, MA, 1991.

[116] G. Mints. Gentzen-type systems and resolution rules. Part I. Propositional logic. In P. Martin-Löf and G. Mints, editors, *COLOG-88*, volume 417 of *Lecture Notes in Computer Science*, pages 198–231. Springer Verlag, 1990.

[117] G. Mints, V. Orevkov, and T. Tammet. Transfer of sequent calculus strategies to resolution. In *Proof Theory of Modal Logic*, Studies in Pure and Applied Logic. Kluwer Academic Publishers, 1996. To appear.

[118] M. Moser, C. Lynch, and J. Steinbach. Model elimination with basic ordered paramodulation. Technical Report AR-95-11, Fakultät für Informatik, Technische Universität München, München, 1995.

[119] M. Moser and J. Steinbach. STE-modification revisited. Technical Report AR-97-03, Fakultät für Informatik, Technische Universität München, München, 1997.

[120] R. Nieuwenhuis. Simple LPO constraint solving methods. *Information Processing Letters*, 47:65–69, 1993.

[121] 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.

[122] 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.

[123] R. Nieuwenhuis and A. Rubio. Theorem proving with ordering and equality constrained clauses. *Journal of Symbolic Computations*, 19:321–351, 1995.

[124] S.A. Norgela. On the size of derivations under minus-normalization in Russian. In V.A. Smirnov, editor, *The Theory of Logical Inference*. Institute of Philosophy, Moscow, 1974.

[125] V.P. Orevkov. On nonlengthening applications of equality rules (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, p.77-79.

[126] J. Pais and G.E. Peterson. Using forcing to prove completeness of resolution and paramodulation. *Journal of Symbolic Computations*, 11:3–19, 1991.

[127] U. Petermann. A complete connection calculus with rigid E-unification. In *JELIA'94*, volume 838 of *Lecture Notes in Computer Science*, pages 152–166, 1994.

[128] G.E. Peterson. A technique for establishing completeness results in theorem proving with equality. *SIAM Journal of Computing*, 12(1):82–100, 1983.

[129] D.A. Plaisted. Special cases and substitutes for rigid E-unification. Technical Report MPI-I-95-2-010, Max-Planck-Institut für Informatik, November 1995.

[130] D.A. Plaisted and S. Greenbaum. A structure-preserving clause form translation. *Journal of Symbolic Computations*, 2:293–304, 1986.

[131] A. Pliuškevičiene. Elimination of cut-type rules in axiomatic theories with equality (in Russian). *Zapiski Nauchnyh Seminarov LOMI*, 16:175–184, 1969. English Translation in: Seminars in Mathematics: Steklov Math. Inst. 16, Consultants Bureau, NY-London, 1971, p.90–94.

[132] 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.

[133] G. Robinson and L. Wos. Completeness of paramodulation. *Journal of Symbolic Logic*, 34(1):159–160, 1969.

[134] G. Robinson and L.T. Wos. Paramodulation and theorem-proving in first order theories with equality. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, volume 4, pages 135–150. Edinburgh University Press, 1969.

[135] J.A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the Association for Computing Machinery*, 12(1):23–41, 1965.

[136] J.A. Robinson. The generalized resolution principle. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, volume 3, pages 77–94. American Elsevier, N.Y., 1968.

[137] M.G. Rogava. On sequential modifications of applied predicate calculi (in Russian). *Zapiski Nauchnyh Seminarov LOMI*, 4:175–189, 1967. English Translation in: Seminars in Mathematics: Steklov Math. Inst. 4, Consultants Bureau, NY-London, 1969, p.77–81.

[138] 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.

[139] J. Schumann. Tableau-based theorem provers: Systems and implementations. *Journal of Automated Reasoning*, 13(3):409–421, 1994.

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

[141] N. Shankar. Proof search in the intuitionistic sequent calculus. In D. Kapur, editor, *11th International Conference on Automated Deduction*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 522–536, Saratoga Springs, NY, USA, June 1992. Springer Verlag.

[142] R. Shostak. An algorithm for reasoning about equality. *Communications of the ACM*, 21:583–585, July 1978.

[143] J.R. Slagle. Automated theorem-proving for theories with simplifiers, commutativity and associativity. *Journal of the Association for Computing Machinery*, 21(4):622–642, 1974.

[144] R.M. Smullyan. *First-Order Logic*. Springer Verlag, 1968.

[145] W. Snyder and C. Lynch. Goal-directed strategies for paramodulation. In R.V. Book, editor, *Rewriting Techniques and Applications*, volume 488 of *Lecture Notes in Computer Science*, pages 150–161, 1991.

[146] M. Stickel. Automated deduction by theory resolution. *Journal of Automated Reasoning*, 1:333–355, 1985.

[147] T. Tammet. A resolution theorem prover for intuitionistic logic. In M.A. McRobbie and J.K. Slaney, editors, *Automated Deduction — CADE-13*, volume 1104 of *Lecture Notes in Computer Science*, pages 2–16, New Brunswick, NJ, USA, 1996.

[148] M. Veanes. Uniform representation of recursively enumerable sets with simultaneous rigid E-unification. UPMAIL Technical Report 126, Uppsala University, Computing Science Department, 1996.

[149] M. Veanes. *On Simultaneous Rigid E-Unification.* PhD thesis, Uppsala University, 1997.

[150] M. Veanes. The undecidability of simultaneous rigid E-unification with two variables. In G. Gottlob, A. Leitsch, and D. Mundici, editors, *Computational Logic and Proof Theory. 5th Kurt Gödel Colloquium, KGC'97*, volume 1289 of *Lecture Notes in Computer Science*, pages 305–318, Vienna, Austria, 1997.

[151] M. Veanes. The relation between second-order unification and simultaneous rigid E-unification. Technical Report MPI-I-98-2-005, Max-Planck Institut für Informatik, Saarbrücken, February 1998.

[152] P.J. Voda and J. Komara. On Herbrand skeletons. Technical report, Institute of Informatics, Comenius University Bratislava, July 1995.

[153] A. Voronkov. LISS - the logic inference search system. In Mark Stickel, editor, *Proc. 10th Int. Conf. on Automated Deduction*, volume 449 of *Lecture Notes in Computer Science*, pages 677–678, Kaiserslautern, Germany, 1990. Springer Verlag.

[154] A. Voronkov. A proof-search method for the first order logic. In P. Martin-Löf and G. Mintz, editors, *COLOG'88*, volume 417 of *Lecture Notes in Computer Science*, pages 327–340. Springer Verlag, 1990.

[155] 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.

[156] A. Voronkov. On proof-search in intuitionistic logic with equality, or back to simultaneous rigid E-Unification. UPMAIL Technical Report 121, Uppsala University, Computing Science Department, January 1996.

[157] A. Voronkov. Proof search in intuitionistic logic based on constraint satisfaction. In P. Miglioli, U. Moscato, D. Mundici, and M. Ornaghi, editors, *Theorem Proving with Analytic Tableaux and Related Methods. 5th International Workshop, TABLEAUX '96*, volume 1071 of *Lecture Notes in Artificial Intelligence*, pages 312–329, Terrasini, Palermo Italy, May 1996.

[158] A. Voronkov. Proof-search in intuitionistic logic based on the constraint satisfaction. UPMAIL Technical Report 120, Uppsala University, Computing Science Department, January 1996. Updated March 11, 1996.

[159] A. Voronkov. Proof search in intuitionistic logic with equality, or back to simultaneous rigid E-unification. In M.A. McRobbie and J.K. Slaney, editors, *Automated Deduction — CADE-13*, volume 1104 of *Lecture Notes in Computer Science*, pages 32–46, New Brunswick, NJ, USA, 1996.

[160] A. Voronkov. Herbrand's theorem, automated reasoning and semantic tableaux. UPMAIL Technical Report 151, Uppsala University, Computing Science Department, February 1998.

[161] A. Voronkov. Simultaneous rigid E-unification and other decision problems related to Herbrand's theorem. UPMAIL Technical Report 152, Uppsala University, Computing Science Department, February 1998.

[162] H. Wang. Towards mechanical mathematics. *IBM J. of Research and Development*, 4:2–22, 1960.

[163] L. Wos, R. Overbeek, and E. Lusk. Subsumption, a sometimes undervalued procedure. In J.-L. Lassez and G. Plotkin, editors, *Computational Logic. Essays in Honor of Alan Robinson.*, pages 3–40. The MIT Press, Cambridge, MA, 1991.

[164] L. Wos, G. Robinson, D. Carson, and L. Shalla. The concept of demodulation in theorem proving. *Journal of the Association for Computing Machinery*, 14:698–709, 1967.

[165] H. Zhang and D. Kapur. First-order theorem proving using conditional rewrite rules. In E. Lusk and R. Overbeek, editors, *9th International Conference on Automated Deduction*, volume 310 of *Lecture Notes in Computer Science*, pages 1–20. Springer Verlag, 1988.

Notation for calculi and rules

Index