A Completion-Based Method for Mixed Universal and Rigid *E*-Unification

Bernhard Beckert

Universität Karlsruhe
Institut für Logik, Komplexität und Deduktionssysteme
76128 Karlsruhe, Germany
beckert@ira.uka.de Tel. +49-721-608-4324

Abstract. We present a completion-based method for handling a new version of E-unification, called "mixed" E-unification, that is a combination of the classical "universal" E-unification and "rigid" E-unification. Rigid E-unification is an important method for handling equality in Gentzen-type first-order calculi, such as free-variable semantic tableaux or matings. The performance of provers using E-unification can be increased considerably, if mixed E-unification is used instead of the purely rigid version. We state soundness and completeness results, and describe experiments with an implementation of our method.

1 Introduction

We present a completion-based method for handling a new version of E-unification called "mixed" E-unification, that is a combination of the classical "universal" E-unification and "rigid" E-unification [9]. There has been a growing interest in rigid E-unification, because it is an important method for handling equality in Gentzen-type first-order calculi, such as free-variable semantic tableaux [4] or matings [9, 15]. The performance of provers using E-unification for handling equality can be increased considerably, if mixed E-unification is used instead of the purely rigid version [4].[1]

The Unfailing Knuth-Bendix-Algorithm (UKBA) [13, 1] with narrowing [14], that is generally considered to be the best algorithm for universal E-unification and has often been implemented, cannot be used to solve rigid or mixed problems. Completion-based methods for rigid E-unification have been described in [9, 10]. These, however, are non-deterministic and unsuited for implementation (they have, in fact, never been implemented). In [4] a method for solving mixed E-unification problems has been introduced that does not use completion but is based on computing equivalence classes.

[1] An equality has often to be applied more than once in a proof, each time with different substitutions for the variables occurring in it. In Gentzen-type calculi the mechanism to do so is to generate several instances of the equality. It is, however, often possible to recognize equalities that are "universal" w.r.t. variables they contain (e.g. equalities that occur on only one branch of a tableau). If *mixed* E-unification is used, this knowledge can be used to avoid generating additional copies of equalities.

The basic idea of our approach—and the main difference to the classical unfailing completion procedure—is that during the completion process *free* variables are never renamed, even if equalities that have variables in common are applied to each other. In addition, constraints consisting of a substitution and an *order condition* are attached to the reduction rules and terms.[2]

The paper is organized as follows: In the next section we give some basic definitions and notations; the different versions of E-unification are described in Section 3. In Sections 4 to 8 we develop our new method for mixed E-unification. Completeness and correctness results are presented in Section 9; the implementation of our method and experiments are described in Sections 10 and 11. Finally, we draw conclusions from our research in Section 12. We assume familiarity with completion-based methods [1] and (universal) E-unification [12].

2 Preliminaries

We use sequences of natural numbers to denote positions in terms; t_p is the subterm at position p in the term t (e.g. $f(a,b)_{\langle 2\rangle} = b$). The equality predicate is denoted by $\approx$, such that no confusion with the meta-level equality $=$ can arise.

For the sake of simplicity and without any loss of generality, we use a slightly non-standard notion of substitutions: They have to be idempotent and of finite domain; **Subst** is the set of these substitutions. id is the empty substitution. The application of a substitution σ to a term t is denoted by $t\sigma$; if a substitution is applied to a quantified rule, equality, or term, the bound variables are never instantiated. $\leq$ denotes the specialization relation on substitutions: $\sigma \leq \tau$ iff there is a σ' such that $\sigma' \circ \sigma = \tau$.

The reduction ordering $\succ_{\text{LPO}}$ on terms is an arbitrary but fixed *lexicographic path ordering* (LPO) [7] that is total on ground terms.[3]

3 Universal, Rigid and Mixed *E*-Unification

To be able to mix rigid and universal E-unification, we have to use equalities (resp. reduction rules) containing two types of variables. To distinguish them syntactically, equalities $(\forall \bar{x})(l \approx r)$ and reduction rules $(\forall \bar{x})(l \to r)$ can be explicitly quantified w.r.t. variables they contain.[4]

Definition 1. A **mixed *E*-unification problem** $\langle E, s, t\rangle$ consists of a finite set E of equalities of the form $(\forall \bar{x})(l \approx r)$ and terms s and t.[5]

[2] In [5] a similar type of constraints is used for E-unification—but only for its purely universal version. In [4] substitutions are used to restrict the validity of terms. For a completion-based approach, however, this is not sufficient because the validity of reduction rules depends on the ordering on terms.

[3] Other reduction orderings can be used, provided the satisfiability of constraints (Def. 6) is still decidable.

[4] $(\forall \bar{x})$ is an abbreviation for $(\forall x_1)\cdots(\forall x_n)$.

[5] Without making a real restriction, we require the sets of bound and free variables in the problem to be disjoint.

A substitution σ is a **solution** to the problem, iff $E\sigma \models (s\sigma \approx t\sigma)$, where the free variables in $E\sigma$ are "held rigid", i.e. treated as constants.

The major differences between this definition and that generally given in the (extensive) literature on (universal) E-unification are: (i) the equalities in E are *explicitly* quantified (instead of considering all the variables in E to be *implicitly* universally quantified); (ii) in difference to the "normal" notion of logical consequence, free variables in $E\sigma$ are "held rigid"; (iii) the substitution σ is applied not only to the terms s und t but as well to the set E.

Definition 2. An E-unification problem $\langle E, s, t\rangle$ is **purely universal** iff there are no free variables in E; it is **purely rigid** iff there are no bound variables in E.[6]

The intention of defining different versions of E-unification is to allow the equalities in $E\sigma$ to be used differently in a proof for $E\sigma \models (s\sigma \approx t\sigma)$: in the purely universal case the equalities can be applied several times with different instantiations for the variables they contain; in the purely rigid case they can be applied more than once but with only one instantiation for each variable x (namely $\sigma(x)$); in the mixed case there are both types of variables. The following table shows some simple examples:

E	s	t	MGUs	Type
$\{f(x) \approx x\}$	$f(a)$	a	$\{x/a\}$	purely rigid
$\{f(a) \approx a\}$	$f(x)$	a	$\{x/a\}$	ground
$\{(\forall x)(f(x) \approx x)\}$	$g(f(a), f(b))$	$g(a, b)$	id	purely universal
$\{f(x) \approx x\}$	$g(f(a), f(b))$	$g(a, b)$	—	purely rigid
$\{(\forall x)(f(x, y) \approx f(y, x))\}$	$f(a, b)$	$f(b, a)$	$\{y/b\}$	mixed

The following well known [17, 9] feature of purely rigid and purely universal E-unification can be proven to be valid for mixed E-unification as well: Supposed the substitution σ is a solution to the mixed E-unification problem $\langle E, s, t\rangle$, then every specialization τ of σ is a solution to $\langle E, s, t\rangle$ as well.

Since our aim is to find *most general* unifiers (MGUs), a subsumption relation on substitutions has to be defined. One could use the specialization relation $\leq$. But, for solving mixed E-unification problems, the subsumption relation $\leq_E$ is better suited:[7]

Definition 3. Let E be a set of equalities. The subsumption relation $\leq_E$ is defined on the set of substitutions by: $\sigma \leq_E \tau$ iff there is a substitution σ' such that $E\tau \models (\sigma' \circ \sigma)(x) \approx \tau(x)$ for all variables x, where the free variables in $E\tau$ are held rigid.

The intuitive meaning of $\sigma \leq_E \tau$ is: there is a specialization of σ that can be derived from τ by applying equalities from $E\tau$. In contrary to $\leq$ the relation $\leq_E$ depends on the set E of equalities.

[6] If E is ground, the problem is both purely rigid and purely universal.

[7] A similar subsumption relation—for purely rigid problems—has been defined in [9].

Since purely universal E-unification is already undecidable, mixed E-unification is—in general—undecidable as well. It is, however, possible to enumerate a complete set of MGUs. Purely rigid E-unification is NP-complete and, therefore, decidable [8].[8]

Often, in particular for applications in automated theorem proving, several E-unification problems have to be solved simultaneously:

Definition 4. A finite set $\{\langle E_1, s_1, t_1\rangle, \ldots, \langle E_n, s_n, t_n\rangle\}$ of mixed E-unification problems ($n \geq 1$) is called **simultaneous** E-unification problem.

A substitution σ is a solution to the simultaneous problem iff it is a solution to every component $\langle E_k, s_k, t_k\rangle$ ($1 \leq k \leq n$).

A simultaneous E-unification problem can be solved by searching for common specializations of solutions to its components.[9]

4 Constraints

For different substitutions σ, the completion of $E\sigma$ contains different reduction rules. Nevertheless, a single completion can be computed for all $E\sigma$, if constraints are attached to the rules to restrict their validity to certain (sets of) substitutions.

The first part of the constraints we attach to reduction rules and terms is an *order condition*[10]; it expresses a restriction on the ordering of terms w.r.t. the LPO used.

Example 1. A reduction system equivalent to $E\sigma = \{x \approx y\}\sigma$ either consists of the rule $(x \to y)\sigma$ or the rule $(y \to x)\sigma$, depending on which of the terms $\sigma(x)$ and $\sigma(y)$ is greater w.r.t. the LPO used.

The expression $x \succ y$ is the natural choice for a restriction such as "the term substituted for x has to be greater than that substituted for y":

Definition 5. Order conditions are composed of the **atomic** order conditions $s \succ t$ (s and t are terms) using the logical connectives $\neg$, $\wedge$, $\vee$ and $\supset$, and the constants *true* and *false*.

Ground order conditions, i.e., order conditions that contain no variables, are assigned a truth value by interpreting the (predicate) symbol $\succ$ by a (fixed) LPO.

A (non-ground) order condition O is **true** iff $O\sigma$ is true for all ground substitutions σ, **false** (or inconsistent) iff its negation $\neg O$ is true, and **consistent** iff it is not false.

Since LPOs are total on ground terms, the truth value of ground order conditions is well defined; non-ground order conditions are (similar to first order formulas) either consistent or inconsistent, and may be true or false.

[8] In [9] a proof is given that is based on a non-deterministic algorithm for computing a *finite* set of MGUs that is complete w.r.t. $\leq_E$.

[9] Simultaneous *purely rigid* E-unification is decidable [10].

[10] These are similar to the "constraints" in [16], but there are some differences.

Example 2. The order condition $f(a) \succ a$ is true; $(x \succ y) \wedge (y \succ x)$ is false; and $x \succ y$ is consistent. The truth value of $a \succ b$ depends on the LPO used to interpret $\succ$.

In some cases, order conditions are not sufficient for describing the set of substitutions for which a reduction rule is valid:

Example 3. Suppose $E = \{f(b) \approx a,\ f(x) \approx c\}$; the reduction rule $c \to a$ is part of the completion of $E\sigma$ iff $\sigma(x) = b$ (then the equalities are a critical pair).

One could use the formula $x \approx t$ to express conditions of the form "x has to be substituted by (an instance of) t", if the predicate symbol $\approx$ were allowed in order conditions. That, however, would make the handling of conditions unnecessarily complicated. Instead the substitution $\{x/t\}$ itself becomes part of the constraint:

Definition 6. A **constraint** $c = \langle \sigma, O \rangle$ consists of a substitution σ and an order condition O such that the variables in the domain of σ do not occur in O, i.e. $O = O\sigma$.

A substitution τ **satisfies** a constraint $c = \langle \sigma, O \rangle$ iff τ is a specialization of σ and $O\tau$ is true. τ satisfies a set C of constraints iff there is a $c \in C$ satisfied by τ.

$\mathrm{Sat}(c)$ (resp. $\mathrm{Sat}(C)$) is the set of substitutions satisfying the constraint c (resp. the set C of constraints).

Note, that sets of constraints implicitly represent *disjunctions*. To simplify the handling of constraints, we give some additional definitions and notations:

Definition 7. A constraint c_1 **subsumes** a constraint c_2 iff the substitutions satisfying c_2 satisfy as well c_1: $\mathrm{Sat}(c_2) \subset \mathrm{Sat}(c_1)$.

A constraint c^{-1} that is satisfied by the substitutions *not* satisfying c is called **negation** of c: $\mathrm{Sat}(c^{-1}) = \mathbf{Subst} \setminus \mathrm{Sat}(c)$.

A constraint $c_1 \sqcap c_2$ that is satisfied by the constraints satisfying both c_1 and c_2 is called a **combination** of c_1 and c_2: $\mathrm{Sat}(c_1 \sqcap c_2) = \mathrm{Sat}(c_1) \cap \mathrm{Sat}(c_2)$.

The *empty constraint* $\epsilon = \langle id, true \rangle$ consists of the empty substitution id and the order condition $true$; it is satisfied by all substitutions.

There are efficient algorithms for computing negations and combinations of constraints. Since a constraint c_1 subsumes a constraint c_2 iff $c_1^{-1} \sqcap c_2$ is inconsistent, these are an important part of the implementation. Deciding whether a constraint is satisfiable is NP-hard [6]. The problem can however simplified considerably: The order condition $(s \succ x) \wedge (x \succ t)$ is inconsistent if there is no term between s and t (w.r.t. the LPO used). Without causing any harm, we can do without checking for such inconsistencies, that are very difficult to detect.

5 Constrained Terms and Reduction Rules

Since—syntactically—constrained reduction rules can be considered to be constrained terms,[11] it suffices to define the latter:

[11] Over a different signature that contains $\to$ as a function symbol.

Definition 8. A **constrained term** $\mathbf{t} = (\forall \bar{x})(t \ll c)$ is a term t with a constraint $c = \langle \sigma, O \rangle$ attached to it such that $t\sigma = t$.[12] It can be universally quantified w.r.t. some or all of the variables it contains (the quantification includes the constraint).

On first sight quantified terms may look strange, but, later on, a constrained term $\mathbf{t}$ is used to express the fact that it can be derived from another term $\mathbf{t}'$. Therefore, it is important to be able to make a distinction between rigid and non-rigid (quantified) variables.

Using constraints, for every equality an equivalent set of reduction rules can be constructed; even for those that cannot be oriented without constraints.

Example 4. The equality $f(x) \approx g(y)$ cannot be oriented without constraints, since (i) its instance $f(g(a)) \approx g(a)$ has to be oriented from left to right, while (ii) its instance $f(a) \approx g(f(a))$ has to be oriented from right to left. The *constrained* rules $f(x) \to g(y) \ll \langle id, f(x) \succ g(y) \rangle$ and $g(y) \to f(x) \ll \langle id, g(y) \succ f(x) \rangle$, however, define the same derivability relation as the equality $f(x) \approx g(y)$.

Other typical examples are the pair of constrained rules $x \to y \ll \langle id, x \succ y \rangle$ and $y \to x \ll \langle id, y \succ x \rangle$, that corresponds to the equality $x \approx y$; and the constrained rule $(\forall x)(\forall y)(f(x,y) \to f(y,x) \ll \langle id, f(x,y) \succ f(y,x) \rangle)$, that is equivalent to $(\forall x)(\forall y)(f(x,y) \approx f(y,x))$.

The possibility to orient every equality justifies the following definition, that assigns to each set of equalities a constrained reduction system. Since it will be the starting point of the completion process, it is called the initial system:

Definition 9. Let E be a set of equalities. Then

$$\{(\forall \bar{x})(s \to t \ll \langle id, s \succ t \rangle) \mid (\forall \bar{x})(s \approx t) \in E \text{ or } (\forall \bar{x})(t \approx s) \in E\}$$

is the **initial** constrained reduction system assigned to E.

A constrained reduction system $\mathcal{R}$ defines derivability relations $\Rightarrow_{\mathcal{R}}$ and $\Rrightarrow_{\mathcal{R}}$ on the set of constrained terms:

Definition 10. Let $\mathcal{R}$ be a constrained reduction system and $\mathbf{t} = (\forall \bar{x})(t \ll c_t)$ a constrained term. Iff there is a rule $\mathbf{r} = (\forall \bar{y})(l \to r \ll c_r)$ in $\mathcal{R}$, such that

1. $\{x_1, \ldots, x_n\} \cap \mathrm{Var}(r) = \emptyset$ and $\{y_1, \ldots, y_m\} \cap \mathrm{Var}(t) = \emptyset$,[13]
2. p is a position in t where $t_{|p}$ is not a variable unless $t_{|p} = l = x_i$,
3. $t_{|p}$ and l are (syntactically) unifiable with an MGU ν,
4. the combination $c_{new} = \langle \mu, O_{new} \rangle = c_t \sqcap c_r \sqcap \langle \nu, true \rangle$ is consistent,

then $\mathbf{t} \Rightarrow_{\mathcal{R}} \mathbf{t}'$, where $\mathbf{t}' = (\forall \bar{x})(\forall \bar{y})((t[p/r])\mu \ll c_{new})$.[14]

Iff in addition (i) $t_{|p} = l\mu$, and (ii) c_{new} subsumes c_t, then $\mathbf{t} \Rrightarrow_{\mathcal{R}} \mathbf{t}'$. We call the triple $\langle \mathbf{r}, p, \mu \rangle$ a justification for $\mathbf{t} \Rightarrow_{\mathcal{R}} \mathbf{t}'$ (resp. $\mathbf{t} \Rrightarrow_{\mathcal{R}} \mathbf{t}'$).

[12] The symbol $\ll$ means "if".

[13] This is not a real restriction, since the bound variables can be renamed.

[14] If the constraint c_{new} expresses restrictions on *bound* variables that do not occur in $t[p/r]$, these restrictions can be omitted. For example, $(\forall x)(a \to b \ll \langle id, x \succ c \rangle)$ can be reduced to $a \to b \ll \epsilon$.

The intuitive meaning of $(\forall\bar{x})(s \ll c_s) \Rightarrow_{\mathcal{R}} (\forall\bar{y})(t \ll c_t)$ is: there is a substitution σ such that $t\sigma$ can be derived from $s\sigma$ using a rule from $\mathcal{R}$, and σ satisfies the constraints c_s, c_t and that attached to the rule.

The main difference between the two derivability relations $\Rightarrow_{\mathcal{R}}$ and $\Rrightarrow_{\mathcal{R}}$ (which is a sub-relation of $\Rightarrow_{\mathcal{R}}$) is that the derivation $\mathbf{t} \Rrightarrow_{\mathcal{R}} \mathbf{t}'$ is "reversible", if the order on terms is not taken into concern. The derived term $\mathbf{t}'$ can—in combination with the rules in $\mathcal{R}$—take on the functions of $\mathbf{t}$. In contrary to that, a derivation $\mathbf{t} \Rightarrow_{\mathcal{R}} \mathbf{t}'$ is "irreversible" (provided $\mathbf{t} \not\Rrightarrow_{\mathcal{R}} \mathbf{t}'$).

Example 5. Some examples for derivations and their justification:

$$\begin{array}{rll}
(g(a,c) \ll \epsilon) \Rrightarrow (g(a,b) \ll \epsilon) & - & \langle (c \to b \ll \epsilon),\, \langle 2\rangle,\, id\rangle \\
(f(c) \ll \epsilon) \Rrightarrow (c \ll \epsilon) & - & \langle ((\forall x)(f(x) \to x \ll \epsilon),\, \langle\rangle,\, id\rangle \\
(a \ll \epsilon) \Rrightarrow (y \ll \langle\{x/a\}, a \succ y\rangle) & - & \langle (x \to y \ll \langle id, x \succ y\rangle),\, \langle\rangle,\, \{x/a\}\rangle \\
(f(c) \ll \epsilon) \Rrightarrow (c \ll \langle\{x/c\}, true\rangle) & - & \langle (f(x) \to x \ll \epsilon),\, \langle\rangle,\, \{x/c\}\rangle
\end{array}$$

It is useful to define a subsumption relation on constrained terms. It is similar to the relation between a term (without constraint) and its instances:

Definition 11. A constrained term $\mathbf{t}_1 = (\forall\bar{x})(t_1 \ll c_1)$ **subsumes** a constrained term $\mathbf{t}_2 = (\forall\bar{y})(t_2 \ll c_2)$, iff (i) t_2 is an instance of t_1, and (ii) the combination $c_1 \sqcap \langle\mu, true\rangle$ subsumes the constraint c_2 (Def. 7).

Example 6. The constrained term $a \ll \epsilon$ subsumes $a \ll \langle\{x/a\}, true\rangle$.

If $b \succ_{\text{LPO}} a$, then the constrained rule $x \to a \ll \langle id, x \succ a\rangle$ subsumes the rule $b \to a \ll \langle\{x/b\}, true\rangle$.

6 Completion of Constrained Reduction Systems

6.1 Goal of the Completion

The following transformation rules define a method for completing constrained reduction systems. If this rules are applied repeatedly (in a fair way) to an initial system $\mathcal{R} = \mathcal{R}^0$, a system $\mathcal{R}^\infty$ is approximated. It represents the (classical) completions of all the different instances of E.

In general, the instances of $\mathcal{R}^\infty$ will not be irreducible and, therefore, not canonical. Nevertheless, the relation $\Rightarrow_{\mathcal{R}^\infty}$ will be confluent (in a sense clarified in Lemma 19), and thus have the feature crucial for computing normal forms of constrained terms and solving E-unification problems.

The following example shows that it would not make sense to expect the instances to be canonical:

Example 7. None of the transformation rules introduced in the next section can be applied to the reduction system $\mathcal{R}^\infty = \{f(x) \to c \ll \epsilon,\ a \to b \ll \epsilon\}$. Nevertheless, its instance $\{f(a) \to c,\ a \to b\}$ is not canonical, since it can be simplified to $\{f(b) \to c,\ a \to b\}$.

6.2 The Transformation Rules

The rules that have to be applied to complete a reduction system are presented in form of transformation rules.[15]

Deletion A rule that has an inconsistent constraint attached to it can be removed, because it cannot be applied anyway:

$$\textbf{(Del)}\quad \frac{\mathcal{R} \cup \{(\forall \bar{x})(s \to t \ll c)\}}{\mathcal{R}} \quad c \text{ inconsistent}$$

Example 8. The rule $x \to f(x) \ll \langle id, x \succ f(x)\rangle$ can be deleted, because its constraint is inconsistent.

Subsumption A constrained rule that is subsumed by another rule (Def. 11) can be removed:

$$\textbf{(Sub)}\quad \frac{\mathcal{R} \cup \{\mathbf{r}, \mathbf{r}'\}}{\mathcal{R} \cup \{\mathbf{r}\}} \quad \mathbf{r} \text{ subsumes } \mathbf{r}'$$

Equivalence Transformation A constraint c attached to a reduction rule can be replaced by a set $\{c_1, \ldots, c_n\}$ of constraints that—disjunctively connected—are equivalent to c (i.e. $\mathrm{Sat}(c) = \bigcup_{1 \le i \le n} \mathrm{Sat}(c_i)$). Since only a single constraint can be attached to a rule, n copies of the original rule are generated:

$$\textbf{(Equ)}\quad \frac{\mathcal{R} \cup \{(\forall \bar{x})(l \to r \ll c)\}}{\mathcal{R} \cup \{(\forall \bar{x})(l\sigma \to r\sigma \ll \langle \sigma, O\rangle) \mid \langle \sigma, O\rangle \in C\}} \quad \begin{array}{l}\mathrm{Sat}(c) = \mathrm{Sat}(C),\\ C \text{ finite}\end{array}$$

Though this equivalence rule is not necessary for the completeness of our method, it is very useful; it allows to transform constraints into a normal form, and thus simplify their handling significantly.

Example 9. The rule $f(x,y) \to f(a,b) \ll \langle id, f(x,y) \succ f(a,b)\rangle$ can be replaced by $f(x,y) \to f(a,b) \ll \langle id, x \succ a\rangle$ and $f(a,y) \to f(a,b) \ll \langle \{x/a\}, y \succ b\rangle$.

Critical Pair Rule, Combination, Simplification The transformation rules described so far allow to delete rules or to replace them by new ones without using the derivability relation $\Rightarrow_{\mathcal{R}}$. But, to complete a reduction system, $\Rightarrow_{\mathcal{R}}$ has to be taken into concern by applying one rule $\mathbf{r}_2 \in \mathcal{R}$ to another rule $\mathbf{r}_1 \in \mathcal{R}$. Suppose $\mathbf{r}_1 = (\forall \bar{x})(s \to t \ll c_1)$, $\mathbf{r}_2 = (\forall \bar{y})(l \to r \ll c_2)$, and the rule $\mathbf{r}_2$ can be applied to $\mathbf{r}_1$ to derive the rule $\mathbf{r}'_1 = (\forall \bar{x})(\forall \bar{y})(s_{new} \to t_{new} \ll c_{new})$, i.e., $\mathbf{r}_1 \Rightarrow \mathbf{r}'_1$ with a justification $\langle \mathbf{r}_2, p, \mu\rangle$. We cannot just add the new rule $\mathbf{r}'_1$ to $\mathcal{R}$: Firstly, instances of $\mathbf{r}'_1$ may be oriented differently; we therefore have to use the two symmetrical versions

$$\begin{aligned}\mathbf{r}_{new1} &= (\forall \bar{x})(\forall \bar{y})(s_{new} \to t_{new} \ll c_{new} \sqcap \langle id, s_{new} \succ t_{new}\rangle)\\ \mathbf{r}_{new2} &= (\forall \bar{x})(\forall \bar{y})(t_{new} \to s_{new} \ll c_{new} \sqcap \langle id, t_{new} \succ s_{new}\rangle)\ .\end{aligned}$$

[15] The set of constrained rules below the line can be derived from the set above the line if the conditions on the right are met.

Secondly, the form of the transformation rule depends on whether (i) $\mathbf{r}_1 \Rrightarrow \mathbf{r}'_1$ (besides $\mathbf{r}_1 \Rightarrow \mathbf{r}'_1$) or not,[16] and (ii) which side of $\mathbf{r}_1$ the rule $\mathbf{r}_2$ has been applied to, i.e., whether p is a position in s or in t.

If $\mathbf{r}_1 \Rrightarrow \mathbf{r}'_1$, then $\mathbf{r}_{new1}$ and $\mathbf{r}_{new2}$ allow—together with $\mathbf{r}_2$—all the derivations possible with $\mathbf{r}_1$. If, in addition, $\mathbf{r}_2$ has been applied to the right side of $\mathbf{r}_1$, one can conclude that the constraint attached to $\mathbf{r}_{new2}$ is inconsistent. In that case the transformation is called *simplification* (Sim), since $\mathbf{r}_1$ can be replaced by the single new rule $\mathbf{r}_{new1}$:

$$\textbf{(Sim)} \quad \frac{\mathcal{R}}{(\mathcal{R} \setminus \{\mathbf{r}_1\}) \cup \{\mathbf{r}_{new1}\}} \quad p \text{ in } t,\ \mathbf{r}_1 \Rrightarrow_{\mathcal{R}} \mathbf{r}'_1$$

Else, if $\mathbf{r}_2$ has been applied to the left side of $\mathbf{r}_1$, the rule $\mathbf{r}_{new2}$ cannot be left out, because the constraint attached to it may be consistent. Such a transformation is called *composition* (Com).

$$\textbf{(Com)} \quad \frac{\mathcal{R}}{(\mathcal{R} \setminus \{\mathbf{r}_1\}) \cup \{\mathbf{r}_{new1}, \mathbf{r}_{new2}\}} \quad p \text{ in } s,\ \mathbf{r}_1 \Rrightarrow_{\mathcal{R}} \mathbf{r}'_1$$

If $\mathbf{r}_1 \not\Rrightarrow \mathbf{r}'_1$, the new rules cannot replace the old rule $\mathbf{r}_1$; it cannot be removed. Nevertheless, the transformation has to be carried out provided $\mathbf{r}_2$ has been applied to the left side of $\mathbf{r}_1$. Then $\mathbf{r}_1$ and $\mathbf{r}_2$ are a *critical pair*, and the new rules are needed to make the reduction system confluent:

$$\textbf{(CP)} \quad \frac{\mathcal{R}}{\mathcal{R} \cup \{\mathbf{r}_{new1}, \mathbf{r}_{new2}\}} \quad p \text{ in } s,\ \mathbf{r}_1 \not\Rrightarrow_{\mathcal{R}} \mathbf{r}'_1$$

In difference to the critical pair rule defined in [9] the unifier μ is only applied locally to the new rules (not to the whole system $\mathcal{R}$).

Example 10. Suppose $f \succ_{\text{LPO}} c \succ_{\text{LPO}} b \succ_{\text{LPO}} a$, and $\mathcal{R}$ contains the constrained reduction rules

$$\begin{array}{lcllcl} \mathbf{r}_1 & = & f(c) \to b \ll \epsilon & \mathbf{r}_3 & = & (\forall x)(f(x) \to y \ll \langle id, f(x) \succ y\rangle) \\ \mathbf{r}_2 & = & b \to a \ll \epsilon & \mathbf{r}_4 & = & f(x) \to y \ll \langle id, f(x) \succ y\rangle \end{array}$$

The simplification rule (Sim) can be applied to $\mathbf{r}_1$ and $\mathbf{r}_2$ to replace $\mathbf{r}_1$ by the single new rule $f(c) \to a \ll \epsilon$.

The composition rule (Com) can be applied to $\mathbf{r}_1$ and $\mathbf{r}_3$ to replace $\mathbf{r}_1$ by $y \to b \ll \langle id, (f(c) \succ y \wedge y \succ b)\rangle$ and $b \to y \ll \langle id, (f(c) \succ y \wedge b \succ y)\rangle$.

The critical pair rule (CP) can be applied to $\mathbf{r}_1$ and $\mathbf{r}_4$ (note, that in $\mathbf{r}_4$ the variable x is not quantified); the new rules $y \to b \ll \langle \{x/c\}, (f(c) \succ y \wedge y \succ b)\rangle$ and $b \to y \ll \langle \{x/c\}, (f(c) \succ y \wedge b \succ y)\rangle$ have to be added.

6.3 Fair Completion Procedures

In general, an infinite number of transformation steps can be necessary to complete a reduction system. But even if the computation does not terminate, a completion $\mathcal{R}^\infty$ is approximated, consisting of the *persistent* reduction rules, that occur in all but a finite number of the resulting system. To generate a confluent reduction systems, certain fairness conditions have to be met:

[16] That is, $\mathbf{r}_1 \Rrightarrow \mathbf{r}'_1$ with the same justification as $\mathbf{r}_1 \Rightarrow \mathbf{r}'_1$; whether $\mathbf{r}_1 \Rrightarrow \mathbf{r}'_1$ with a different justification is not relevant.

Definition 12. $\mathcal{R} \vdash \mathcal{R}'$ means that the constrained reduction system $\mathcal{R}'$ can be derived from $\mathcal{R}$ by applying one of the transformation rules from Section 6.2.

A **transformation procedure** specifies, when supplied with an initial reduction system $\mathcal{R}^0$, in which way (in particular: in which order) the transformation rules are to be applied to generate a sequence $\mathcal{R}^0 \vdash \mathcal{R}^1 \vdash \mathcal{R}^2 \vdash \cdots$ of reduction systems. Then, the reduction system

$$\mathcal{R}^\infty = \begin{cases} \mathcal{R}^m & \text{if the sequence is of length } m \\ \bigcup_{k\geq 0} \bigcap_{m\geq k} \mathcal{R}^m & \text{if the sequence is infinite} \end{cases}$$

is called the **completion** of $\mathcal{R} = \mathcal{R}^0$, and the completion of the set E of equalities if $\mathcal{R}$ is the initial system for E.

A transformation procedure is **fair** provided:

1. There is no infinite sequence $(\mathbf{r}_i)_{i\geq 0} \subset \bigcup_{m\geq k} \mathcal{R}^m$ such that for all $i \geq 0$ the rule $\mathbf{r}_{i+1}$ has been derived from $\mathbf{r}_i$ by an equivalence transformation.
2. There is no infinite sequence $(\mathbf{r}_i)_{i\geq 0} \subset \bigcup_{m\geq k} \mathcal{R}^m$ such for all $i \geq 0$ the rule $\mathbf{r}_{i+1}$ subsumes $\mathbf{r}_i$, and $\mathbf{r}_i$ has therefore been removed.
3. For every persistent critical pair $\mathbf{r}_1, \mathbf{r}_2 \in \mathcal{R}^\infty$ there is an $i \geq 0$ such that $\mathcal{R}^{i+1}$ has been derived by applying the critical pair transformation rule to $\mathbf{r}_1, \mathbf{r}_2 \in \mathcal{R}^i$.

The first two fairness conditions are of a more technical nature: Condition 1 avoids infinite sequences of equivalence transformations. Condition 2 assures that, if there is an infinite sequence of rules subsuming each other, at least one of them is in the completion $\mathcal{R}^\infty$.

Condition 3 is the most important: it assures the application of the critical pair transformation rule to all persistent critical pairs. It is essential for achieving confluence of the completion.

Provided, the above fairness conditions are met, arbitrary heuristics can be used to choose the next transformation rule to apply.

7 Computing Normal Forms

7.1 Normalization Rules

Using constrained reduction systems and terms, a term has more than one normal form—in general an infinite number of them.

Example 11. With $\mathcal{R}^\infty = \{b \to a \ll \epsilon,\ d \to c \ll \epsilon\}$ the constrained term $x \ll \epsilon$ has three normal forms: $a \ll \langle\{x/b\}, \mathit{true}\rangle$, $c \ll \langle\{x/d\}, \mathit{true}\rangle$, and $x \ll \epsilon$ itself.

The above example shows that there can be redundancies in a set of normal forms: the validity of $x \ll \epsilon$ is not restricted to substitutions σ such that $\sigma(x) \neq a$ and $\sigma(x) \neq b$.

The computation of normal forms is—similar to the completion procedure—presented in form of transformation rules operating on sets of constrained terms:

Definition 13. To compute the normal forms of a set $\mathcal{T}$ of constrained terms, the rules **deletion** (Del), **equivalence** (Equ), **subsumption** (Sub), **simplification** (Sim), and **deduction** (Ded) can be applied to $\mathcal{T}$; the rules depend on a constrained reduction system $\mathcal{R}$:

$$\textbf{(Del)} \quad \frac{\mathcal{T} \cup \{(\forall \bar{x})(t \ll c)\}}{\mathcal{T}} \qquad c \text{ inconsistent}$$

$$\textbf{(Equ)} \quad \frac{\mathcal{T} \cup \{(\forall \bar{x})(t \ll c)\}}{\mathcal{T} \cup \{(\forall \bar{x})(t\sigma \ll \langle \sigma, O\rangle) \mid \langle \sigma, O\rangle \in C\}} \qquad \begin{array}{l}\mathrm{Sat}(c) = \mathrm{Sat}(C),\\ C \text{ finite}\end{array}$$

$$\textbf{(Sub)} \quad \frac{\mathcal{T} \cup \{\mathbf{t}, \mathbf{t}'\}}{\mathcal{T} \cup \{\mathbf{t}\}} \qquad \mathbf{t} \text{ subsumes } \mathbf{t}'$$

$$\textbf{(Sim)} \quad \frac{\mathcal{T} \cup \{\mathbf{t}\}}{\mathcal{T} \cup \{\mathbf{t}'\}} \qquad \mathbf{t} \Rrightarrow_{\mathcal{R}} \mathbf{t}'$$

$$\textbf{(Ded)} \quad \frac{\mathcal{T} \cup \{\mathbf{t}\}}{\mathcal{T} \cup \{\mathbf{t}, \mathbf{t}'\}} \qquad \mathbf{t} \Rightarrow_{\mathcal{R}} \mathbf{t}',\ \mathbf{t} \not\Rrightarrow_{\mathcal{R}} \mathbf{t}'$$

7.2 Fair Normalization Procedures

As for completion, an infinite number of normalization steps can be necessary; similar fairness conditions have to be met. A set $\mathcal{T}^\infty$ of normal forms is approximated, consisting of the *persistent* terms, that occur in all but a finite number of the sets.

Definition 14. $\mathcal{T} \vdash \mathcal{T}'$ means that the set $\mathcal{T}'$ of constrained terms can be derived from $\mathcal{T}$ by applying one of the normalization rules from Definition 13.

A **normalization procedure** specifies, when supplied with an initial set $\mathcal{T}^0$ of constrained terms and a reduction system $\mathcal{R}$, in which way the rules are to be applied to generate a sequence $\mathcal{T}^0 \vdash \mathcal{T}^1 \vdash \mathcal{T}^2 \vdash \cdots$ of sets of constrained terms. Then, the set

$$\mathcal{T}^\infty = \begin{cases} \mathcal{T}^m & \text{if the sequence is of length } m \\ \bigcup_{k\geq 0} \bigcap_{m\geq k} \mathcal{T}^m & \text{if the sequence is infinite} \end{cases}$$

is called the **set of normal forms** of $\mathcal{T} = \mathcal{T}^0$ (w.r.t. $\mathcal{R}$).

A normalization procedure is **fair** provided:

1. There is no infinite sequence $(\mathbf{t}_i)_{i\geq 0} \subset \bigcup_{m\geq k} \mathcal{T}^m$ such that for all $i \geq 0$ the term $\mathbf{t}_{i+1}$ has been derived from $\mathbf{t}_i$ by an application of equivalence (Equ).
2. There is no infinite sequence $(\mathbf{t}_i)_{i\geq 0} \subset \bigcup_{m\geq k} \mathcal{T}^m$ such that for all $i \geq 0$ the term $\mathbf{t}_{i+1}$ subsumes $\mathbf{t}_i$, and $\mathbf{t}_i$ has therefore been removed.
3. For every persistent term $\mathbf{t} \in \mathcal{T}^\infty$ that a rule $\mathbf{r} \in \mathcal{R}$ can be applied to, there is an $i \geq 0$ such that $\mathcal{T}^{i+1}$ has been derived by applying $\mathbf{r}$ to $\mathbf{t} \in \mathcal{T}^i$.

The first two fairness conditions are similar to that of fair completion procedures (Def. 12). Condition 3 assures that whenever possible deduction and simplification are applied to persistent terms.

7.3 Combining Completion and Normalization

Although a completion $\mathcal{R}^\infty$ may be infinite, one has to abandon the computation of further reduction rules at a certain point, if completion and normalization of terms are separated. It is very difficult to decide when this point is reached. Therefore, it is better to combine the completion and the normalization process:

Definition 15. A **completion and normalization sequence** $(\langle \mathcal{R}^i, \mathcal{T}^i \rangle)_{i \geq 0}$ consists of constrained reduction systems $\mathcal{R}^i$ and sets $\mathcal{T}^i$ of constrained terms, where (for $i \geq 0$) either (i) $\mathcal{R}^{i+1}$ has been derived from $\mathcal{R}^i$ by applying a transformation rule (Sec. 6.2) and $\mathcal{T}^i = \mathcal{T}^{i+1}$; or (ii) $\mathcal{T}^{i+1}$ has been derived from $\mathcal{T}^i$ by applying a normalization rule (Def. 13) and $\mathcal{R}^i = \mathcal{R}^{i+1}$.

Of course, when completion and normalization are combined, the fairness conditions (Def. 12 and 14) still have to be met.

8 Solving Mixed *E*-Unification Problems

Now we can solve an arbitrary mixed E-unification problem $\langle E, s, t \rangle$ by completing the initial reduction system $\mathcal{R}^0$ for E and computing the sets of normal forms of the constrained terms $s \ll \epsilon$ and $t \ll \epsilon$. Using these normal forms, sets $\mathcal{C}^i$ of constraints can be computed that are satisfied by solutions to the unification problem. These approximate a set $\mathcal{C}$ such that $\mathrm{Sat}(\mathcal{C})$ is a complete set of unifiers:

Definition 16. Let $\langle E, s, t \rangle$ be a mixed E-unification problem, $\mathcal{R}^0$ the initial system for E, $\mathcal{S}^0 = \{s \ll \epsilon\}$, $\mathcal{T}^0 = \{t \ll \epsilon\}$, and $(\langle \mathcal{R}^i, \mathcal{S}^i \rangle)_{i \geq 0}$ and $(\langle \mathcal{R}^i, \mathcal{T}^i \rangle)_{i \geq 0}$ fair completion and normalization procedures. Then, for $(i = 0, 1, 2, \ldots, \infty)$ the sets $\mathcal{C}^i(\langle E, s, t \rangle)$ consist of the constraints

$$\{c_1 \sqcap c_2 \sqcap \langle \mu, \mathit{true} \rangle \mid (\forall \bar{x})(r_1 \ll c_1) \in \mathcal{S}^i,\ (\forall \bar{y})(r_2 \ll c_2) \in \mathcal{T}^i,\ r_1 \text{ and } r_2 \text{ are (syntactically) unifiable with an MGU } \mu\ \}$$

$\mathcal{C}(\langle E, s, t \rangle)$ denotes their union $\bigcup_{i \geq 0} \mathcal{C}^i(\langle E, s, t \rangle)$.

9 Soundness, Completeness, Confluence

In this section we state soundness and completeness results for our method. Due to space restrictions the proofs are omitted; they can be found in [2].

Theorem 17 Soundness. *Let $\langle E, s, t \rangle$ be a mixed E-unification problem. A substitution σ satisfying one of the constraints in $\mathcal{C}(\langle E, s, t \rangle)$ (Def. 16) is a solution to $\langle E, s, t \rangle$.*

Theorem 18 Completeness. *Let $\langle E, s, t \rangle$ be a mixed E-unification problem. The set $\mathrm{Sat}(\mathcal{C}(\langle E, s, t \rangle))$ of unifiers is ground-complete w.r.t. the subsumption relation $\leq_E$ (Def. 3), i.e., for every ground unifier σ of $\langle E, s, t \rangle$ there is a substitution $\tau \in \mathrm{Sat}(\mathcal{C}(\langle E, s, t \rangle))$ such that $\tau \leq_E \sigma$.*

A ground-complete set of unifiers w.r.t. the relation $\leq$ can be computed by inverting the constrained rules in a completion $\mathcal{R}^\infty$ for E (i.e., by changing their orientation, not the validity of their constraints), and applying the inversion to the unifiers in $\mathrm{Sat}(\mathcal{C}(\langle E, s, t\rangle))$. Computing these additional solutions can be necessary—in theory—to find solutions to a simultaneous E-unification problem by combining solutions to its components. Fortunately, in practice this turns out to be very rarely the case, in particular in the semantic tableau framework.

$\stackrel{*}{\Rightarrow}_{\mathcal{R}^\infty}$ is in general not well founded. Therefore, our method is only a semi-deciding procedure for unifiability—even if the completion $\mathcal{R}^\infty$ is finite (it is an open problem, whether $\stackrel{*}{\Rightarrow}_{\mathcal{R}^\infty}$ is well founded for purely rigid E-unification problems). The following example shows that, in addition, $\stackrel{*}{\Rightarrow}_{\mathcal{R}^\infty}$ cannot be expected to be confluent:

Example 12. Supposed there are rules $f(a) \to a \ll \epsilon$ and $f(b) \to b \ll \epsilon$ in $\mathcal{R}^\infty$. Then from the constrained term $\mathbf{s} = f(x) \ll \epsilon$ terms $\mathbf{t}_1 = a \ll \langle\{x/a\}, \mathit{true}\rangle$ and $\mathbf{t}_2 = b \ll \langle\{x/b\}, \mathit{true}\rangle$ can be derived (i.e. $\mathbf{s} \Rightarrow_{\mathcal{R}^\infty} \mathbf{t}_1$ and $\mathbf{s} \Rightarrow_{\mathcal{R}^\infty} \mathbf{t}_2$).

If $\Rightarrow_{\mathcal{R}^\infty}$ were confluent, there would have to be a term derivable from both $\mathbf{t}_1$ and $\mathbf{t}_2$. That would not make any sense but contradicts soundness.

However, the derivability relation $\stackrel{*}{\Rightarrow}_{\mathcal{R}^\infty}$ can be proven to be "weak" confluent (the proof of Theorem 18 is based upon that):

Lemma 19. *If $\mathcal{R}^\infty$ is a fair completion, $\mathbf{s}$, $\mathbf{t}_1$ and $\mathbf{t}_2$ are constrained terms such that (i) $\mathbf{s} \stackrel{*}{\Rightarrow}_{\mathcal{R}^\infty} \mathbf{t}_1$ and $\mathbf{s} \stackrel{*}{\Rightarrow}_{\mathcal{R}^\infty} \mathbf{t}_2$, and (ii) the combination $c_1 \sqcap c_2$ is consistent, then there are constrained terms $\mathbf{u}_1$ and $\mathbf{u}_2$, such that (i) $\mathbf{t}_1 \stackrel{*}{\Rightarrow}_{\mathcal{R}^\infty} \mathbf{u}_1$ and $\mathbf{t}_2 \stackrel{*}{\Rightarrow}_{\mathcal{R}^\infty} \mathbf{u}_2$, and (ii) $\mathbf{u}_1$ and $\mathbf{u}_2$ have a common instance.*

10 Implementation

The completion-based method for mixed E-unification we have described, has been implemented as part of the tableau-based theorem prover ${}_3T^AP$ [3]. The implementation consists of about 2500 lines of code, written in Quintus Prolog. Besides the possibility to prove theorems from predicate logic with equality, the E-unification module can be used "stand alone" to solve simultaneous mixed E-unification problems. Complete sets of unifiers w.r.t. both $\leq$ and $\leq_E$ can be computed.[17] The experiments described in the next section have been carried out using this implementation. Upon request the source code is available from the author.

11 Experiments

If free variables occur in only one side of an equality that cannot be oriented without using constraints, a lot of different critical pairs are generated. The worst

[17] Because these sets are infinite in general, they can only be enumerated.

case are sets of equalities such as $E = \{x_1 \approx y_1, x_2 \approx y_2, x_3 \approx y_3\}$. Its completion consists of 126 reduction rules, that take more than 10 minutes[18] to compute. A similar example is $E = \{f(x) \approx g(y), f(f(u)) \approx g(g(v))\}$. The completion, consists of 16 rules and is computed in 6.1s; 19 critical pairs are generated. In practice, however, such equalities are only very rarely used to formulate theories; therefore, the problem does not occur too often.

The standard example for purely universal completion are the axioms from group theory. The completion generated (ten rules) is the same that is computed by an implementation of the UKBA; all rules have the empty constraint attached to them.

The following example, taken from [11], shows that our method can be superior to the UKBA for purely universal problems:

The equalities $(\forall x)(\forall y)(m(x,y) \approx m(y,x))$ and $(\forall x)(\forall y)(p(x,y) \approx p(y,x))$, expressing commutativity of m and p, combined with $(\forall x)(m(x,1) \approx x)$ and $m(a,b) \approx p(a,1)$ are difficult to complete using the UKBA, because the former cannot be oriented. Our implementation, however, computes the following completion, generating only two critical pairs (instead of 16):

$$\begin{aligned}
\mathbf{r}_3 &= (\forall x)(\forall y)(p(y,x) \to p(x,y) \ll \langle id, y \succ x\rangle)\\
\mathbf{r}_4 &= (\forall x)(m(x,1) \to x \ll \epsilon)\\
\mathbf{r}_6 &= (\forall x)(\forall y)(m(y,x) \to m(x,y) \ll \langle id, y \succ x\rangle)\\
\mathbf{r}_7 &= m(a,b) \to p(1,a) \ll \epsilon\\
\mathbf{r}_8 &= (\forall x)(m(1,x) \to x \ll \langle id, x \succ 1\rangle)
\end{aligned}$$

There are other similar examples, where using constraints, the completion can be computed in a few seconds, whereas using an implementation of the UKBA, hundreds of critical pairs have to be generated, and several minutes are needed.

Experiments using the theorem prover ${}_3T^AP$ showed that for adding equality to semantic tableaux our method is superior to other approaches (see [4]). A major and still unsolved problem is the fact that ${}_3T^AP$ closes the tableau branches one after the other. In fact, finding a closing substitution for a tableau virtually never fails because a single branch could not be closed, but because the search for a single substitution closing all branches simultaneously takes too long.

12 Conclusion

The method presented is the first completion-based algorithm for mixed E-unification and the first completion-based algorithm for purely rigid E-unification that has been implemented. In addition, there are examples (Sec. 11) where it is superior to the classical UKBA for purely universal problems.

In contrary to other methods for purely rigid E-unification, the algorithm is deterministic, i.e., backtracking has never to be used to complete a system of equalities or to solve a unification problem. Moreover, the completion process does not depend on the terms to be unified. That is important for applications,

[18] Runtimes have been measured on a SUN SPARC 10 workstation.

where often a lot of different terms have to be unified using the same set of equalities.

Completion-based mixed E-unification is a promising way to add the handling of equality to semantic tableaux and other Gentzen-type calculi for first order logic; however, methods have to be developed for composing a substitution that closes all branches of a tableau simultaneously from a great number of substitutions closing single branches.

References

1. L. Bachmair, N. Dershowitz, and D. Plaisted. Completion without failure. In H. Aït-Kaci and M. Nivat, editors, *Resolution of Equations in Algebraic Structures, Volume 2*, chapter 1. Academic Press, 1989.
2. B. Beckert. Ein vervollständigungsbasiertes Verfahren zur Behandlung von Gleichheit im Tableaukalkül mit freien Variablen. Diploma thesis, Univ. Karlsruhe, 1993.
3. B. Beckert, S. Gerberding, R. Hähnle, and W. Kernig. The tableau-based theorem prover ${}_3T^AP$ for multiple-valued logics. In *Proceedings, 11th International Conference on Automated Deduction (CADE), Albany/NY*, LNCS. Springer, 1992.
4. B. Beckert and R. Hähnle. An improved method for adding equality to free variable semantic tableaux. In *Proceedings, 11th International Conference on Automated Deduction (CADE), Albany/NY*, LNCS. Springer, 1992.
5. J. Chabin, S. Anantharaman, and P. Réty. E-unification via constrained rewriting. Unpublished, 1993.
6. H. Comon. Solving inequations in term algebras. In *Proceedings, 5th Symposium on Logic in Computer Science (LICS), Philadelphia/PA*. IEEE Press, 1990.
7. N. Dershowitz. Termination of rewriting. *J. of Symbolic Computation*, 3(1), 1987.
8. J. Gallier, P. Narendran, D. Plaisted, and W. Snyder. Rigid E-unification: NP-completeness and application to equational matings. *Information and Computation*, pages 129–195, 1990.
9. J. Gallier, P. Narendran, S. Raatz, and W. Snyder. Theorem proving using equational matings and rigid E-unification. *Journal of the ACM*, 39(2), 1992.
10. J. Goubault. Simultaneous rigid E-unifiability is NEXPTIME-complete. Technical report, Bull Corporate Research Center, 1993.
11. J. Hsiang and J. Mzali. SbREVE user's guide. Technical report, LRI, Université de Paris-Sud, 1988.
12. C. Kirchner, editor. *Unification.* Academic Press, 1990.
13. D. E. Knuth and P. B. Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Computational Problems in Abstract Algebras.* Pergamon Press, Oxford, 1970.
14. W. Nutt, P. Réty, and G. Smolka. Basic narrowing revisited. *Journal of Symbolic Computation*, 7(3/4):295–318, 1989.
15. U. Petermann. A framework for integrating equality reasoning into the extension procedure. In *Proc., 2nd Workshop on Theorem Proving with Analytic Tableaux and Related Methods, Marseille.* MPI für Informatik, 92-213, Saarbrücken, 1993.
16. G. Peterson. Complete sets of reductions with constraints. In *Proceedings, 10th International Conference on Automated Deduction (CADE), Kaiserslautern*, LNCS. Springer, 1990.
17. J. Siekmann. Universal unification. *Jour. of Symbolic Computation*, 7(3/4), 1989.