UPMAIL Technical Report No. 89
November 15, 1994

A New Procedural Interpretation of Horn Clauses with Equality

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

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

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

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

Abstract

We introduce the *equality elimination method* which is a new procedure for dealing with Horn clause logic programs with equality. The method combines SLD-resolution with a bottom-up equation solving. By solving equations, we try to transform a logic program with equality to a logic program without equality. The transformation uses basic superposition as the main operation. We prove soundness and completeness of the equality elimination method. We also show that approaches based on complete sets of E-unifiers are not satisfying. In particular, we provide a negative solution to the open problem of completeness of SLDE$^+$-resolution.

Contents

Section 1

Introduction

This paper deals with a new approach to logic programming with equality. By a logic program with equality we understand a Horn clause logic program in which equality may occur both in heads and bodies of clauses (in mathematical logic and universal algebra they are called quasi-identities). The model-theoretic semantics of such clauses is well known since long ago [Bir 44, Maĺ 56, Bir 67, Maĺ 73]. However, there is no uniform generally accepted procedural interpretation. The most natural generalization of SLD-resolution introduced in [GR 86] as SLDE-$^+$-resolution is incomplete, as we demonstrate below (Example 2.1). Various definitions of procedural interpretation are actively studied in the theorem proving and logic programming communities. Most existing approaches use special conditions on the set of equations which may occur in programs (see, e.g., the survey [Han 94]), like confluence and/or termination.

Many approaches to handling equality in logic programs use ideas generated in the field of automated theorem proving for theories with equality. Following the well-established tradition [Len], we try to distinguish the three major sources of ideas.

Plotkin [Plo 72] introduced the notion of a unifier w.r.t. a given equational theory, i.e. that of an E-unifier. He has also shown a possibility to build-in equational theories into the calculus of resolution with paramodulation, thus replacing unification by E-unification. However, such embedding of equality is only fully possible when the equality theory is separated from the rest of axioms. The separation means that the rest of axioms does not contain positive occurrences of equality. Equational logic programs are defined by this restriction as a pair $\langle P, E\rangle$, where P is a logic program without positive occurrences of equality, and E is a set of equality facts in [JLM 84, GM 86] or facts and rules [Höl 89, She 92]. Jaffar et. al. defined SLDE-resolution as the procedural interpretation of such a program.

An extension of SLD-resolution called SLDE$^+$-resolution has been proposed in [GR 86, GR 89]. Completeness of SLDE$^+$-resolution was only proved for *well-behaved programs,* which are exactly equational logic programs in the sense of [JLM 84, GM 86, Höl 89]. Completeness for arbitrary logic programs with equality has been left open. (We refute completeness in Example 2.1). Dynamic separation of the equality theory was later used in tableaux methods [BH 92, Bec 94] and connection (or mating) methods [GNRS 92, Pet 93].

The next major idea is the modified method by Brandt [Bra 75]. He has shown how to get rid of the equality substitution axioms by pulling nonvariable terms out from argument lists. This method transforms the original Horn clause program with equality into "flat" Horn clauses without equality[1]. Then one can use the ordinary SLD-resolution as the inference mechanism for

[1]Brandt preserves the denotation $s = t$ for the equality predicate in the transformed program. We say that the transformed program is equality-free since there are no equality axioms left, and thus equality can be replaced by an

the transformed program. In logic programming community, this method has been considered in [BBLM 86, CP 86, BGM 88, Elc 89].

The main influence on the nowadays technique of equational logic programming was made by methods using refinements of inference in the paramodulation-based calculus via ordering strategies and deletion of redundant clauses. This technique appeared [KB 70, Sla 74, Lan 75] and has been further developed in [Hul 80, Pet 83, HR 86, Höl 88, ZK 88, BG 90, Rus 91, Soc 92, NR 92b, BG 94]. In logic programming, this technique was introduced in many papers, including [Fri 84, Hus 85, Red 85, VE 87, Yam 87, Kap 87, GM 88, Pad 88, FHS 89, GHR 92], especially in the framework of integration of logic and functional programming languages.

In this paper we propose a new method which uses all three ideas in a new way. We try to make all equality-related reasoning in the beginning using a special strategy — basic superposition [NR 92a, BGLS 92], the counterpart of basic narrowing [Hul 80, Höl 88, NRS 89]. Using solutions to equations provided by basic superposition, we transform a logic program with equality into a logic program without equality.

The equality elimination method combines bottom-up solutions of equations with top-down query answering in the transformed logic program. Thus, it has a flavor of tabulation, but instead of tabulating solutions to equations we tabulate *conditional solutions.* The use of conditional solutions is very essential: conditional solutions may generate a finite logic program where unconditional solutions would give an infinite one. Also, solving equations in a bottom-up way gives a complete procedure for logic programs with equality, while the top-down approach ($SLDE^+$-resolution [GR 86]) is incomplete.

We also consider some optimizations of our technique. One of the optimizations (variant folding) often allows one to obtain elegant finite programs without equality, using recursive description of solutions to equations.

The equality elimination method is a complete and sound procedure for *arbitrary* logic programs with equality. It does not impose any conditions on the equality theory. In particular, the set of all positive equations need not be confluent or need only contain unconditional equations.

---

arbitrary binary predicate.

Section 2

Preliminaries

We present here a brief overview of notions and preliminary definitions necessary for the paper. For a more thorough coverage, see books and overviews [CL 73, HO 80, Llo 87, Höl 89, Sie 90, DJ 90, Han 94]. We assume the basic knowledge of substitutions and unification.

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

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

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

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

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

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

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

In [GR 86] Gallier and Raatz proposed two extensions of SLD-resolution. Both extensions (SLDE- and SLDE$^+$-resolution) are based on E-unification. These extensions use the idea that in order to solve an equation $s = t$, one tries to find a E-unifier of $s = t$ w.r.t. a subset of equations from the heads of program clauses. SLDE-resolution requires the use of *all* E-unifiers, SLDE$^+$-resolution only uses a *complete* set of E-unifiers. Finding *all* E-unifiers is computationally expensive. For example, the set of all E-unifiers can be infinite when there are finite complete sets of unifiers. Techniques developed in automated reasoning (e.g. narrowing) are oriented to finding complete sets of E-unifiers. In [GR 86] completeness of SLDE-resolution is proved. Completeness of SLDE$^+$-resolution is stated as an open problem. Furbach et. al. in [FHS 89] conjecture a positive solution to completeness. The following example gives the negative answer to this problem.

Example 2.1 *Consider the following program $\mathcal{P}$:*

$$v \cdot v = v \leftarrow P(v)$$
$$P(f(a \cdot a))$$

and the query

$$\leftarrow f(x \cdot x) \cdot f(y) = f(x \cdot x)$$

It is obvious that the terms $f(x \cdot x) \cdot f(y)$ and $f(x \cdot x)$ are not unifiable. They are, however, E-unifiable w.r.t. $E = \{v \cdot v = v\}$ [Her 86]. Apply the E-unification algorithm from [HO 80], we obtain a complete set of E-unifiers consisting of one element $\sigma = [y \leftarrow x]$. In order to resolve the goal using σ, one has to find possible proofs in E of the equation $f(x \cdot x) \cdot f(y) = f(x \cdot x)$. Any such proof would require unification of $v \cdot v$ with a term $t \cdot t$ and unification of $v \cdot v$ with the term $f(t \cdot t) \cdot f(t \cdot t)$. It means that the equation will be reduced to a new subgoal G containing both $P(t)$ and $P(s[f[t]])$, for suitable terms s and t. The set of non-equation atoms provable from $\mathcal{P}$ in first-order logic has the form

$$\{P((f(a \cdot a)^n) \mid n \geq 1\}$$

where t^n denotes

$$\underbrace{t \cdot \ldots \cdot t}_{n \text{ occurrences of } t}$$

Thus, G is not provable from $\mathcal{P}$ in first-order logic. Using soundness of SLDE-$^+$-resolution, we obtain that G is not derivable by SLDE$^+$-resolution and hence $f(x \cdot x) \cdot f(y) = f(x \cdot x)$ is not derivable either.

It is easy to see that there is a correct answer substitution $[x \leftarrow a, y \leftarrow a \cdot a]$. To prove this goal by SLDE$^+$-resolution, one has to use the E-unifier $\theta = [y \leftarrow x \cdot x]$, equivalent to σ w.r.t. E. With this E-unifier, the goal will be reduced to $P(f(z \cdot z))$, which is solved using the program fact.

This example also demonstrates the "wild nature" of the notion of a complete set of unifiers. Consider the following situation. Assume that CE is a set of conditional equations in the signature Σ. This set defines a congruence relation $=_{CE}$ on the set of terms $T(\Sigma, X)$, defined by $t =_{CE} s \Leftrightarrow CE \models t = s$. Let $E \Leftrightarrow \{s = t \mid CE \models s = t\}$. (It means that the set of quasivarieties satisfying CE and the set of varieties satisfying E have the same freely generated systems [Maĺ 73].) Consider now a set E' of equations defined as follows:

$$E' = \{s\sigma = t\sigma \mid \quad \exists (s = t \leftarrow s_1 = t_1, \ldots, s_n = t_n) \in CE \text{ such that } \sigma \in U_{CE}(\{\langle s_1, t_1 \rangle, \ldots, \langle s_n, t_n \rangle\})\}$$

where $U_{CE}(\{\langle s_1, t_1 \rangle, \ldots, \langle s_n, t_n \rangle\})$ is a procedure which generates a complete set of E-unifiers for $\{\langle s_1, t_1 \rangle, \ldots, \langle s_n, t_n \rangle\}$. If one wants to define a fixpoint theory for equational logic programs that uses complete sets of unifiers instead of most general unifiers in the theory of logic programming, it should follow that E is equivalent to E'.

We show that E' can define a congruence relation different from $=_{CE}$. Indeed, let

$$CE = \{v = v \cdot v \leftarrow P(v) = true, \quad P(f(a \cdot a)) = true \leftarrow\}$$

Then

$$U_{CE}(\{\langle P(v), \mathit{true}\rangle\}) = [v \leftarrow (f(a \cdot a))^2]$$

is a complete set of E-unifiers for $P(v) = \mathit{true}$. Thus,

$$E' = \{(f(a \cdot a))^2 = (f(a \cdot a))^4, P(f(a \cdot a)) = \mathit{true}\}$$

Hence, $f(a \cdot a)$ is equal to $(f(a \cdot a))^2$ in CE, but not in E'.

Section 3

The equality elimination method

The equality elimination method is based on a transformation of clauses that removes the equality predicate from the program. The resulting program may, in general, be infinite. There are some very general cases when it will always be finite. In Section 3.1 we consider some optimizations that allow one to obtain a finite program more often.

Let $X = x_1, x_2, \ldots$ be a fixed countable sequence of variables. A term t is called *X-normal* iff its variables (in the order of their first occurrences in t) are $x_1, \ldots, x_n$. An *X-normal form* of a term t is the variant of t which is X-normal.

Let $\mathcal{P}$ be an logic program with equality. We always assume that the query is a part of the program. The equality-free program $\mathcal{EI}(\mathcal{P})$ is constructed in the following way. Let Pr be a countable set of predicate symbols disjoint from predicate symbols of Σ. Let there be a mapping that associates with each X-normal atom B a unique predicate symbol $A_B \in Pr$. The arity of A_B is the number of different variables in B.

Let B be an arbitrary atom and B' be the X-normal form of B. Then $B = B'[x_1 \leftarrow y_1, \ldots, x_n \leftarrow y_n]$, for some n, x_i, y_i. We define

$$\overline{B} = A_B(y_1, \ldots, y_n)$$

For example,

$$\overline{P(y, f(z), z)} = A_{P(x_1, f(x_2), x_2)}(y, z)$$

and

$$\overline{P(a)} = A_{P(a)}$$

The transformed program $\mathcal{EI}(\mathcal{P})$ will use a finite subset of Pr as its predicate symbols, and function symbols of Σ as its function symbols. The program $\mathcal{EI}(\mathcal{P})$ is defined as the limit of programs $\mathcal{EI}(\mathcal{P})_0 \subseteq \mathcal{EI}(\mathcal{P})_1 \subseteq \ldots$ defined below.

Definition 3.1 (superposition) *Let $\succ$ be a reduction ordering. Then* left superposition *and* right superposition *are defined as inference rules*

$$\frac{s' = t' \leftarrow T_1 \qquad s[u] = t \leftarrow T_2}{(s[t'] = t \leftarrow T_1, T_2)\sigma}$$

and

$$\frac{s' = t' \leftarrow T_1 \qquad H \leftarrow s[u] = t, T_2}{(H \leftarrow s[t'] = t, T_1, T_2)\sigma}$$

where

1. *σ is a most general unifier of u and s'.*
2. *$t'\sigma \not\succeq s'\sigma$;*
3. *$t\sigma \not\succeq s[u]\sigma$;*
4. *u is not a variable.*

We shall also say that the clause at the conclusion of either of these rules is a superposition of the clauses at the premises of the rule.

We always assume that equations are commutative. For example, replacing $s' = t'$ in the above rule by $t' = s'$ gives a valid superposition.

The construction of $\mathcal{EI}(\mathcal{P})$ uses a reduction ordering $\succ$ and consists of the following steps.

Step 1. Let $\mathcal{EI}(\mathcal{P})_0 = \emptyset$.

Step 2. Let C_1, C_2 be variants of clauses of $\mathcal{P}$ having no common variables. Let C_1 have the form

$$\ldots \leftarrow \ \ldots \ P(t_1, \ldots, t_n) \ \ldots$$

and C_2 have the form

$$P(s_1, \ldots, s_n) \ \leftarrow \ B_1, \ldots, B_m$$

and P is not the equality predicate.

Then we add the following clause to $\mathcal{EI}(\mathcal{P})_0$:

$$\overline{P(t_1, \ldots, t_n)} \leftarrow \overline{t_1 = s_1}, \ldots, \overline{t_n = s_n}, \overline{B_1}, \ldots, \overline{B_m}$$

For example, if the first clause is

$$P(f(x)) \ \leftarrow \ Q(f(y), x)$$

and the second clause is

$$Q(g(a, z), z) \ \leftarrow \ R(u, z)$$

then we have to add to $\mathcal{EI}(\mathcal{P})$ the clause

$$\begin{array}{rcl} A_{Q(f(x_1),x_2)}(y, x) & \leftarrow & A_{f(x_1)=g(a,x_2)}(y, z), \\ & & A_{x_1=x_2}(x, z), \\ & & A_{R(x_1,x_2)}(u, z) \end{array}$$

Note that C_1, C_2 may be variants of the same clause.

Step 3. For the query

$$\leftarrow B_1, \ldots, B_n$$

add the query

$$\leftarrow \overline{B_1}, \ldots, \overline{B_n}$$

to $\mathcal{EI}(\mathcal{P})_0$.

Step 4. Construct two sets of clauses $used_0$ and new_0. Initially, let

$$\begin{array}{rcl} used_0 & = & \emptyset \\ new_0 & = & \{s = t \leftarrow \overline{B_1}, \ldots, \overline{B_n} \mid s = t \leftarrow B_1, \ldots, B_n \in \mathcal{P}\} \bigcup \\ & & \{\overline{s = t} \leftarrow s = t \mid \overline{s = t} \text{ occurs in } \mathcal{EI}(\mathcal{P})\} \end{array}$$

For example, if

$$(f(x) = f(y) \leftarrow P(x, y, y)) \in \mathcal{P}$$

then

$$(f(x) = f(y) \leftarrow A_{P(x_1, x_2, x_2)}(x, y)) \in new_0$$

Step 5. We assume that $\mathcal{EI}(\mathcal{P})_i$, new_i and $used_i$ are already constructed at this point. If $new_i = \emptyset$ then terminate and let $\mathcal{EI}(\mathcal{P})_k = \mathcal{EI}(\mathcal{P})_i$, for all $k > i$. Select a clause $C \in new_i$. Let $new_{i+1} = new_i \setminus \{C\}$ and $used_{i+1} = used_i \bigcup \{C\}$. Let $\mathcal{EI}(\mathcal{P})_{i+1} = \mathcal{EI}(\mathcal{P})_i$.

1. If C has the form $H \leftarrow T, \bar{s} = \bar{t}$, where T contains no equations and $\bar{s}, \bar{t}$ are unifiable with a most general unifier θ, then add the clause $H\theta \leftarrow T\theta$ to $\mathcal{EI}(\mathcal{P})_{i+1}$.
2. Add superpositions of C and all clauses from $used_{i+1}$ to new_{i+1}

We assume that on this step the choice of clauses from new_i is *fair*, i.e. if a clause belongs to new_i, it will be selected on some iteration.

Definition 3.2 ($\mathcal{EI}(\mathcal{P})$) *We denote by* $\mathcal{EI}(\mathcal{P})$ *the program* $\bigcup_{i=0}^{\infty} \mathcal{EI}(\mathcal{P})_i$.

Let us note that the transformation described here does not use basic superposition to full extent. By introducing new predicate names, we get rid of non-basic positions in the beginning. However, superposition of equations may introduce new non-basic positions, that we do not check. Our procedure can gain of using full basic superposition[1]. We do not give precise definitions here since it would introduce too many technical details.

To illustrate the algorithm, we turn back to Example 2.1.

[1]In basic superposition, superpositions can only be made into positions that existed in the original problem, but not in those appeared as a result of (other) superpositions.

i	$\mathcal{EI}(\mathcal{P})_i \setminus \mathcal{EI}(\mathcal{P})_{i-1}$	$used_i \setminus used_{i-1}$	new_i
0	$A(v) \leftarrow E_0(v)$ $\leftarrow E_1(x, y)$		$E_0(x) \leftarrow x = f(a \cdot a)$ $E_1(x, y) \leftarrow f(x \cdot x) \cdot f(y) = f(x \cdot x)$ $v \cdot v = v \leftarrow A(v)$
1	$E_0(f(a \cdot a))$	$E_0(x) \leftarrow x = f(a \cdot a)$	$E_1(x, y) \leftarrow f(x \cdot x) \cdot f(y) = f(x \cdot x)$ $v \cdot v = v \leftarrow A(v)$
2		$E_1(x, y) \leftarrow$ $f(x \cdot x) \cdot f(y) = f(x \cdot x)$	$v \cdot v = v \leftarrow A(v)$
3		$v \cdot v = v \leftarrow A(v)$	$E_0(x) \leftarrow x = f(a), A(a)$ $E_1(x, x \cdot x) \leftarrow$ $f(x \cdot x) = f(x \cdot x), A(f(x \cdot x))$ $E_1(x, y) \leftarrow f(x) \cdot f(y) = f(x \cdot x), A(x)$
4	$E_0(f(a)) \leftarrow A(a)$	$E_0(x) \leftarrow x = f(a), A(a)$	$E_1(x, x \cdot x) \leftarrow$ $f(x \cdot x) = f(x \cdot x), A(f(x \cdot x))$ $E_1(x, y) \leftarrow f(x) \cdot f(y) = f(x \cdot x), A(x)$
5	$E_1(x, x \cdot x) \leftarrow$ $A(f(x \cdot x))$	$E_1(x, x \cdot x) \leftarrow$ $f(x \cdot x) = f(x \cdot x)$, $A(f(x \cdot x))$	$E_1(x, y) \leftarrow f(x) \cdot f(y) = f(x \cdot x), A(x)$
6		$E_1(x, y) \leftarrow$ $f(x) \cdot f(y) = f(x \cdot x), A(x)$	$E_1(x, x) \leftarrow$ $f(x) = f(x \cdot x), A(f(x)), A(x)$
7		$E_1(x, x) \leftarrow$ $f(x) = f(x \cdot x)$, $A(f(x)), A(x)$	$E_1(x, x) \leftarrow$ $f(x) = f(x)$, $A(f(x)), A(x), A(x)$
8	$E_1(x, x) \leftarrow$ $A(f(x)), A(x), A(x)$	$E_1(x, x) \leftarrow$ $f(x) = f(x)$, $A(f(x)), A(x), A(x)$	

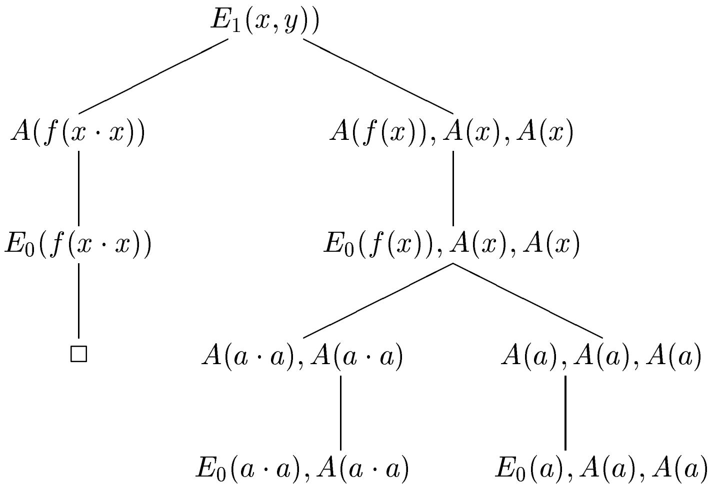
Figure 3.1: The equality elimination procedure for the program of Example 2.1

Example 3.1 *Consider the program from Example 2.1. Assume that we use recursive path ordering [DJ 90] with $\cdot > f$. Introduce new predicate names:*

$$
\begin{array}{rcl}
A(v) & = & A_{P(x_1)}(v) \\
E_0(x) & = & A_{x_1=f(a,a)}(x) \\
E_1(x,y) & = & A_{f(x_1\cdot x_1)\cdot f(x_2)=f(x_1\cdot x_1)}(x,y)
\end{array}
$$

The construction of $\mathcal{EI}(\mathcal{P})_i$ is shown on Figure 3.1

The (complete) SLD-tree for this query is



i.e. not only the answer is found, but also the tree is finite. It is interesting that if we use the opposite recursive path ordering with $f > \cdot$ we would obtain the same program, but in a different order.

Theorem 3.1 (soundness) *The equality elimination method is sound, i.e. if $\mathcal{EI}(\mathcal{P})$ is refutable with a computed answer substitution θ, then θ is a correct answer substitution for $\mathcal{P}$.*

Proof. For every atom B in the new signature (with predicates $\overline{B}$), we introduce an atom $\underline{B}$ defined as follows:

$$
\underline{B} = \begin{cases} s=t & B \text{ has the form } s=t \\ D[x_1\leftarrow t_1,\ldots,x_n\leftarrow t_n] & B \text{ has the form } A_D(t_1,\ldots,t_n) \end{cases}
$$

We extend this to conjunctions, clauses and programs:

$$
\begin{array}{rcl}
\underline{B_1,\ldots,B_n} & = & \underline{B_1},\ldots,\underline{B_n} \\
\underline{H\leftarrow T} & = & \underline{H}\leftarrow\underline{T} \\
\underline{\{C_1,\ldots,C_n\}} & = & \{\underline{C_1},\ldots,\underline{C_n}\}
\end{array}
$$

For example, if the clause C has the form

$$A_{P(x_1,f(x_2))}(f(x),y) \leftarrow f(x)=f(y), A_{x_1=g(x_2)}(a,f(y))$$

then $\underline{C}$ is

$$P(f(x),f(y)) \leftarrow f(x)=f(y), a=g(f(y))$$

Lemma 3.1 $\underline{(\overline{B})} = B$.

Proof. Obvious. □

Lemma 3.2 *For every clause $C \in \mathcal{EI}(\mathcal{P})_0 \bigcup new_0$ we have $\mathcal{P} \vdash \underline{C}$.*

Proof. The program $\mathcal{EI}(\mathcal{P})_0$ consist of clauses C of the form

$$\overline{P(t_1,\ldots,t_n)} \leftarrow \overline{t_1=s_1},\ldots,\overline{t_n=s_n},\overline{B_1},\ldots,\overline{B_m}$$

such that

$$(P(s_1,\ldots,s_n) \ \leftarrow \ B_1,\ldots,B_m) \in \mathcal{P}$$

By Lemma 3.1, $\underline{C}$ is

$$P(t_1,\ldots,t_n) \leftarrow t_1=s_1,\ldots,t_n=s_n,B_1,\ldots,B_m$$

which is a consequence of $\mathcal{P}$. The case $C \in new_0$ and the case when C is the query are similar. □

Lemma 3.3 *Consider an application of the resolution rule*

$$\frac{A \leftarrow B, T_1 \qquad B' \leftarrow T_2}{(A \leftarrow T_1, T_2)\theta}$$

where θ is a most general unifier of B and B'. Then

$$\frac{\underline{A \leftarrow B, T_1} \qquad \underline{B' \leftarrow T_2}}{\underline{(A \leftarrow T_1, T_2)\theta}}$$

is a valid resolution rule.

Proof. Immediately follows from two facts:

1. If B, B' are unifiable with a most general unifier θ, then $\underline{B}, \underline{B'}$ are unifiable with the same most general unifier.
2. $\underline{C}\theta = \underline{C\theta}$.

□

Lemma 3.4 *Consider an application of the superposition rule*

$$\frac{s' = t' \leftarrow T_1 \qquad s[u] = t \leftarrow T_2}{(s[t'] = t \leftarrow T_1, T_2)\sigma}$$

where σ is a most general unifier of u and s'. Then

$$\frac{s' = t' \leftarrow \underline{T_1} \qquad s[u] = t \leftarrow \underline{T_2}}{(s[t'] = t \leftarrow \underline{T_1, T_2})\sigma}$$

is a valid superposition rule.

Similar for the right superposition rule.

Proof. Similar to that of Lemma 3.3. We only have to note that for positions of term u in the first rule there are corresponding positions in the second rule. □

Lemma 3.5 *For every clause $C \in \mathcal{EI}(\mathcal{P})_n \bigcup used_n \bigcup new_n$ we have $\mathcal{P} \vdash \underline{C}$.*

Proof. Use induction on n. For $n = 0$ it has already been proved. Note that all clauses from $used_n$ were in new_{n-1}, so we may only consider new_n and $\mathcal{EI}(\mathcal{P})_n$.

1. A clause $C \in new_n$ is obtained by superposition. Then apply Lemma 3.4.
2. A clause $C \in \mathcal{EI}(\mathcal{P})_n$ is obtained from a clause $C' \in new_i$. Similar to the proof of Lemma 3.3.

□

Denote by $Q = \leftarrow Q'$ the query of $\mathcal{EI}(\mathcal{P})$. Let $\mathcal{EI}(\mathcal{P})^- = \mathcal{EI}(\mathcal{P}) \setminus \{Q\}$.

Lemma 3.6 *Assume that there is a refutation of $\mathcal{EI}(\mathcal{P})^-$ and a query R with computed answer substitution θ. Then there is a refutation of $\underline{\mathcal{EI}(\mathcal{P})^-}$ and $\underline{R}$ with the same answer substitution.*

Proof. It is known that the refutation may be obtained by applications of the resolution rule (see Lemma 3.3). Then apply Lemma 3.3. □

We come back to the proof of Theorem 3.1. Assume that $\mathcal{EI}(\mathcal{P})$ is refutable with a computed answer substitution θ. Then $\mathcal{EI}(\mathcal{P})^-$ and Q are refutable with θ. By Lemma 3.6, $\underline{\mathcal{EI}(\mathcal{P})^-}$ and $\underline{Q}$ are refutable with θ, i.e. $\underline{\mathcal{EI}(\mathcal{P})} \vdash \underline{Q'}\theta$. By Lemma 3.1, $\underline{Q'}$ is the query of $\mathcal{P}$. Then apply Lemma 3.5. □

Note that by compactness [Maĺ 73] Theorem 3.2 means that whenever $\mathcal{EI}(\mathcal{P})$ is refutable, some $\mathcal{EI}(\mathcal{P})_i$ is also refutable.

Theorem 3.2 (completeness) *Assume that $\mathcal{P}$ is refutable. Then $\mathcal{EI}(\mathcal{P})$ is refutable.*

Proof. We prove some technical lemmas.

Lemma 3.7 *Let the program $\mathcal{P}'$ be obtained from $\mathcal{P}$ by adding clauses of the form $\overline{B} \leftarrow B$, where B has a negative occurrence in $\mathcal{P}$. Then $\mathcal{P}$ is satisfiable iff $\mathcal{P}'$ is satisfiable.*

Proof. Obvious, since new predicates A_B do not occur in $\mathcal{P}$. $\square$

Lemma 3.8 *Let $\mathcal{P}$ be a program in which all positive occurrences of the predicate symbol A_B are in the clause $C = (\overline{B} \leftarrow B)$. Then $\mathcal{P}$ is satisfiable iff there is a model $\mathcal{M}$ of $\mathcal{P}$ such that $\mathcal{M} \models \forall(\overline{B} \equiv B)$.*

Proof. Consider the model $\mathcal{M}'$ which interprets all predicates but A_B as $\mathcal{M}$ does, and $\mathcal{M}' \models \forall(\overline{B} \equiv B)$. Then $\mathcal{M}' \models C$. Any other clause C' is true in $\mathcal{M}'$ because A_B may only occur negatively in C'. $\square$

Lemma 3.9 *Let $\mathcal{P}$ be a program in which all positive occurrences of the predicate symbol A_B are in the clause $C = (\overline{B} \leftarrow B)$. Let $(H \leftarrow B, T) \in \mathcal{P}$ and $\mathcal{P}'$ be obtained from $\mathcal{P}$ by replacing this clause with $(H \leftarrow \overline{B}, T)$. Then $\mathcal{P}$ is satisfiable iff $\mathcal{P}'$ is satisfiable.*

Proof. Note that both $\mathcal{P}$ and $\mathcal{P}'$ satisfy conditions of Lemma 3.8. Let $\mathcal{M}$ be a model of any of this programs such that $\mathcal{M} \models \forall(\overline{B} \equiv B)$. Then $\mathcal{M}$ is obviously a model of the other program. $\square$

Lemma 3.10 *Let the set of all definitions of a predicate P in the program $\mathcal{P}$ be*

$$
\begin{array}{l}
P(\bar{t_1}) \leftarrow T_1 \\
\ldots \\
P(\bar{t_n}) \leftarrow T_n
\end{array}
$$

Let

$$\varphi(\bar{x}) = \exists \bar{y_1}(\bar{x} = \bar{t_1} \wedge T_1) \vee \ldots \vee \bar{y_n}(\bar{x} = \bar{t_n} \wedge T_n)$$

be the Clark's completion of P [Llo 87]. Then if $\mathcal{P}$ is satisfiable then there is a model $\mathcal{M}$ of $\mathcal{P}$ such that $\mathcal{M} \models \forall(\varphi(\bar{x}) \equiv P(\bar{x}))$.

Proof. Analogous of Lemma 3.8, but taking into consideration that φ may contain positive occurrences of P, and we have to make P be the least solution of the recursive equation $P = \varphi$. $\square$

Lemma 3.11 *Let the set of all definitions of a predicate P in the program $\mathcal{P}$ be*

$$
\begin{array}{l}
P(\bar{t_1}) \leftarrow T_1 \\
\ldots \\
P(\bar{t_n}) \leftarrow T_n
\end{array}
$$

Let $H \leftarrow P(\bar{s}) \in \mathcal{P}$. Let $\mathcal{P}'$ be obtained from $\mathcal{P}$ by replacement of this clause to the set of clauses

$$
\begin{array}{l}
H \leftarrow \bar{s} = \bar{t_1}, T_1 \\
\ldots \\
H \leftarrow \bar{s} = \bar{t_n}, T_n
\end{array}
$$

Then $\mathcal{P}$ is satisfiable iff $\mathcal{P}'$ is satisfiable.

Proof. Follows from Lemma 3.10. □

Lemma 3.12 *Let the predicate P does not occur in $\mathcal{P}$ negatively. Let $\mathcal{P}'$ be obtained from $\mathcal{P}$ be deletion of all clauses defining $\mathcal{P}$. Then $\mathcal{P}$ is satisfiable iff $\mathcal{P}'$ is satisfiable.*

Proof. Obvious. □

Combining Lemmas 3.7, 3.9, 3.11 and 3.12, we obtain

Lemma 3.13 *$\mathcal{P}$ is refutable iff $\mathcal{EI}(\mathcal{P})_0 \bigcup new_0$ is refutable.*

We come back to the proof of Theorem 3.2. Let $\mathcal{P}$ is refutable. Then, by Lemma 3.13, $\mathcal{EI}(\mathcal{P})_0 \bigcup new_0$ is refutable. The refutation of the latter program can be obtained in the calculus consisting of basic paramodulation, (binary) resolution and reflection (resolving with the atom $x = x$). Due to the basic restriction, there is refutation in which all paramodulations precede all other rules, and all reflections precede all resolutions (see [Deg 82, DV 86]). Let us call such refutation *regular.* In the initial set $\mathcal{EI}(\mathcal{P})_0 \bigcup new_0$ all arguments of non-equality predicates are variables. Then the basic strategy requires that all paramodulations be made into equations, thus becoming left or right superpositions. For regular refutations, reflections can only be made into clauses containing no positive equations, and resolution can only be applied to clauses without equations.

Consider any regular refutation Π. All superpositions from the initial set can be made by our algorithm. All reflections from Π are made when we move clauses from new_i to $\mathcal{EI}(\mathcal{P})_{i+1}$. Thus, all topmost clauses in Π which do not contain equality are members of $\mathcal{EI}(\mathcal{P})$. □

Consider now some pragmatic aspects of the implementation of the equality elimination method. The major problem with the method is that the procedure may be non-terminating. We discuss some optimizations designed in order to make it terminate in Section 3.1. When the process is non-terminating, we can only suggest the following solution: SLD-resolution on the transformed program $\mathcal{EI}(\mathcal{P})$ can be run concurrently with the construction of $\mathcal{EI}(\mathcal{P})$.

In general, it seems inevitable to avoid infiniteness. Consider other top-down procedures. SLDE-resolution may search infinitely at *any part* of the proof tree. Thus, SLDE-resolution is even more unrealistic from the computational viewpoint. SLDE$^+$-resolution is incomplete. The modification method [Bra 75] always gives a finite program, but *every* solution of equation will be non-terminating[2], even when our transformation terminates.

The equality elimination method is also different from the mentioned methods in that it tabulates solutions to equations. The use of the basic strategy makes the following effect on tabulation: solutions to equations are searched and tabulated in such a way so every solvable dynamic instance of an equation will be guaranteed to have enough tabulated solutions to be resolved.

Thus, the equality elimination method seems to be the only method that preserves the top-down search by *SLD*-resolution, is complete and uses no special equality axioms like transitivity [Bra 75].

Our method is also applicable to subsets of Prolog with equality. If equality axioms are defined by definite clauses without any extralogical features our method still works, even if the rest of the

[2]If an equality $a = b$ is used in the Brandt's method, two clauses $a = x \leftarrow b = x$ and $b = x \leftarrow a = x$ will be introduced, which is needed to simulate transitivity.

program uses some non-logical control primitives. For example, our method is compatible with cuts of the form

$$A \leftarrow T, [!B_1, \ldots, B_n!], T'.$$

used in languages Arity Prolog and Gödel.

Our method is, however, incompatible with unrestricted cuts, like

$$A \leftarrow A_1, \ldots, !, \ldots, A_n$$

3.1 Optimizations

There are a two obvious optimizations of the equality elimination procedure which may save a great deal of work and prevent creation of unnecessary clause. Here we explain some of them.

Tautology deletion If a clause of the form $t = t \leftarrow \ldots$ is generated, it can be removed from new_i. In fact, we used tautology deletion in Figure 3.1: when $i = 3$, the clause $v = v \leftarrow A(v), A(v)$ should have been generated by superposition of $v \cdot v \leftarrow A(v)$ with itself.

Tautology replacement If a clause of the form $H \leftarrow t = t, T$ is generated, it can be replaced by $H \leftarrow T$. Have we used this optimization, the equality elimination process on Figure 3.1 would have stopped after $i = 6$, generating two clauses less.

Subsumption We can remove subsumed clauses from $\mathcal{EI}(\mathcal{P})$. (If we apply subsumption in the same way to $used_i$ or new_i, we can lose completeness[3]. However we can always safely remove from new_i *variants* of clauses from $used_k$.)

Double literal elimination If a clause $H \leftarrow A, A, T$ is generated, it can be replaced by $H \leftarrow A, T$. For example, on Figure 3.1 the program clause $E_1(x, x) \leftarrow A(f(x)), A(x), A(x)$ can be replaced by $E_1(x, x) \leftarrow A(f(x)), A(x)$.

Equation rewriting Every clause of the form

$$H \leftarrow x = f(t_1, \ldots, t_n), T$$

can be replaced by

$$H \leftarrow x = f(x_1, \ldots, x_n), x_1 = t_1, \ldots, x_n = t_n, T$$

Constructor solution If a function symbol f is a free constructor, i.e. $\mathcal{P} \models f(\bar{x}) = f(\bar{y}) \supset \bar{x} = \bar{y}$, then every clause of the form

$$H \leftarrow f(s_1, \ldots, s_n) = f(t_1, \ldots, t_n), T$$

can be replaced by

$$H \leftarrow s_1 = t_1, \ldots, s_n = t_n, T$$

Some techniques for constructor solution has been considered in papers on mechanization of inductive proofs in the field of algebraic specifications [HH 82, JK 86, Fri 89].

Unsolvable equation elimination If an equation $t_1 = t_2$ has no solutions, i.e. $\mathcal{P} \not\models \exists(t_1 = t_2)$, then any clause

$$H \leftarrow t_1 = t_2, T$$

may be deleted. This situation can often arise when working with free constructors. For example, `[]` is never equal `[`t_1`|`t_2`]` for lists.

[3]Techniques for eliminating redundancies in context of the basic strategy have been considered in [Ret 87, MH 92, BGLS 92, NR 92b].

The problem of finding necessary conditions for application of the last two optimizations is an instance of the problem of *consistency of equational enrichment* and has been studied e.g. in [GTW 78, EM 85, AD 92].

The next optimization we are going to propose is more sophisticated and more powerful than the other ones. We call it *recursive folding.*

Assume that after superposition we obtained a clause

$$A_{s=t}(r_1, \ldots, r_n) \leftarrow s' = t', T$$

and $\overline{s' = t'} = A_{s''=t''}(u_1, \ldots, u_k)$. Then

1. If $A_{s''=t''}$ was not introduced before, we replace the above clause by the two clauses

   $$A_{s''=t''} \leftarrow s'' = t''$$

   and

   $$A_{s=t}(r_1, \ldots, r_n) \leftarrow \overline{s' = t'}, T$$

2. If $A_{s''=t''}$ was introduced before, replace the clause by

   $$A_{s=t}(r_1, \ldots, r_n) \leftarrow \overline{s' = t'}, T$$

This rule is called recursive folding because it can introduce recursive clauses representing solutions to equations.

Example 3.2 *Consider the following program defining the function ap (append) and a predicate rev (reverse) for lists as follows:*
$ap(\texttt{[]}, x) = x$
$ap(\texttt{[}x \,|\, y\texttt{]}, z) = \texttt{[}x \,|\, ap(y, z)\texttt{]}$
$rev(\texttt{[]}, \texttt{[]})$
$rev(\texttt{[}x \,|\, y\texttt{]}, ap(z, \texttt{[}x\texttt{]})) \leftarrow rev(y, z)$
$\leftarrow rev(\texttt{[}a, b\texttt{]}, u)$

According to the equality elimination procedure we introduce new predicate symbols:
$A_0(y, z) \equiv rev(y, z)$
$A_1(u) \equiv rev(\texttt{[}a, b\texttt{]}, u)$
$E_0(y) \equiv y = \texttt{[]}$
$E_1(y) \equiv y = \texttt{[}a, b\texttt{]}$
$E_2(z, u) \equiv z = u$
$E_3(x, y, y') \equiv \texttt{[}x \,|\, y\texttt{]} = y'$
$E_4(z, x, z') \equiv ap(z, \texttt{[}x\texttt{]}) = z'$
$E_5(x, y) \equiv \texttt{[}a, b\texttt{]} = \texttt{[}x \,|\, y\texttt{]}$
$E_6 \equiv \texttt{[}a, b\texttt{]} = \texttt{[]}$

Comparison of terms will be made according to the recursive path ordering with precedence $ap > \texttt{[|]} > []$. *Then in both equations for ap the left hand side is greater than the right hand side. After 10 steps of the procedure we obtain sets of clauses shown in Figure 3.2.*

Applying equality rewriting, we replace the clause

$$E_4(\texttt{[}x \,|\, y\texttt{]}, z, v) \leftarrow \texttt{[}x \,|\, ap(y, \texttt{[}z\texttt{]})\texttt{]} = v$$

i	$\mathcal{EI}(\mathcal{P})_i \setminus \mathcal{EI}(\mathcal{P})_{i-1}$	$used_i \setminus used_{i-1}$	new_i
0	$A_0(y,z) \leftarrow$ $E_0(y), E_0(z)$ $A_0(y',z') \leftarrow$ $E_3(x,y,y')$, $E_4(z,x,z')$, $A_0(y,z)$ $A_1(u) \leftarrow$ $E_6, E_0(u)$ $A_1(u) \leftarrow$ $E_5(x,y)$, $E_4(z,x,u)$, $A_0(y,z)$ $\leftarrow A_1(u)$		$ap([\,],x) = x$ $ap([x \mid y],z) = [x \mid ap(y,z)]$ $E_0(y) \leftarrow y = [\,]$ $E_1(y) \leftarrow y = [a,b]$ $E_2(z,u) \leftarrow z = u$ $E_3(x,y,y') \leftarrow [x \mid y] = y'$ $E_4(z,x,z') \leftarrow ap(z,[x]) = z'$ $E_5(x,y) \leftarrow [a,b] = [x \mid y]$ $E_6 \leftarrow [a,b] = [\,]$
7	$E_0([\,])$ $E_1([a,b])$ $E_2(z,z)$ $E_3(x,y,[x \mid y])$ $E_4(z,x,ap(z,[x]))$ $E_5(a,[b])$	$E_0(y) \leftarrow y = [\,]$ $E_1(y) \leftarrow y = [a,b]$ $E_2(z,u) \leftarrow z = u$ $E_3(x,y,y') \leftarrow [x \mid y] = y'$ $E_4(z,x,z') \leftarrow ap(z,[x]) = z'$ $E_5(x,y) \leftarrow [a,b] = [x \mid y]$ $E_6 \leftarrow [a,b] = [\,]$	$ap([\,],x) = x$ $ap([x \mid y],z) = [x \mid ap(y,z)]$
8		$ap([\,],x) = x$	$E_4([\,],x,z') \leftarrow [x] = z'$ $ap([x \mid y],z) = [x \mid ap(y,z)]$
9	$E_4([\,],x,[x])$	$E_4([\,],x,z') \leftarrow [x] = z'$	$ap([x \mid y],z) = [x \mid ap(y,z)]$
10		$ap([x \mid y],z) = [x \mid ap(y,z)]$	$E_4([x \mid y],z,v) \leftarrow [x \mid ap(y,[z])] = v$

Figure 3.2: The equality elimination procedure for the program describing *ap* and *rev*

by

$$E_4(\mathtt{[}x\,|\,y\mathtt{]}, z, \mathtt{[}v\,|\,u\mathtt{]}) \leftarrow \mathtt{[}x_1\,|\,u\mathtt{]} = v, x = x_1, ap(y, \mathtt{[}z\mathtt{]}) = u$$

Then, by recursive folding, we replace it by

$$E_4(\mathtt{[}x\,|\,y\mathtt{]}, z, \mathtt{[}v\,|\,u\mathtt{]}) \leftarrow \mathtt{[}x_1\,|\,u\mathtt{]} = v, x = x_1, E_4(y, z, u)$$

add to $\mathcal{EI}(\mathcal{P})$ *the clause*

$$E_4(\mathtt{[}x\,|\,y\mathtt{]}, z, \mathtt{[}x\,|\,u\mathtt{]}) \leftarrow E_4(y, z, u)$$

and terminate. Thus, recursive folding created a predicate definition of append!

We are going to consider recursive folding in more details in a forthcoming publication.

Section 4

Related work

In this paper we are trying to "compile out" equational reasoning with the aim of obtaining an equivalent logic program without equality, which can be executed by ordinary SLD-resolution. This reminds the process of completion of equational theories which can also be characterized as a transformation of an equational theory to a canonical form [HO 80]. As a rule, this kind of processes does not terminate, i.e. one cannot obtain a finite canonical program (see e.g. *unfailing completion [BDP 89].* However, any query that is a consequence of the program can be answered at a finite stage of this process by narrowing. For Horn clause theories it has been developed in [Kou 88, Gan 91]. Our approach is very different from the cited papers because we succeed in obtaining a finite program more often, using generation of conditional and recursive rules.

The idea of "suspended resolution" has been considered in E-resolution [Mor 69, And 70] and RUE-resolution [DH 86] and repeated in [Höl 89] as "lazy resolution". All these methods are either incomplete, or use special equality axioms. Our system is complete and the transformed program does not use any equality axioms. The use of the basic strategy that we achieve by introducing new predicate names can significantly reduce the number of clauses in the transformed program. Moreover, the basic strategy reduces the work needed to obtain the transformed program.

The basic strategy has been introduced by one of the authors in [Deg 79, Deg 82] (published in Russian[1]) and independently in [BGLS 92, NR 92a] in theorem proving papers. The motivation was to reduce the search space (many of superfluous consequences are not created). This strategy gives a solution to the problem posed in [Wos 88]: "to restrict paramodulation at the term level".

The technique of moving all work with equalities in the beginning of inferences has been studied in proof theory [Kan 63, Lif 68, Ore 69], but only for the ground case. It cannot be easily generalized to the general case with variables, since the standard lifting does not work in presence of equality.

The main logical system described in our paper is very simple. If we forget about optimizations, it consists of two inference rules: basic superposition and SLD-resolution. This is achieved by introduction of new predicate symbols. These new predicate symbols collect substitutions solving equations in the program, similar to "answer literals" introduced in [Gre 69].

The transformation proposed by Brandt [Bra 75] achieves similar aims, but his technique requires transitivity axioms to be added, which immediately creates branching in the SLD-tree. He also does not use tabulation. We also note that the ideas of using the basic strategy can be traced to this paper of Brandt [Bra 75].

The technique we use to obtain an equality-free program uses ideas of bottom-up program execution with tabulation. Recursive folding can be implemented via a procedure similar to SLD-

[1]The basic strategy was also explained in our survey translated to English [DV 86].

AL-resolution [Vie 89]. The idea of introducing new predicate symbols to simulate basic strategy is essentially used in the inverse method, as presented in [Vor 92].

Bibliography

[And 70] R. Anderson. Completeness results for E-resolution. In J. Siekmann and G. Wrightson, editors, *Automation of Reasoning. Classical Papers on Computational Logic*, volume 2, pages 317–320. Springer Verlag, 1983. Originally appeared in 1970.

[AD 92] V. Antimirov and A. Degtyarev. Consistency of equational enrichments. In A. Voronkov, editor, *Logic Programming and Automated Reasoning. International Conference LPAR'92.*, volume 624 of *Lecture Notes in Artificial Intelligence*, pages 393–402, St.Petersburg, Russia, July 1992.

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

[BG 90] L. Bachmair and H. Ganzinger. On restriction of ordered paramodulation with simplification. In M.E. Stickel, editor, *Proc. 10th CADE*, volume 449 of *Lecture Notes in Artificial Intelligence*, pages 427–441, 1990.

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

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

[BBLM 86] R. Barbuti, M. Bellia, G. Levi, and M. Martelli. LEAF: a language which integrates logic, equations and functions. In *Logic Programming, Functions, Relations and Equations*, pages 201–238. Prentice Hall, 1986.

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

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

[Bir 44] G. Birkhoff. Subdirect unions in universal algebras. *Bull. Amer. Math. Soc.*, 50:764–768, 1944.

[Bir 67] G. Birkhoff. *Lattice Theory.* AMS Colloquium Publications, 3rd edition, 1967.

[BGM 88] P.G. Bosco, E. Giovanetti, and C. Moiso. Narrowing vs. SLD-resolution. *Theoretical Computer Science*, 59:3–23, 1988.

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

[CL 73] C.L. Chang and R.C.T. Lee. *Symbolic Logic and Mechanical Theorem Proving*. Academic Press, 1973.

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

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

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

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

[DJ 84] N. Dershowitz and N.A. Josephson. Logic programming by completion. In S.-Å. Tärnlund, editor, *2nd International Logic Programming Conference*, pages 312–320, Uppsala, 1984.

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

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

[EM 85] H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification*, volume 1. Springer Verlag, 1985.

[Elc 89] E.W. Elcock. Prolog: Subsumption of equality axioms by the homogeneous form. *Journal of Logic Programming*, pages 45–56, 1989.

[Fri 84] L. Fribourg. Oriented equational clauses as a programming language. *Journal of Logic Programming*, 1:165–177, 1984.

[Fri 85] L. Fribourg. SLOG: A logic programming language interpreter based on clausal superposition and rewriting. In *IEEE International Symposium on Logic Programming*, pages 172–184, Boston, 1985.

[Fri 89] L. Fribourg. A strong restriction of the inductive completion procedure. *Journal of Symbolic Computations*, 8(3):253–276, 1989.

[FHS 89] U. Furbach, S. Hölldobler, and J. Schreiber. Horn equality theories and paramodulation. *Journal of Automated Reasoning*, 5:309–337, 1989.

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

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

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

[Gan 91] H. Ganzinger. A completion procedure for conditional equations. *Journal of Symbolic Computations*, 11:51–81, 1991.

[GM 88] E. Giovanetti and C. Moiso. A completeness result for E-unification algorithms based on conditional narrowing. In *Proceedings of the Workshop on Foundations of Logic and Functional Programming*, volume 306 of *Lecture Notes in Computer Science*, pages 151–167. Springer Verlag, 1988.

[GTW 78] J. Goguen, J. Thatcher, and E. Wagner. An initial algebra approach to the specification, correctness and implementation of abstract data types. In *Current Trends in Programming Methodology*, volume 4, pages 80–149. Prentice Hall, 1978.

[GM 86] J.A. Goguen and J. Meseguer. EQLOG: Equality, types and generic modules for logic programming. In *Logic Programming, Functions, Relations and Equations*, pages 295–363. Prentice Hall, 1986.

[GM 87] J.A. Goguen and J. Meseguer. Models and equality for logic programming. In *TAPSOFT'87*, volume 250 of *Lecture Notes in Computer Science*, pages 1–22. Springer Verlag, 1987.

[GHR 92] J.C. González-Moreno, M.T. Hortalá-González, and M. Rodrí guez Artalejo. On the completeness of narrowing as the operational semantics of functional logic programming. In *CSL'92 (Proc. 6th Workshop on Computer Science Logic)*, volume 702 of *Lecture Notes in Computer Science*, pages 216–230. Springer Verlag, 1992.

[Gre 69] C.C. Green. *The Application of Theorem Proving to Question-Answering Systems.* PhD thesis, Computer Science Department, Stanford University, June 1969.

[Han 94] M. Hanus. The integration of functions into logic programming: from theory to practice. *Journal of Logic Programming*, 19,20:583–628, 1994.

[Her 86] A. Herold. Narrowing techniques applied to idempotent unification. SEKI-Report SR 86-16, Univ. Kaiserslautern, 1986.

[Höl 88] S. Hölldobler. From paramodulation to narrowing. In *5th International Conference on Logic Programming*, pages 327–342, 1988.

[Höl 89] S. Hölldobler. *Foundations of Equational Logic Programming*, volume 353 of *Lecture Notes in Artificial Intelligence*. Springer Verlag, 1989.

[HR 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.

[HO 80] G. Huet and D.C. Oppen. Equations and rewrite rules. In Book, editor, *Formal Languages: Perspectives and Open Problems*. Academic Press, 1980.

[HH 82] G. Huet and J.M. Hullot. Proofs by induction in equational theories with constructors. *Journal of Computer and System Sciences*, 25:239–266, 1982.

[Hul 80] J.M. Hullot. Canonical forms and unification. In *5th CADE*, volume 87 of *Lecture Notes in Computer Science*, pages 318–334, 1980.

[Hus 85] H. Hussmann. Unification in conditional-equational theories. In *Proceedings of the European Conference on Computer Algebra (EUROCAL)*, volume 204 of *Lecture Notes in Computer Science*, pages 543–553, 1985.

[JLM 84] J. Jaffar, J.-L. Lassez, and M.J. Maher. A theory of complete logic programs with equality. *Journal of Logic Programming*, 1(3):211–223, 1983.

[JK 86] J.-P. Jouannaud and E. Kounalis. Automatic proofs by induction in equational theories with constructors. In *Proc. IEEE Symp. on Logic in Computer Science*, pages 358–360, 1986.

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

[Kap 87] S. Kaplan. Simplifying conditional term rewriting systems: Unification, termination and confluence. *Journal of Symbolic Computations*, 4(3):295–334, 1987.

[KB 70] 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.

[Kou 88] E. Kounalis and Rusinowitch. On word problems in horn theories. In E. Lusk and R. Overbeek, editors, *9th CADE*, volume 310 of *Lecture Notes in Computer Science*, pages 527–537, Argonne, Illinois, 1988. Springer Verlag.

[Lan 75] D.S. Lankford. Canonical inference. Technical report, Department of Mathematics, South-Western University, Georgetown, Texas, 1975.

[Len] V.I. Lenin. The three sources and the three parts of Marxism (in Russian). In *Lenin. V.I. The Complete Collection.* Politizdat, Moscow.

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

[Llo 87] J.W. Lloyd. *Foundations of Logic Programming (2nd edition).* Springer Verlag, 1987.

[Maĺ 56] A. Maĺcev. Subdirect products of models. *Dokl. Akad. Nauk SSSR (in Russian)*, 109:264–266, 1956.

[Maĺ 73] A. Maĺcev. *Algebraic Systems.* Springer Verlag, 1973.

[Man 91] A.V. Mantsivoda. Flang: A functional logic language. In H.Boley and M.M.Richter, editors, *Processing Declarative Knowledge (Proceedings of the International Workshop PDK'91*, volume 567 of *Lecture Notes in Artificial Intelligence*, pages 257–270. Springer Verlag, 1991.

[MH 92] A. Middeldorp and E. Hamoen. Counterexamples to completeness results for basic narrowing. In *Proceedings of the 3rd International Conference on Algebraic and Logic Programming*, volume 632 of *Lecture Notes in Computer Science*, pages 244–258, 1992.

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

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

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

[NRS 89] W. Nutt, P. Réty, and G. Smolka. Basic narrowing revisited. *Journal of Symbolic Computations*, 7(3):259–317, 1989.

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

[Pad 88] P. Padawitz. *Computing in Horn Clause Theories*, volume 16 of *EATCS Monographs in Theoretical Computer Science*. Springer Verlag, 1988.

[Pet 83] G.E. Petersson. A technique for establishing completeness results in theorem proving with equality. *SIAM Journal of Computing*, 12(1):82–100, 1983.

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

[Plo 72] G. Plotkin. Building-in equational theories. In *Machine Intelligence*, volume 7. Edinburgh University Press, Edinburgh, 1972.

[Red 85] U.S. Reddy. Narrowing as the operational semantics of functional languages. In *IEEE International Symposium on Logic Programming*, pages 138–151, Boston, 1985.

[Ret 85] P. Réty, C. Kirchner, H. Kirchner, and P. Lescanne. NARROWER: a new algorithm for unification and its application to logic programming. In *Proceedings of the 1st Conference on Rewriting Techniques and Applications, Dijon*, volume 202 of *Lecture Notes in Computer Science*, pages 141–157, 1985.

[Ret 87] P. Réty. Improving basic narrowing techniques. In *Proceedings of the Conference on Rewriting Techniques and Applications*, volume 256 of *Lecture Notes in Computer Science*, pages 228–241, 1987.

[RW 69] G. Robinson and L.T. Wos. Paramodulation and theorem-proving in first order theories with equality. In *Machine Intelligence*, volume 4. Edinburgh University Press, Edinburgh, 1969.

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

[Sel 72] A. Selman. Completeness of calculi for axiomatically defined classes of algebras. *Algebra Universalis*, 2:20–32, 1972.

[She 92] J.C. Sheperdson. SLDNF-resolution with equality. *Journal of Automated Reasoning*, 8:297–306, 1992.

[Sie 90] J.H. Siekmann. An introduction to unification theory. In R.B. Banerji, editor, *Formal Techniques in Artificial Intelligence*, pages 369–425. Elsevier Science, 1990.

[Sla 74] 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.

[Soc 92] R. Socher-Ambrosius. A goal-oriented strategy based on completion. In *Proc. ALP'92*, volume 632 of *Lecture Notes in Computer Science*, pages 435–445. Springer Verlag, 1992.

[VL 84] M.H. Van Emden and J.W. Lloyd. A logic reconstruction of prolog ii. In S.-Å. Tärnlund, editor, *2nd International Logic Programming Conference*, pages 35–40, Uppsala, 1984.

[VE 87] M.H. Van Emden and K. Yukawa. Logic programming with equations. *Journal of Logic Programming*, 4:265–288, 1987.

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

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

[Wos 88] L. Wos. *Automatic Reasoning: 33 Basic Research Problems.* Prentice Hall, 1988.

[Yam 87] A. Yamamoto. A theoretical combination of SLD-resolution and narrowing. In *Proc. ICLP'87*, pages 470–487. MIT Press, 1987.

[ZK 88] H. Zhang and D. Kapur. First-order theorem proving using conditional rewrite rules. In *9th International Conference on Automated Deduction*, volume 310 of *Lecture Notes in Computer Science*, pages 1–20, 1988.