UPMAIL Technical Report No. 101
April 11, 1995
Revised June 11, 1997
ISSN 1100-0686

Handling Equality in Logic Programming via Basic Folding

Anatoli Degtyarev*
Computing Science Department
Uppsala University
Box 311, S-751 05 Uppsala,
Sweden
email anatoli@csd.uu.se

Yuri Koval†
Department of Cybernetics
Kiev University
Vladimirskaya 64
Kiev 252017
Ukraine
email smith@forge.kiev.ua

Andrei Voronkov‡
Computing Science Department
Uppsala University
Box 311, S-751 05 Uppsala,
Sweden
email voronkov@csd.uu.se

*Supported by grant from the Swedish Institute and the Swedish Royal Academy of Sciences
†Partially supported by a grant from the Swedish Royal Academy of Sciences
‡Supported by a TFR grant

Abstract

We introduce a new approach to handling equality in logic programs. The method is based on the transformation of equational logic programs into logic programs without equality using basic superposition and the basic folding. We prove soundness, completeness and the existence of terminating strategies. The efficiency of the method is demonstrated on examples. An implementation is briefly discussed.[2]

Section 1

Introduction

This paper further develops an approach to logic programming with equality introduced in [14]. By a logic program with equality we understand a Horn clause logic program in which equality may occur both in the heads and in the bodies of clauses. The model-theoretic semantics of such programs is well known since long ago [7, 43]. However, there is no generally accepted procedural interpretation. The most natural generalization of SLD-resolution known as SLDE-$^+$-resolution [23] have been proved in general incomplete in [14]. Various definitions of procedural interpretations are actively studied in the theorem proving and logic programming communities.

The main influence on modern techniques of equational logic programming was made by equational theorem proving methods using refinements of inference in the paramodulation-based calculus via ordering strategies and deletion of redundant clauses. This technique appeared in [37, 55, 39] and has been further developed in [33, 50, 30, 62, 53, 56, 3, 47]. In logic programming, it was introduced in many papers, including [21, 57, 60, 36, 25, 49, 24, 26, 45], especially in the framework of integration of logic and functional programming languages.

In [14] we introduced a new method called *equality elimination.* This method performs all equality-related reasoning steps before all other inference steps using a special strategy — basic superposition [46, 4]. 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 using complete sets of E-unifiers (SLDE$^+$-resolution [22]) is incomplete. An analysis of semantics of logic programs with equality based on complete sets of E-unifiers is contained in [17].

Basic superposition used in the equality elimination method makes severe restrictions on equality rules, while preserving completeness. However basic superposition does not guarantee termination — there may be infinite chains of superpositions. The main aim of this paper is to enrich the equality elimination method by a new inference rule called the *basic folding.* By adding the basic folding, we can ensure termination of the equality elimination procedure, thus allowing us to obtain elegant finite programs without equality from equational logic programs. Moreover, the resulting program without equality can sometimes give a unique (canonical) solution from an infinite number of solutions to the original program.

The basic folding method is complete and sound for *arbitrary* logic programs with equality.

Most existing approaches use special conditions on the set of equations which may occur in programs (see, e.g. the recent survey [28]), like confluence and/or termination. Such conditions ensure comparatively efficient operational semantics (e.g. some sophisticated narrowing strategies). Unfortunately, these conditions are not easy to verify. For example, both confluence and termination are not even semi-decidable in general. Even for the decreasing conditional rewrite rules having lots of desirable properties confluence is only semi-decidable. Thus, we have a family of programming languages for which the notion of a program is undecidable. The set of programs is decidable only for non-standard notions of completeness, for example w.r.t. *strict equality* and very refined restrictions on the functional logic programs.

The basic folding method does not impose any conditions on the equality theory. In particular, the set of all positive equations need not be confluent or need not only occur in facts. At the same time, the use of orderings and simplification can significantly reduce both the number of clauses in the final program and the time needed to obtain these clauses.

A preliminary version of this paper appeared in [16].

Section 2

An Informal Example

Since the formal definition of the basic folding is a bit tricky, we shall consider the main ideas on an example. The inference system we use consists of three rules: basic superposition (in fact, substitution of equal for equal with some restrictions), equality solution and the basic folding (introduction of a new definition and its use to "fold" program clauses).

Consider the following program $\mathcal{P}$[1]:

$$v \cdot v \simeq v \leftarrow P(v)$$
$$P(u) \leftarrow u \simeq f(a \cdot a) \tag{2.1}$$
$$query(x,y) \leftarrow f(x \cdot x) \cdot f(y) \simeq f(x \cdot x) \tag{2.2}$$

There are three instances of the term $f(y \cdot y)$ in this set of clauses. The *basic folding rules* allow us to introduce a new predicate $E(x,y)$ expressing the fact $x \simeq f(y \cdot y)$ together with its definition $E(x,y) \leftarrow x \simeq f(y \cdot y)$ and replace all instances of $f(y \cdot y)$ using $E(x,y)$. Applying the basic folding to (2.1) and (2.2) we obtain a new set of clauses

$$v \cdot v \simeq v \leftarrow P(v)$$
$$P(u) \leftarrow u \simeq y, E(y,x), x \simeq a \tag{2.3}$$
$$query(x,y) \leftarrow z \cdot f(y) \simeq z, E(z,x) \tag{2.4}$$
$$E(x,y) \leftarrow x \simeq f(y \cdot y) \tag{2.5}$$

There is a positive equation $v \cdot v \simeq v \leftarrow P(v)$ in this clause set. It means that we can substitute any term $t \cdot t$ by t provided that $P(t)$ is a logical consequence of the program. The *basic superposition rule* allows us to make this substitution of terms for equal terms. Applying it into clauses (2.4) and (2.5) yields two new clauses

$$query(x,y) \leftarrow P(f(y)), f(y) \simeq f(y), E(f(y),x) \tag{2.6}$$
$$E(x,y) \leftarrow P(y), x \simeq f(y) \tag{2.7}$$

[1]As we have shown in [14], this program demonstrates the incompleteness of SLDE$^+$-resolution [23] based on complete sets of E-unifiers [51]

The *equality solution* rule allows us to solve an equation $s \simeq t$ with unifiable s, t occurring in the body of a clause by removing the equation and applying a most general unifier of s, t to the rest of the clause. We can apply this rule to clauses (2.3), (2.5), (2.6) and (2.7) obtaining

$$P(u) \leftarrow E(u, a) \tag{2.8}$$

$$E(f(y \cdot y), y) \tag{2.9}$$

$$query(x, y) \leftarrow P(f(y)), E(f(y), x) \tag{2.10}$$

$$E(f(y), y) \leftarrow P(y) \tag{2.11}$$

There are further possible applications of these inference rules to the clauses generated so far. The theory developed in Sections 4 and 5 tells us that the derivation we have obtained is *saturated.* It means that we can stop the transformations and remove all clauses containing equality obtaining the logic program without equality consisting of four clauses (2.8)–(2.11).

The soundness and completeness theorems of Section 5 imply that the original logic program with equality is in some sense equivalent to this logic program without equality. There is a correct answer substitution $\{a \cdot a/y, a/x\}$ for this program and the query $query(x, y)$. In addition, SLD-tree for this program and the query $query(x, y)$ is finite.

Section 3

Preliminaries

We present here a brief overview of notions and preliminary definitions necessary for understanding the paper. For a more thorough coverage, see books and overviews [40, 32, 42, 31, 54, 18, 28]. We assume the basic knowledge of substitutions and unification.

All formulas belong to a fixed signature Σ containing at least one constant and the binary equality symbol $\simeq$. For any formula φ all whose free variables are $x_1, \ldots, x_n$, we denote by $\forall\varphi$ the formula $\forall x_1 \ldots \forall x_n \varphi$. The notation $\rightleftharpoons$ stands for "equal by definition".

By X we denote a set of variables. The set of all terms in the signature Σ with variables from X is denoted by $T_\Sigma(X)$. By T_Σ we denote $T_\Sigma(\emptyset)$, i.e. the set of all ground terms in the signature Σ.

We write $A[s]$ to indicate that an expression A contains s as a subexpression and denote by $A[t]$ the result of replacing the occurrence of s in A by t. By $A\sigma$ we denote the result of applying the substitution σ to A, any such $A\sigma$ is called an *instance* of A.

The domain $dom(\sigma)$ of the substitution σ is the set $\{x \mid x\sigma \neq x\}$. The *range* $ran(\sigma)$ of the substitution σ is defined by $ran(\sigma) \rightleftharpoons \{x\sigma \mid x \in dom(\sigma)\}$. Substitutions θ with the domain $x_1, \ldots, x_n$ is denoted by $\{x_1\theta/x_1, \ldots, x_n\theta/x_n\}$. The empty substitution is denoted by ε.

Let σ_1, σ_2 be substitutions such that $dom(\sigma_1) \cap dom(\sigma_2) = \emptyset$. By $\sigma_1 \cup \sigma_2$ we denote the substitution σ such that $dom(\sigma) \rightleftharpoons dom(\sigma_1) \cup dom(\sigma_2)$ and

$$x\sigma \rightleftharpoons \begin{cases} x\sigma_1, & \text{if } x \in dom(\sigma_1) \\ x\sigma_2, & \text{if } x \in dom(\sigma_2) \end{cases}$$

for all $x \in dom(\sigma)$.

The symbol $\vdash$ stands for provability *in classical first order logic with equality.*

Definition 3.1 (equation, clause)
An *equation* is an atomic formula of the form $s \simeq t$. We shall not distinguish the formula $s \simeq t$ from $t \simeq s$.

A *clause* is any formula of the form $A_1 \wedge \ldots \wedge A_n \supset A$, where $n \geq 0$, denoted

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

When $n = 0$, the symbol $\leftarrow$ will be omitted. A *logic program* is a finite set of clauses.

By a *ground expression* (i.e. term, equation, clause etc.) we mean an expression containing no variables.

By $var(t)$ ($var(C)$) we denote the set of all variables of a term t (clause C). For a substitution σ, $var(\sigma) \rightleftharpoons var(dom(\sigma)) \cup var(ran(\sigma))$.

Definition 3.2 (reduction ordering)
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_Σ. For the rest of the paper $\succ$ stands for a fixed reduction ordering.

Following [5] we distinguish terms occurring in the original formula from terms introduced by a substitution by using *closures.*

Definition 3.3 (closure)
A *closure* is a pair $C \cdot \sigma$ such that

1. C is a clause;
2. σ a substitution such that $dom(\sigma) \subseteq var(C)$;
3. whenever an atomic formula $P(t_1, \ldots, t_n)$ occurs in C and P is a predicate symbol different from $\simeq$, then all terms t_i are variables.

The application of a substitution ρ to such a closure is defined by $(C \cdot \sigma)\rho \rightleftharpoons C \cdot \sigma\rho$.

The set $var(C \cdot \sigma)$ is defined as $var(C) \cup var(\sigma)$. Two closures $\mathcal{C}_1, \mathcal{C}_2$ have *disjoint variables* iff $var(\mathcal{C}_1) \cap var(\mathcal{C}_2) = \emptyset$.

Condition (3) is introduced to prevent the use of non-equality literals in basic superposition or basic folding introduced below.

We make two further conventions of a technical nature. First, we assume that all substitutions be idempotent [19]. Second, whenever we use the closure notation $C \cdot \sigma$ such that $dom(\sigma) \not\subseteq var(C)$, we mean the closure $C \cdot \tau$, where τ is a restriction of σ on $var(C)$. These two conventions will considerably simplify notation.

Definition 3.4 (position)
A *position* is a finite sequence of positive integers. The notions of a *position in a term* t and the subterm $t \mid_\pi$ of t at the position π are defined inductively as follows:

1. The *empty position* λ is a position in any term t and $t \mid_\lambda \rightleftharpoons t$.
2. Let the term t have the form $f(t_1, \ldots, t_n)$, where $n \geq 1$, π be a position in t_i and $t_i \mid_\pi = s$. Then $i\pi$ is a position in t and $t \mid_{i\pi} \rightleftharpoons s$.

A position π in t is a *non-variable position* iff $t \mid_\pi$ is not a variable.

By $t[\pi \leftarrow s]$ we denote the term obtained from t by substituting the term s in the position π.

The notion of a position is naturally extended to atoms and clauses. Thus, we can speak about non-variable positions in a clause. A non-variable position in a closure $C \cdot \sigma$ is a non-variable position in C.

Definition 3.5 (variants)
Two literals, terms or clauses are *variants* iff they are instances of each other.

Two closures $C_1 \cdot \sigma_1$ and $C_2 \cdot \sigma_2$ are *variants* iff $C_1\sigma_1$ is a variant of $C_2\sigma_2$ and the clauses C_1 and C_2 have the same non-variable positions.

In the following we identify closures which are variants of each other.

The closure $C \cdot \sigma$ represents the clause $C\sigma$. Intuitively, C contains only basic positions (positions from the original set of clauses) and σ denotes a substitution introducing non-basic positions which appeared as a result of applications of unification in inference rules. When we use a closure $C \cdot \sigma$ on either side of the symbol $\vdash$, it can be replaced by the clause $C\sigma$.

The *basic superposition* rules derive closures from closures. They are defined as in [5]. We always assume that premises of rules have disjoint variables. As usual, to make variables disjoint we can replace closures by their variants.

Definition 3.6 (basic superposition)
The *basic superposition rules* are the following inference rules:

Basic right superposition

$$\frac{(s \simeq t \leftarrow T_1) \cdot \sigma_1 \quad (u[s'] \simeq v \leftarrow T_2) \cdot \sigma_2}{(u[t] \simeq v \leftarrow T_1, T_2) \cdot (\sigma_1 \cup \sigma_2)\rho}\ \mathcal{RS}$$

where

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

Conditions 2 and 3 are be called *the ordering conditions.*

Basic left superposition

$$\frac{(s \simeq t \leftarrow T_1) \cdot \sigma_1 \quad (H \leftarrow T_2, u[s'] \simeq v, T_3) \cdot \sigma_2}{(H \leftarrow T_2, T_1, u[t] \simeq v, T_3) \cdot (\sigma_1 \cup \sigma_2)\rho}\ \mathcal{LS}$$

with the same conditions as for basic right superposition and an additional restriction: the equality symbol $\simeq$ does not occur in T_2. (The last condition can be considered as a special case of a *selection function* [3].)[1]

We need one more inference rule. The *equality solution rule* derives closures from closures:

[1]The reason why *left* superposition is a superposition into the tail (i.e. into the *right* hand side of the clause) is historical: superposition is often defined as an inference rule on *sequents,* i.e. expressions of the form $A_1, \dots, A_n \to B_1, \dots, B_m$, then left superposition is a superposition in one of A_i. A clause $H \leftarrow T$ corresponds to the sequent $T \to H$, thus left superposition becomes a superposition into the tail.

Definition 3.7 (equality solution)
The *equality solution rule* is the following inference rule on closures:

$$\frac{H \leftarrow T_1, s \simeq t, T_2 \cdot \sigma}{H \leftarrow T_1, T_2 \cdot \sigma\rho} \; \mathcal{ES}$$

where ρ is a most general unifier of $s\sigma, t\sigma$ and T_1 has no occurrences of $\simeq$.

In the subsequent sections we shall describe a program transformation system, i.e. a system transforming programs (sets of closures) into programs (sets of closures). We shall formalize it as a transition system on sets of closures. To this end we represent logical calculi acting on sets of closures as sets of transition rules.

Let Ω be any calculus of closures, i.e. any set of inference rules inferring closures from closures. The calculus Ω can be considered as an inference system Ξ allowing to derive *sets of closures* from sets of closures in the following way: the inference rule

$$\frac{S}{S \cup \{\mathcal{C}\}}$$

where S is a set of closures, is an inference rule of Ξ iff there are closures $\mathcal{C}_1, \ldots, \mathcal{C}_n \in S$ such that the rule

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

belongs to Ω. Thus, in the subsequent sections, basic superposition and equality solution will also be considered as rules on sets of closures. We shall denote inference rules acting on sets of closures

$$\frac{S_1}{S_2}$$

by $S_1 \Rightarrow S_2$ in order to stress that such rules can be considered as transitions. We need to consider ordinary inference rules as rules on sets of closures because we shall also use basic folding rules *replacing* closures by other closures which can be easily formalized as transitions.

Section 4

The Basic Folding

The basic folding is based on a transformation of sets of closures removing the equality predicate from the program. Unlike the equality elimination method studied in [14], there are strategies for the basic folding which always terminate resulting in a finite program. The essence of the basic folding method is the following. We systematically try to solve equations in the bodies of closures using basic superposition. When there are growing terms so that basic superposition transformations do not terminate, we introduce a new predicate definition via the basic folding rule and "fold" the closure containing the terms.

Let $\mathcal{X} = x_0, x_1, x_2, \ldots$ be a fixed countable sequence of variables.

Definition 4.1 ($\mathcal{X}$-normal term)
A term t is called *$\mathcal{X}$-normal* iff its variables (in the order of their first occurrences in t) are $x_1, \ldots, x_n$. The *$\mathcal{X}$-normal form* of a term t is the variant of t which is $\mathcal{X}$-normal.

For example, the $\mathcal{X}$-normal form of the term $g(x, y, f(a, x))$ is $g(x_1, x_2, f(a, x_1))$.

Let S be a set of closures. By $\mathcal{R}(S)$ we denote the set of clauses

$$\mathcal{R}(S) \rightleftharpoons \{C\sigma \mid C \cdot \sigma \in S\}$$

Let $\mathcal{P}$ be a logic program in a signature Σ. We assume that the program contains a definition of the predicate *query* of n arguments.

Definition 4.2 (answer a program)
An *answer* to the program $\mathcal{P}$ is any tuple $\langle t_1, \ldots, t_n \rangle$ of terms, where n is the arity of the predicate *query* in $\mathcal{P}$. Such an answer is a *correct answer to* $\mathcal{P}$ iff $\mathcal{P} \vdash query(t_1, \ldots, t_n)$.

This definition can be rephrased for closures:

Definition 4.3 (answer to a set of closures)
An *answer* to the set of closures S is any tuple $\langle t_1, \ldots, t_n \rangle$ of terms, where n is the arity of the predicate *query* in S. Such an answer is a *correct answer to* S iff $\mathcal{R}(S) \vdash query(t_1, \ldots, t_n)$.

From now on we assume that the signature Σ is fixed and all logic programs are written in this signature. Let Pr be a countable set of predicate symbols disjoint from predicate symbols of Σ. Let there be a mapping that associates with each $\mathcal{X}$-normal term t a unique predicate symbol $\mathcal{E}_t \in Pr$. The arity of $\mathcal{E}_t$ is the number of different variables in t plus 1. We shall use the new predicate symbols $\mathcal{E}_t$ for introducing new predicate definitions for the atoms $x_0 \simeq t$.

Definition 4.4 (flattening)
Let τ be a substitution with $dom(\tau) = \{y_1, \ldots, y_m\}$. The *flattening* of τ (denoted $\mathcal{S}_\tau$) is the sequence of equations $y_1 \simeq y_1\tau, \ldots, y_m \simeq y_m\tau$.

Let t be an $\mathcal{X}$-normal term. The *E-definition* of t (denoted Def_t) is the closure

$$\mathcal{E}_t(x_0, x_1, \ldots, x_n) \leftarrow x_0 \simeq t \cdot \varepsilon$$

The following definition is the main definition of this section.

Definition 4.5 (basic folding)
The *basic left folding rule* is defined as follows. Assume that S is a set of closures and $\mathcal{C} \in S$ is a closure of the form:

$$H \leftarrow T_1, A[t\tau], T_2 \cdot \theta$$

where t is not a variable. Let the term t' be $\mathcal{X}$-normal, variables $x_1, \ldots, x_n$ be all variables of t', the variables $x_0, x_1, \ldots, x_n$ do not occur in $var(\mathcal{C}) \cup var(\tau)$, $dom(\tau) = \{y_1, \ldots, y_m\}$, every y_i has a unique occurrence in t[1], and $y_1, \ldots, y_m$ do not occur in $var(\mathcal{C})$. Let $\mathcal{C}'$ be the closure

$$H \leftarrow T_1, A[x_0], \mathcal{E}_{t'}(x_0, x_1, \ldots, x_n), \mathcal{S}_\tau, T_2 \cdot \sigma \cup \theta$$

such that

1. Every non-variable position π in t is a non-variable position in t';
2. $dom(\sigma) = \{x_1, \ldots, x_n\}$;
3. $t'\sigma = t\theta$;
4. $\mathcal{C}$ is not a variant of $Def_{t'}$.

Then the basic left folding rule is the rule

$$\frac{S}{S \setminus \{\mathcal{C}\} \cup \{\mathcal{C}', Def_{t'}\}} \; \mathcal{BF}$$

The *basic right folding rule* is defined in the same way as the basic left folding rule, but with $\mathcal{C}, \mathcal{C}'$ defined as

$$A[t\tau] \leftarrow T \cdot \theta$$

and

$$A[x_0] \leftarrow \mathcal{E}_{t'}(x_0, x_1, \ldots, x_n), \mathcal{S}_\tau, T \cdot \sigma \cup \theta$$

respectively. We also say that $\mathcal{C}'$ is obtained by a basic folding *into* $\mathcal{C}$.

Below we shall prove that one can use a weaker form of the folding:

$$\frac{S}{S \setminus \{\mathcal{C}\} \cup \{\mathcal{C}'\}}$$

provided that the predicate $\mathcal{E}_{t'}$ occurs in S.

Consider an example.

[1]The requirement that every y_i have a unique occurrence in t can be dropped, but it simplifies many proofs.

Example 4.6 Let $\mathcal{C}$ be the closure

$$f(y,z) \simeq a \leftarrow B(y,z), f(g(z),y) \simeq b \cdot \{b/z, h(a)/y\}$$

Following the notation of Definition 4.5, let

$$\begin{array}{l} A[x_0] \rightleftharpoons x_0 \simeq b \\ \theta \rightleftharpoons \{b/z, h(a)/y\} \\ t \rightleftharpoons f(y_1, y) \\ \tau \rightleftharpoons \{g(z)/y_1\} \\ t' \rightleftharpoons f(x_1, h(x_2)) \\ Def_{t'} \rightleftharpoons \mathcal{E}_{f(x_1,h(x_2))}(x_0, x_1, x_2) \leftarrow x_0 \simeq f(x_1, h(x_2)) \cdot \varepsilon \\ \mathcal{S}_\tau \rightleftharpoons y_1 \simeq g(z) \end{array}$$

By the condition $t\theta = t'\sigma$ we infer $\sigma = \{y_1/x_1, a/x_2\}$. Then $\mathcal{C}'$ has the form

$$f(y,z) \simeq a \leftarrow B(y,z), x_0 \simeq b, \mathcal{E}_{f(x_1,h(x_2))}(x_0, x_1, x_2), y_1 \simeq g(z) \cdot \{b/z, h(a)/y, y_1/x_1, a/x_2\}$$

Lemma 4.7 Let $\{\mathcal{C}\} \Rightarrow \{\mathcal{C}', Def_{t'}\}$ be an application of the basic folding. Then $\mathcal{C}', Def_{t'} \vdash \mathcal{C}$.

Proof. Straightforward. □

In the definition of the basic folding, we allow to "fold" closures using the definition for a term t' which can be different from t. For completeness, it is enough to only fold using t. However such a restriction of the basic folding would disallow some elegant applications of the folding, like the one used below in Example 6.2. In fact the idea of using t' instead of t has been obtained during experiments with our first implementation.

Some further explanation of the intended meaning of the substitution τ used in the basic folding rules is given in Example 6.1 in Section 6. We need to keep $\mathcal{S}_\tau$ for not losing non-variable positions which may result in incompleteness.

The basic folding method uses the logical system **BF** defined below that acts on sets of closures of a special kind. It always starts with a set of *initial* closures:

Definition 4.8 (ε-closure, initial closure)
A closure S is called an *ε-closure* iff it has the form $C \cdot \varepsilon$. A closure S is called *initial* iff it is an ε-closure and predicate symbols $\mathcal{E}_t$ do not occur in S.

There are many ways of transforming a logic program into a set of initial closures. One way which is a folklore in logic programming is the following: all clauses C of the form

$$H \leftarrow T_1, A(t_1, \ldots, t_n), T_2$$

where A is not an equation, are repeatedly replaced by

$$H \leftarrow T_1, y_1 \simeq t_1, \ldots, y_n \simeq t_n, A(y_1, \ldots, y_n), T_2 \cdot \varepsilon$$

and all clauses of the form

$$A(t_1, \ldots, t_n) \leftarrow T$$

where A is not an equation and some t_i is not a variable, are repeatedly replaced by

$$A(y_1, \ldots, y_n) \leftarrow y_1 \simeq t_1, \ldots, y_n \simeq t_n, T \cdot \varepsilon$$

where $y_1, \ldots, y_n$ are new variables. Another, more complicated way is described in [14] and is based on the introduction of new predicate symbols for all complementary pairs of literals in the program.

Definition 4.9 (system BF)
The inference system **BF** is the set of inference rules acting on sets of closures and consisting of 3 inference rules:

1. basic (left and right) superposition;

2. equality solution;

3. basic (left and right) folding.

A **BF**-*derivation* Δ is any sequence of sets $S_0 \Rightarrow S_1 \Rightarrow \ldots$ such that every S_{i+1} is obtained from S_i by an application of an inference rule of **BF**.

The *limit* S_ω of the **BF**-derivation Δ is the set $\bigcup_j \bigcap_{k \geq j} S_k$ of closures. The derivation Δ is called *fair* iff for any application of basic superposition or equality solution of the form

$$\frac{S_\omega}{S_\omega \cup \{\mathcal{C}\}}$$

we have $\mathcal{C} \in S_i$, for some i.

A derivation is called *saturated* iff it is finite and fair.

Note that the limit of a saturated derivation $S_0 \Rightarrow \ldots \Rightarrow S_n$ is its last set S_n.

Lemma 4.10 Let $\Delta = (S_0 \Rightarrow S_1 \Rightarrow \ldots)$ be a **BF**-derivation. Then for every i we have $S_{i+1} \vdash S_i$.

Proof. For applications of the folding it follows from Lemma 4.7. For equality solution and superposition it is obvious because in these cases $S_i \subseteq S_{i+1}$. □

For a set of closures S, by S^- we denote a set of closures obtained from S by the removal of *all* closures in which the equality symbol $\simeq$ occurs.

Given an equational logic programs represented as a set of initial closures, the basic folding method tries to construct a saturated derivation. When such a derivation $S_0 \Rightarrow \ldots \Rightarrow S_n$ is found, all clauses containing the equality predicate are removed from the final set of clauses $\mathcal{R}(S_n)$. In Section 5 we prove the existence of a saturated derivation and soundness and completeness of the method. Soundness and completeness mean that for any fair derivation the given equational logic program S_0 and the final logic program $\mathcal{R}(S_\omega^-)$ without equality are, in some sense, equivalent. In Section 6 we consider examples. Section 7 discusses related works. In Section 8 we describe an implementation of the basic folding. The work of this system on some test problems is shown in Appendices A and B.

Section 5

Soundness and Completeness

In this section we prove the main properties of the basic folding, namely soundness, completeness and termination.

We start with a fundamental lemma which will be used in many proofs of this paper. It states that any sequence of foldings into a closure terminates.

Lemma 5.1 Let S_0 be a finite set of closures. Then there is no infinite **BF**-derivation from S_0 consisting of only foldings.

Proof. Immediate from the following observations:

1. Every folding replaces a closure $\mathcal{C}$ with two closures $\mathcal{C}', Def_t$ such that $\mathcal{C}'$ has less non-variable positions than $\mathcal{C}$ and Def_t has the form $D \cdot \varepsilon$.
2. Every folding into a closure of the form $D \cdot \varepsilon$ gives two closures of the same form, each one having less non-variable positions than $D \cdot \varepsilon$.

□

The following lemma shows that all rules of **BF** preserve equivalence w.r.t. the set of derivable atoms in the original signature, if the input set consists of closures of the original signature:

Lemma 5.2 Let S_0 be a set of initial closures, $S_0 \Rightarrow S_1 \Rightarrow \ldots$ be a **BF**-derivation, S_ω be the limit of this derivation and $\bar{t} = \langle t_1, \ldots, t_n \rangle$ be an arbitrary tuple of terms. Then the following conditions are equivalent

1. $\bar{t}$ is a correct answer for S_0;
2. $\bar{t}$ is a correct answer for S_i, where $i > 0$;
3. $\bar{t}$ is a correct answer for S_ω.

Proof.

$1 \Rightarrow 2$ From Lemma 4.10 it follows that $S_i \vdash S_0$. Hence, every correct answer to S_0 is a correct answer to S_i.

$2 \Rightarrow 1$ Let A be any atom in the original signature Σ. Consider the set of formulas

$$S_{\equiv} \rightleftharpoons S_0 \cup \{\forall(x_0 \simeq t \equiv \mathcal{E}_t(x_0, \ldots, x_n)) \mid t \text{ is an } \mathcal{X}\text{-normal term}\}$$

Since $S_{\equiv}$ is an extension of S_0 by definitions of new predicate symbols, $S_{\equiv}$ conservatively extends S_0 over Σ, i.e. for every formula φ of the signature Σ we have $S_{\equiv} \vdash \varphi$ iff $S_0 \vdash \varphi$. By induction on i one can easily prove that for every closure $\mathcal{C} \in S_i$ we have $S_{\equiv} \vdash \mathcal{C}$. (Basic superposition and equality solution just add logical consequences to S_i; basic folding adds consequences to $S_{\equiv}$ because $S_{\equiv}$ contains every formula of the form $\forall(x_0 \simeq t \equiv \mathcal{E}_t(x_0, \ldots, x_n))$.) Thus, we have $S_{\equiv} \vdash S_i$. Let $\bar{t}$ be a correct answer to S_i, i.e. $S_i \vdash query(\bar{t})$. Then we have $S_{\equiv} \vdash query(\bar{t})$. Since $S_{\equiv}$ is conservative over S_0, we have $S_0 \vdash query(\bar{t})$, i.e. $\bar{t}$ is a correct answer for S_0.

$1 \Rightarrow 3$ Let $S_0 \vdash query(\bar{t})$. By compactness, there is a finite subset $\{\mathcal{C}_1, \ldots, \mathcal{C}_n\} \subseteq S_0$ such that $\mathcal{C}_1, \ldots, \mathcal{C}_n \vdash query(\bar{t})$. Consider the set D of all clauses $\mathcal{D} \in S_\omega$ obtained by repeated applications of the basic folding to $\{\mathcal{C}_1, \ldots, \mathcal{C}_n\}$. By Lemma 5.1, D is finite. Applying Lemma 4.7 we obtain $\mathcal{D}_1, \ldots, \mathcal{D}_m \vdash \mathcal{C}_i$, for all $i \in \{1, \ldots, n\}$. Hence $\mathcal{C}_\omega \vdash \mathcal{C}_i$, for all $i \in \{1, \ldots, n\}$. This implies $S_\omega \vdash query(\bar{t})$.

$3 \Rightarrow 1$ Every closure $\mathcal{C} \in S_\omega$ belongs to some S_i. Hence, $S_{\equiv} \vdash \mathcal{C}$. It follows that $S_{\equiv} \vdash S_\omega$. Then use conservativity of $S_{\equiv}$ over S_0.

□

Theorem 5.1 (soundness)
Let S_0 be a set of initial closures, $\Delta = (S_0 \Rightarrow S_1 \Rightarrow \ldots)$ be any **BF**-derivation, and S_ω be the limit of Δ. Then any correct answer for S_ω^- is a correct answer for S_0.

Proof. Straightforward from Lemma 5.2 and $S_\omega^- \subseteq S_\omega$. □

Theorem 5.2 (completeness)
Let S_0 be a set of initial closures, $\Delta = (S_0 \Rightarrow S_1 \Rightarrow \ldots)$ be any fair derivation, and S_ω be the limit of Δ. If S_0 has a correct answer then S_ω^- has a correct answer.

The (lengthy) proof of this theorem is given in Appendix C.

Note that Theorem 5.2 and the compactness theorem imply that there is n such that $\mathcal{R}(S_n^-)$ has a correct answer. Also note that by Theorem 5.1 every correct answer to $\mathcal{R}(S_n^-)$ is a correct answer to $\mathcal{R}(S_0)$.

Also one can note that we can use a better strategy for folding: it is possible not to add the closure $Def_{t'}$ into the new set of closures provided only that the predicate $\mathcal{E}_{t'}$ occurs in S. We shall not prove this formally but illustrate on an example. Assume that we introduced three new definitions:

$$\mathcal{E}_{f(g(x_1))}(x_0, x_1) \leftarrow x_0 \simeq f(g(x_1)) \cdot \varepsilon \tag{5.1}$$

$$\mathcal{E}_{f(x_1)}(x_0, x_1) \leftarrow x_0 \simeq f(x_1) \cdot \varepsilon \tag{5.2}$$

$$\mathcal{E}_{g(x_1)}(x_0, x_1) \leftarrow x_0 \simeq g(x_1) \cdot \varepsilon \tag{5.3}$$

Then two basic foldings into closure (5.1) will replace it by

$$\mathcal{E}_{f(g(x_1))}(x_0, x_1) \leftarrow \mathcal{E}_{f(x_1)}(x_0, y), \mathcal{E}_{g(x_1)}(y, x_1) \cdot \varepsilon \tag{5.4}$$

Applying resolution, from closures (5.2), (5.3) and (5.4) we can obtain the closure

$$\mathcal{E}_{f(g(x_1))}(x_0, x_1) \leftarrow x_0 \simeq f(y), y \simeq g(x_1) \cdot \varepsilon \tag{5.5}$$

One can always safely replace (5.1) by (5.5), see Lemma C.20 given in the appendix.

The completeness theorem here may seem not very striking since it does not characterize the relation between the computed answer substitutions and all correct answer substitutions. Thus, it looks like a satisfiability statement. Let us consider this aspect in detail. It was precisely one of the main aims of the basic folding to avoid the generation of *all* correct answers. The known operational semantics of functional logic programming (see [28]) also compute only *some* answers. The set of computed answer substitutions for the known operational principles of functional logic programs is *complete* in the sense that for every correct answer substitution θ there is a computed answer substitution σ such that $\sigma \leq_E \theta$. The problem is that the a complete set of answer substitutions w.r.t. an equational theory E does not give much information about the set of all correct answer substitutions. An answer substitution σ for a query Q gives no information on whether another substitution θ is also a correct answer (the problem $x\theta \simeq_E x\sigma$ may be undecidable, even when θ, σ are ground and E is represented by a canonical conditional term rewrite system without extra variables [35]). It means that if we would like to know whether θ is a correct answer substitution to Q we have to ask the query $Q\theta$ instead. But we can as well ask $Q\theta$ in our setting and our completeness theorem guarantees a correct answer in the same way as the completeness theorems for functional logic programs. Thus, the difference between the two formulations of completeness is really marginal. In addition, the complete sets of solutions (complete sets of E-unifiers) do not behave well for equational logic programs in general. For example, in [14] we show that the minimal model of a program is not a fixpoint when we use complete sets of unifiers in the definition of the immediate consequence operator.

Following [45], we could have changed the completeness statement in a way more traditional for logic programming via answer substitutions using a suitable notion of minimality. However it requires several new notions and a different technique. We plan to further investigate this subject.

Theorem 5.3 (termination)
For any set of initial closures S_0 there is a saturated derivation $S_0 \Rightarrow \ldots \Rightarrow S_n$.

Proof. By multiple applications of the basic folding we can obtain a set S_k such that all closures in S_k containing equality in the head have the form

$$x \simeq y \leftarrow T \cdot \varepsilon$$

where T contains only variable positions. Now apply, while possible, all basic superposition and equality solution rules. Any application of basic superposition in such a set makes the number of non-variable positions in the conclusion less than the number of non-variable positions in one of the premises. This implies that after a finite number of steps no more new closures can be obtained. Thus, the resulting set S_n is saturated. □

Note that by Theorems 5.1 and 5.2 it follows that for every saturated derivation $S_0 \Rightarrow \ldots \Rightarrow S_n$

1. Every correct answer to $\mathcal{R}(S_n^-)$ is a correct answer to $\mathcal{R}(S_0)$;

2. If $\mathcal{R}(S_0)$ has a correct answer then $\mathcal{R}(S_n^-)$ has a correct answer.

Theorems 5.1, 5.2 and 5.3 form the theoretical basis for the basic folding. The main aim of the method is to obtain a saturated derivation from the set of initial closures representing a logic program with equality. Then we can remove all closures containing equality from the limit set of the saturated derivation. The above stated properties guarantee that the resulting logic program without equality is sound and complete w.r.t. the initial equational logic program.

Section 6

Examples

In examples, we shall often replace closures by their variants.

We consider the definition of the basic folding on an example. As we have shown in [17], this example demonstrates the incompleteness of SLDE$^+$-resolution introduced in [23] and based on complete sets of E-unifiers [51]. SLDE$^+$-resolution is an extension of SLD-resolution for logic programs with equality[1]. In this section we shall write closures of the form $C \cdot \varepsilon$ as simply C.

Example 6.1 Consider the following program $\mathcal{P}$:

$$\begin{aligned} & v \cdot v \simeq v \leftarrow P(v) \\ & P(u) \leftarrow u \simeq f(a \cdot a) \\ & query(x, y) \leftarrow f(x \cdot x) \cdot f(y) \simeq f(x \cdot x) \end{aligned} \tag{6.1}$$

Assume that we use lexicographic path ordering [18] induced by the precedence relation $\cdot > f$. Applying the basic left folding to (6.1) we obtain[2]

$$\begin{aligned} & \mathcal{E}_{f(x_1 \cdot x_1)}(x, y) \leftarrow x \simeq f(y \cdot y) \\ & v \cdot v \simeq v \leftarrow P(v) \\ & P(u) \leftarrow u \simeq f(a \cdot a) \\ & query(x, y) \leftarrow z \cdot f(y) \simeq z, \mathcal{E}_{f(x_1 \cdot x_1)}(z, x) \end{aligned} \tag{6.2}$$

For simplicity, below we denote $\mathcal{E}_{f(x_1 \cdot x_1)}$ by E. Applying the basic left folding to (6.2) we obtain

$$E(x, y) \leftarrow x \simeq f(y \cdot y) \tag{6.3}$$

[1] SLDE-resolution and SLDE$^+$-resolution use the idea that in order to solve an equation $s \simeq t$, one tries to find a E-unifier of $s \simeq t$ 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. In Example 6.1, the substitution $\sigma \rightleftharpoons \{x/y\}$ is a complete set of E-unifiers for the equality $f(x \cdot x) \cdot f(y) \simeq f(x \cdot x)$ from the query w.r.t. the idempotence axiom $v \cdot v \simeq v$ from the head of a rule. The choice of this E-unifier results in the lost of a solution. The correct answer substitution $\{a/x, a \cdot a/y\}$ for this program can be obtain by using another E-unifier $\{x \cdot x/y\}$.

[2] In this example, we apply the basic folding to *two* occurrences of the same subterm, which is not allowed by our definition, but may as well be justified by a more sophisticated proof.

$$v \cdot v \simeq v \leftarrow P(v) \tag{6.4}$$
$$P(u) \leftarrow u \simeq x_0, E(x_0, x), x \simeq a$$
$$query(x, y) \leftarrow z \cdot f(y) \simeq z, E(z, x)$$

Basic left superposition of (6.4) into (6.3) yields

$$E(x, y) \leftarrow x \simeq f(y \cdot y)$$
$$E(x, y) \leftarrow P(y), x \simeq f(y)$$
$$v \cdot v \simeq v \leftarrow P(v) \tag{6.5}$$
$$P(u) \leftarrow u \simeq x_0, E(x_0, x), x \simeq a$$
$$query(x, y) \leftarrow z \cdot f(y) \simeq z, E(z, x) \tag{6.6}$$

Basic left superposition of (6.5) into (6.6) yields

$$E(x, y) \leftarrow x \simeq f(y \cdot y) \tag{6.7}$$
$$E(x, y) \leftarrow P(y), x \simeq f(y) \tag{6.8}$$
$$v \cdot v \simeq v \leftarrow P(v)$$
$$P(u) \leftarrow u \simeq x_0, E(x_0, x), x \simeq a \tag{6.9}$$
$$query(x, y) \leftarrow z \cdot f(y) \simeq z, E(z, x) \tag{6.10}$$
$$query(x, y) \leftarrow P(v), v \simeq z, E(z, x) \cdot \{f(y)/v, f(y)/z\}$$

Applying equality solution to (6.7)–(6.10) we obtain

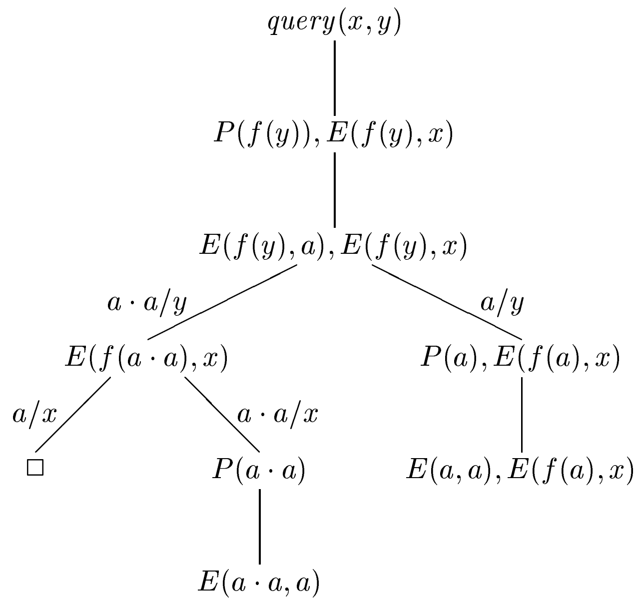
$$E(x, y) \leftarrow x \simeq f(y \cdot y)$$
$$E(x, y) \cdot \{f(y \cdot y)/x\}$$
$$E(x, y) \leftarrow x \simeq f(y), P(y)$$
$$E(x, y) \leftarrow P(y) \cdot \{f(y)/x\}$$
$$v \cdot v \simeq v \leftarrow P(v)$$
$$P(u) \leftarrow E(u, x), x \simeq a$$
$$P(u) \leftarrow E(u, x) \cdot \{a/x\}$$
$$query(x, y) \leftarrow z \cdot f(y) \simeq z, E(z, x)$$
$$query(x, y) \leftarrow v \simeq z, P(v), E(z, x) \cdot \{f(y)/v, f(y)/z\}$$
$$query(x, y) \leftarrow P(v), E(z, x) \cdot \{f(y)/v, f(y)/z\}$$

This derivation is saturated[3]. Now we can remove all closures containing equality, and replace closures by the corresponding clauses:

$$E(f(y \cdot y), y)$$
$$E(f(y), y) \leftarrow P(y)$$
$$P(u) \leftarrow E(u, a)$$
$$query(x, y) \leftarrow P(f(y)), E(f(y), x)$$

[3]Strictly speaking, this derivation is not saturated because one can apply left superposition of $v \cdot v \simeq v \leftarrow P(v)$ into itself. We omit this application because it generates a tautology $v \simeq v \leftarrow \ldots$ (see Section 8).

The (complete) SLD-tree for this program is



i.e. not only the answer is found, but also the tree is finite.[4]

In the following example we shall use a binary function symbol `[...|...]` and a constant `[]` and use the standard Prolog notation for lists:

$[s_1, \ldots, s_n \,|\, t]$ stands for $[s_1 \,|\, \ldots [s_n \,|\, t] \ldots]$

[4]The *modification method* of Brand [9] can be considered as a variant of the basic folding. Brand has shown how to get rid of the equality substitution axioms by pulling nonvariable terms out from argument lists. The modification method transforms the original Horn clause program with equality into "flat" Horn clauses without equality. Then one can use the ordinary SLD-resolution as the inference mechanism for the transformed program. In the logic programming community, this method has been considered in [6, 10, 8, 20]. Brand does not introduce new defining predicates for equations, he puts the equations themselves into bodies of clauses. After his transformation, he treats the equality predicate as the ordinary binary predicate Eq, defined by the fact $Eq(x,x)$. The essential difference between our method and that of Brand is that we try to solve all equations in the bodies by *specialized* applications of equality rules: basic superposition and equality solution. It allows us to automatically "subsume" the axioms of symmetry and transitivity, and also to essentially decrease the number of logical consequences due to the use of orderings. Compare the program obtain by the basic folding method with the program obtained by the Brand's method:

$$
\begin{array}{l}
Eq(x,x) \\
Eq(v,w) \leftarrow Eq(v \cdot v, y), Eq(y,w), P(v) \\
Eq(y,w) \leftarrow Eq(v \cdot v, y), Eq(v,w), P(v) \\
P(u) \leftarrow Eq(f(z),u), Eq(x \cdot x, z), Eq(a,x) \\
query(x,y) \leftarrow Eq(v \cdot z, v), Eq(f(w), v), Eq(x \cdot x, w), Eq(f(y), z)
\end{array}
$$

This logic program is much more complicated and the SLD-tree for this program is infinite.

$[s_1, \ldots, s_n]$ stands for $[s_1 \mid \ldots [s_n \mid []] \ldots]$

Example 6.2 Consider the following program defining the function ap (append) and the predicate rev (reverse) for lists as follows:

$$
\begin{array}{l}
ap([], x) \simeq x \\
ap([x \mid y], z) \simeq [x \mid ap(y, z)] \\
rev([], []) \\
rev([x \mid y], ap(z, [x])) \leftarrow rev(y, z) \\
query(x) \leftarrow rev([a, b], x)
\end{array}
$$

It can be translated into the following set of initial closures:

$$
\begin{array}{ll}
ap([], x) \simeq x & \\
ap([x \mid y], z) \simeq [x \mid ap(y, z)] & \\
rev(x, x) \leftarrow x \simeq [] & \\
rev(u, v) \leftarrow u \simeq [x \mid y], v \simeq ap(z, [x]), rev(y, z) & (6.11) \\
query(x) \leftarrow y \simeq [a, b], rev(y, x) &
\end{array}
$$

For the folding we shall use the new predicate $\mathcal{E}_{ap(x_1, [x_2])}$, for simplicity denoting it by E. Comparison of terms will be made according to the recursive path ordering with precedence $ap > [\mid] > []$. Then in both equations for ap the left hand side is greater than the right hand side. The basic left folding into (6.11) yields

$$
\begin{array}{ll}
ap([], x) \simeq x & (6.12) \\
ap([x \mid y], z) \simeq [x \mid ap(y, z)] & (6.13) \\
rev(x, x) \leftarrow x \simeq [] & \\
E(x, y, z) \leftarrow x \simeq ap(y, [z]) & (6.14) \\
rev(u, v) \leftarrow u \simeq [x \mid y], v \simeq x_0, E(x_0, z, x), rev(y, z) & \\
query(x) \leftarrow y \simeq [a, b], rev(y, x) &
\end{array}
$$

Basic left superpositions of (6.12) and (6.13) into (6.14) yield

$$
\begin{array}{ll}
ap([], x) \simeq x & \\
ap([x \mid y], z) \simeq [x \mid ap(y, z)] & \\
rev(x, x) \leftarrow x \simeq [] & \\
E(x, y, z) \leftarrow x \simeq ap(y, [z]) & \\
E(x, y, z) \leftarrow x \simeq u \cdot \{[]/y, [z]/u\} & \\
E(x, y, z) \leftarrow x \simeq [u \mid ap(v, w)] \cdot \{[u \mid v]/y, [z]/w\} & (6.15) \\
rev(u, v) \leftarrow u \simeq [x \mid y], v \simeq x_0, E(x_0, z, x), rev(y, z) & \\
query(x) \leftarrow y \simeq [a, b], rev(y, x) &
\end{array}
$$

Applying the basic left folding to (6.15) we obtain

$$
\begin{aligned}
& ap(\texttt{[]},x) \simeq x \\
& ap(\texttt{[}x\,|\,y\texttt{]},z) \simeq \texttt{[}x\,|\,ap(y,z)\texttt{]} \\
& rev(x,x) \leftarrow x \simeq \texttt{[]} && (6.16) \\
& E(x,y,z) \leftarrow x \simeq ap(y,\texttt{[}z\texttt{]}) && (6.17) \\
& E(x,y,z) \leftarrow x \simeq u \cdot \{\texttt{[]}/y, \texttt{[}z\texttt{]}/u\} && (6.18) \\
& E(x,y,z) \leftarrow x \simeq \texttt{[}u\,|\,x_1\texttt{]}, E(x_1,v,z) \cdot \{\texttt{[}u\,|\,v\texttt{]}/y\} && (6.19) \\
& rev(u,v) \leftarrow u \simeq \texttt{[}x\,|\,y\texttt{]}, v \simeq x_0, E(x_0,z,x), rev(y,z) && (6.20) \\
& query(x) \leftarrow y \simeq \texttt{[}a,b\texttt{]}, rev(y,x) && (6.21)
\end{aligned}
$$

After applying equality solution to (6.16)–(6.21) we obtain a saturated set of closures. Now we can remove all closures containing equality and replace closures by clauses:

$$
\begin{aligned}
& rev(\texttt{[]},\texttt{[]}) \\
& E(ap(y,\texttt{[}z\texttt{]}),y,z) \\
& E(\texttt{[}z\texttt{]},\texttt{[]},z) && (6.22) \\
& E(\texttt{[}u\,|\,x\texttt{]},\texttt{[}u\,|\,v\texttt{]},z) \leftarrow E(x,v,z) && (6.23) \\
& rev(\texttt{[}x\,|\,y\texttt{]},v) \leftarrow E(v,z,x), rev(y,z) \\
& query(x) \leftarrow rev(\texttt{[}a,b\texttt{]},x)
\end{aligned}
$$

Note that clauses (6.22)–(6.23) look like the usual recursive definition of the *append* predicate[5].

SLD-resolution gives 4 solutions for this program:

$$
\begin{aligned}
& \{ap(\texttt{[}b\texttt{]},\texttt{[}a\texttt{]})/x\} \\
& \{\texttt{[}b,a\texttt{]}/x\} \\
& \{\texttt{[}b\,|\,ap(\texttt{[]},\texttt{[}a\texttt{]})\texttt{]}/x\} \\
& \{ap(ap(\texttt{[]},\texttt{[}b\texttt{]}),\texttt{[}a\texttt{]})/x\}
\end{aligned}
$$

The program generated by Brand's method gives an infinite number of solutions.

Our logical system is highly non-deterministic. There are many ways of defining a concrete strategy for applying the rules. In fact, the strategy of applying the basic folding in our proof of Theorem 5.3 guarantees termination, but may not give interesting programs. A more interesting strategy is to check for repeated occurrences of the same subterm in equalities and to apply the folding to these subterms. Thus, unlike works on folding-unfolding transformations of programs, our strategies of the folding are easily mechanizable. In fact, the subterm $ap(y,\texttt{[}z\texttt{]})$ used for a folding in Example 6.2, has been found by our system completely automatically.

[5] In fact, $E(x,y,z)$ does not denote $x \simeq ap(y,z)$ but rather $x \simeq ap(y,\texttt{[}z\texttt{]})$.

Section 7

Related work

In this paper we try 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 [32]. As a rule, this kind of processes does not terminate, i.e. one cannot obtain a finite canonical program (see e.g. "unfailing completion" [2]). 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 [38, 24]. Our approach is very different from the cited papers because we can always obtain a finite program using generation of conditional and recursive rules.

The basic strategy without ordering restrictions has been introduced by the first author in 1979 (published in Russian, see the survey [11]). The motivation was to reduce the search space by restricting the paramodulation at the term level. The basic superposition inference systems have been introduced in [4, 46].

The technique of moving all work with equalities in the beginning of inferences has been studied in proof theory for sequent systems without free variables [34, 41, 48]. The first completeness proof of resolution with paramodulation [52] constructed the closure of the initial set of clauses under paramodulation and factoring, and then established its unsatisfiability in logic without equality. However, this proof used functional reflexivity axioms.

The main logical system described in our paper is very simple. It consists of three inference rules: basic superposition, equality solution and the basic folding. We introduce new predicate symbols which collect substitutions solving equations in the program, similar to "answer literals" introduced in [27]. The idea of introducing new predicate symbols to simulate basic strategy was used in different forms in [58, 14, 13, 12, 15].

The transformation proposed by Brand [9] achieves similar aims, but his technique requires transitivity and symmetry axioms to be added explicitly or implicitly, which immediately creates branching in the SLD-tree for the resulting programs. We also note that the ideas of using the basic strategy can be traced to Brand [9].

The transformation of programs into the homogeneous form [29, 20] allows one to get rid of the axioms of transitivity and predicate replaceability (i.e. they become "built-in") but the symmetry and the function replaceability axioms remain. Flattening transformation of [10] is analogous to the TE-transformation by Brand and it subsumes all axioms but reflexivity and symmetry. (Symmetry can further be avoided by, in fact, doubling the number of clauses describing equality.) But as we have already demonstrated in the footnote for Example 6.1, the set of clauses obtained by such a transformation is unnecessarily large. Programs obtained as the result of our transformation are more elegant because we partially solve equations using very strong techniques of equality reasoning. This is not achieved by other methods which try to move all equality reasoning to SLD-resolution.

The automatic introduction of new function symbols for defining some useful subterms have been used in automated deduction systems [59, 61].

Section 8

Implementation

Our method is implemented as a program transformation system written in Sicstus Prolog (about 1500 lines of Prolog code). For some more complicated examples a straightforward implementation of our technique can generate many unrelated or excessive closures. In order to make the program transformation faster and more efficient we introduced a few simplification rules:

Basic subsumption allows to remove closures subsumed by other closures. Basic subsumption is described e.g. in [5]. On closures without equality, i.e. clauses used in the resulting program, basic subsumption is equivalent to subsumption (because the resulting closures have no basic positions).

Tautology deletion allows to remove closures $s \simeq t \leftarrow T \cdot \theta$ such that $s\theta = t\theta$ (tautologies).

Double literal elimination consists of removal of the literal B in a closure $H \leftarrow \ldots A \ldots B \ldots \cdot \theta$ such that $A\theta = B\theta$ and A is not an equation.

One more strategy that can be applied but is not yet implemented is the simplification in the basic setting [5].

Currently, our system uses a method for generating initial closures based on the analysis of all complementary pairs of literals [14]. We are going to also include the folklore translation that was used in examples of this paper.

The implementation was powerful enough to transform (completely automatically) both examples of our paper. Despite a different way of generating initial closures, the final programs looked very similar to the ones described in the paper. For instance, the program of Example 6.1 had been transformed into a program having a finite SLD-tree. For the program of Example 6.2, the predicate definition of *append* has been automatically generated. It is interesting that the original folding rule proposed in [14] has been strengthened due to analysis of experimental results.

The complete listing of both examples is given in Appendices A and B.

Appendix A

The idempotence example

This and the following sections give the complete listing of our program transformation system working on Examples 6.1 and 6.2.

The system does not yet have a good interface, so the output is given in a machine-oriented form. For example, it uses variables output by Prolog (like `_141`). The new predicate symbols $\mathcal{E}_A$ ar denoted by `'A'`, for example `'=(X1,f(*(a,a)))'` for $\mathcal{E}_{x1 \simeq f(a \cdot a)}$. Our program generates several sets, the final clauses are in the set called `EI(P)`. These listings are longer than those from the paper because of a different translation into initial closures.

```
% Program set.
_532*_532=_532:-p(_532).
p(f(a*a)).
:-f(_534*_534)*f(_531)=f(_534*_534).

% add to New set.
'=(X1,f(*(a,a)))'(_547):-_547=f(a*a).
% add to EI(P) set.
'p(X1)'(_547):-'=(X1,f(*(a,a)))'(_547).
% add to New set.
_547*_547=_547:-'p(X1)'(_547).
% add to New set.
'=(*(f(*(X1,X1)),f(X2)),f(*(X1,X1)))'(_549,_546):-
  f(_549*_549)*f(_546)=f(_549*_549).
% add to EI(P) set.
:-'=(*(f(*(X1,X1)),f(X2)),f(*(X1,X1)))'(_549,_546).
% EI(P) set.
'p(X1)'(_577):-'=(X1,f(*(a,a)))'(_577).
:-'=(*(f(*(X1,X1)),f(X2)),f(*(X1,X1)))'(_575,_576).
% New set.
'=(X1,f(*(a,a)))'(_617):-_617=f(a*a).
_615*_615=_615:-'p(X1)'(_615).
'=(*(f(*(X1,X1)),f(X2)),f(*(X1,X1)))'(_627,_628):-
  f(_627*_627)*f(_628)=f(_627*_627).
% add to EI(P) set.
'=(X1,f(*(a,a)))'(f(a*a)).
% add to Used set.
'=(X1,f(*(a,a)))'(_682):-_682=f(a*a).
% add to Used set.
_680*_680=_680:-'p(X1)'(_680).
```

```
% add to New set.
'=(X1,f(*(a,a)))'(_1070):-
  _1070=f(_680),_680\==a,'p(X1)'(_680).
% add to Used set.
'=(*(f(*(X1,X1)),f(X2)),f(*(X1,X1)))'(_692,_693):-
  f(_692*_692)*f(_693)=f(_692*_692).
% add to New set.
'=(*(f(*(X1,X1)),f(X2)),f(*(X1,X1)))'(_692,_693):-
  _1114=f(_692*_692),_1114\==f(_693),_1114\==f(_692*_692),'p(X1)'(_1114).
% add to New set.
'=(*(f(*(X1,X1)),f(X2)),f(*(X1,X1)))'(_692,_693):-
  f(_692)*f(_693)=f(_692*_692),'p(X1)'(_692).
% add to New set.
'=(*(f(*(X1,X1)),f(X2)),f(*(X1,X1)))'(_692,_693):-
  f(_692)=f(_692*_692)*f(_693),'p(X1)'(_692).
% add to EI(P) set.
'=(X1,f(*(a,a)))'(f(a)):-'p(X1)'(a).
% add to Used set.
'=(X1,f(*(a,a)))'(_690):-_690=f(_682),_682\==a,'p(X1)'(_682).
% add to EI(P) set.
'=(*(f(*(X1,X1)),f(X2)),f(*(X1,X1)))'(_706,_706*_706):-'p(X1)'(f(_706*_706)).
% add to Used set.
'=(*(f(*(X1,X1)),f(X2)),f(*(X1,X1)))'(_706,_707):-
  _700=f(_706*_706),_700\==f(_707),_700\==f(_706*_706),'p(X1)'(_700).
% add to New set.
'=(*(f(*(X1,X1)),f(X2)),f(*(X1,X1)))'(_706,_707):-
  _700=f(_706),_700\==f(_707),_700\==f(_706*_706),'p(X1)'(_700),'p(X1)'(_706).
% add to Used set.
'=(*(f(*(X1,X1)),f(X2)),f(*(X1,X1)))'(_694,_695):-
  f(_694)*f(_695)=f(_694*_694),'p(X1)'(_694).
% add to New set.
'=(*(f(*(X1,X1)),f(X2)),f(*(X1,X1)))'(_694,_695):-
  _1104=f(_694*_694),'p(X1)'(_694),_1104\==f(_695),_1104\==f(_694),'p(X1)'(_1104).
% add to Used set.
'=(*(f(*(X1,X1)),f(X2)),f(*(X1,X1)))'(_694,_695):-
  f(_694)=f(_694*_694)*f(_695),'p(X1)'(_694).
% add to Used set.
'=(*(f(*(X1,X1)),f(X2)),f(*(X1,X1)))'(_708,_709):-
  _702=f(_708),_702\==f(_709),_702\==f(_708*_708),'p(X1)'(_702),'p(X1)'(_708).
% add to Used set.
'=(*(f(*(X1,X1)),f(X2)),f(*(X1,X1)))'(_708,_709):-
  _702=f(_708*_708),'p(X1)'(_708),_702\==f(_709),_702\==f(_708),'p(X1)'(_702).

% EI(P) set.
:-'=(*(f(*(X1,X1)),f(X2)),f(*(X1,X1)))'(_125,_126).
'=(X1,f(*(a,a)))'(f(a*a)).
'p(X1)'(_127):-'=(X1,f(*(a,a)))'(_127).
'=(X1,f(*(a,a)))'(f(a)):-'p(X1)'(a).
'=(*(f(*(X1,X1)),f(X2)),f(*(X1,X1)))'(_132,_132*_132):-'p(X1)'(f(_132*_132)).
```

Appendix B

The append-reverse program

```
% Program set.
ap(nil,_530)=_530.
ap(cons(_532,_533),_536)=cons(_532,ap(_533,_536)).
rev(nil,nil).
rev(cons(_535,_536),ap(_532,cons(_535,nil))):-rev(_536,_532).
:-rev(cons(a,cons(b,nil)),_533).

% add to New set.
ap(nil,_545)=_545.
% add to New set.
ap(cons(_547,_548),_551)=cons(_547,ap(_548,_551)).
% add to New set.
'=(X1,nil)'(_547):-_547=nil.
% add to EI(P) set.
'rev(X1,X2)'(_551,_547):-'=(X1,nil)'(_551),'=(X1,nil)'(_547).
% add to New set.
'=(X1,ap(X2,cons(X3,nil)))'(_547,_1689,_1692):-
  _547=ap(_1689,cons(_1692,nil)).
% add to New set.
'=(X1,cons(X2,X3))'(_551,_1692,_1693):-
  _551=cons(_1692,_1693).
% add to EI(P) set.
'rev(X1,X2)'(_551,_547):-
  '=(X1,cons(X2,X3))'(_551,_1692,_1693),
  '=(X1,ap(X2,cons(X3,nil)))'(_547,_1689,_1692),'rev(X1,X2)'(_1693,_1689).
% add to New set.
'=(cons(a,cons(b,nil)),nil)':-
  cons(a,cons(b,nil))=nil.
% add to EI(P) set.
'rev(cons(a,cons(b,nil)),X1)'(_548):-
  '=(cons(a,cons(b,nil)),nil)','=(X1,nil)'(_548).
% add to New set.
'=(cons(a,cons(b,nil)),cons(X1,X2))'(_3511,_3512):-
  cons(a,cons(b,nil))=cons(_3511,_3512).
% add to EI(P) set.
'rev(cons(a,cons(b,nil)),X1)'(_548):-
  '=(cons(a,cons(b,nil)),cons(X1,X2))'(_3511,_3512),
  '=(X1,ap(X2,cons(X3,nil)))'(_548,_3508,_3511),'rev(X1,X2)'(_3512,_3508).
```

```
% add to EI(P) set.
:-'rev(cons(a,cons(b,nil)),X1)'(_548).
% EI(P) set.
'rev(X1,X2)'(_582,_583):-'=(X1,nil)'(_582),'=(X1,nil)'(_583).
'rev(X1,X2)'(_592,_593):-
  '=(X1,cons(X2,X3))'(_592,_586,_587),
  '=(X1,ap(X2,cons(X3,nil)))'(_593,_579,_586),'rev(X1,X2)'(_587,_579).
'rev(cons(a,cons(b,nil)),X1)'(_580):-
  '=(cons(a,cons(b,nil)),nil)','=(X1,nil)'(_580).
'rev(cons(a,cons(b,nil)),X1)'(_591):-
  '=(cons(a,cons(b,nil)),cons(X1,X2))'(_585,_586),
  '=(X1,ap(X2,cons(X3,nil)))'(_591,_579,_585),'rev(X1,X2)'(_586,_579).
:-'rev(cons(a,cons(b,nil)),X1)'(_575).
% New set.
ap(nil,_613)=_613.
ap(cons(_615,_616),_619)=cons(_615,ap(_616,_619)).
'=(X1,nil)'(_612):-_612=nil.
'=(X1,ap(X2,cons(X3,nil)))'(_618,_619,_620):-_618=ap(_619,cons(_620,nil)).
'=(X1,cons(X2,X3))'(_615,_616,_617):-_615=cons(_616,_617).
'=(cons(a,cons(b,nil)),nil)':-cons(a,cons(b,nil))=nil.
'=(cons(a,cons(b,nil)),cons(X1,X2))'(_621,_622):-
  cons(a,cons(b,nil))=cons(_621,_622).
% add to Used set.
ap(nil,_678)=_678.
% add to Used set.
ap(cons(_680,_681),_684)=cons(_680,ap(_681,_684)).
% add to EI(P) set.
'=(X1,nil)'(nil).
% add to Used set.
'=(X1,nil)'(_677):-_677=nil.
% add to EI(P) set.
'=(X1,ap(X2,cons(X3,nil)))'(ap(_684,cons(_685,nil)),_684,_685).
% add to Used set.
'=(X1,ap(X2,cons(X3,nil)))'(_683,_684,_685):-_683=ap(_684,cons(_685,nil)).
% add to New set.
'=(X1,ap(X2,cons(X3,nil)))'(_683,nil,_685):-
  _1071=_683,_1071\==cons(_685,nil).
% add to New set.
'=(X1,ap(X2,cons(X3,nil)))'(_683,cons(_1073,_1074),_685):-
  _683=cons(_1073,ap(_1074,_1077)),_1077\==cons(_685,nil).
% add to EI(P) set.
'=(X1,cons(X2,X3))'(cons(_681,_682),_681,_682).
% add to Used set.
'=(X1,cons(X2,X3))'(_680,_681,_682):-_680=cons(_681,_682).
% add to Used set.
'=(cons(a,cons(b,nil)),nil)':-cons(a,cons(b,nil))=nil.
% add to EI(P) set.
'=(cons(a,cons(b,nil)),cons(X1,X2))'(a,cons(b,nil)).
% add to Used set.
'=(cons(a,cons(b,nil)),cons(X1,X2))'(_686,_687):-
  cons(a,cons(b,nil))=cons(_686,_687).
% add to EI(P) set.
'=(X1,ap(X2,cons(X3,nil)))'(cons(_688,nil),nil,_688).
```

```
% add to Used set.
'=(X1,ap(X2,cons(X3,nil)))'(_686,nil,_688):-_680=_686,_680\==cons(_688,nil).
% add to EI(P) set.
'=(X1,ap(X2,cons(X3,nil)))'(cons(_692,ap(_693,cons(_697,nil))),cons(_692,_693),_697).
% add to Used set.
'=(X1,ap(X2,cons(X3,nil)))'(_695,cons(_692,_693),_697):-
  _695=cons(_692,ap(_693,_681)),_681\==cons(_697,nil).
% add to New set.
'=(X1,ap(X2,cons(X3,nil)))'(_695,cons(_692,nil),_697):-
  _695=cons(_692,_681),_681\==cons(_697,nil).
% add to New set.
'=(X1,ap(X2,cons(X3,nil)))'(_695,cons(_692,cons(_1113,_1114)),_697):-
  _695=_2255,cons(_697,nil)\==cons(_697,nil),
  '=(X1,ap(X2,cons(X3,nil)))'(_2641,_1114,_697),
  '=(X1,cons(X2,X3))'(_2427,_1113,_2641),'=(X1,cons(X2,X3))'(_2255,_692,_2427).
% add to EI(P) set.
'=(X1,ap(X2,cons(X3,nil)))'(cons(_689,cons(_694,nil)),cons(_689,nil),_694).
% add to Used set.
'=(X1,ap(X2,cons(X3,nil)))'(_692,cons(_689,nil),_694):-
  _692=cons(_689,_681),_681\==cons(_694,nil).
% add to EI(P) set.
'=(X1,ap(X2,cons(X3,nil)))'(_702,cons(_710,cons(_707,_708)),_715):-
  '=(X1,ap(X2,cons(X3,nil)))'(_685,_708,_715),
  '=(X1,cons(X2,X3))'(_678,_707,_685),'=(X1,cons(X2,X3))'(_702,_710,_678).
% add to Used set.
'=(X1,ap(X2,cons(X3,nil)))'(_713,cons(_710,cons(_707,_708)),_715):-
  _713=_702,cons(_715,nil)\==cons(_715,nil),
  '=(X1,ap(X2,cons(X3,nil)))'(_685,_708,_715),'=(X1,cons(X2,X3))'(_678,_707,_685),
  '=(X1,cons(X2,X3))'(_702,_710,_678).

% EI(P) set.
:-'rev(cons(a,cons(b,nil)),X1)'(_125).
'=(X1,nil)'(nil).
'=(X1,ap(X2,cons(X3,nil)))'(ap(_132,cons(_133,nil)),_132,_133).
'=(X1,cons(X2,X3))'(cons(_129,_130),_129,_130).
'=(cons(a,cons(b,nil)),cons(X1,X2))'(a,cons(b,nil)).
'=(X1,ap(X2,cons(X3,nil)))'(cons(_130,nil),nil,_130).
'=(X1,ap(X2,cons(X3,nil)))'(cons(_134,ap(_131,cons(_139,nil))),cons(_134,_131),_139).
'=(X1,ap(X2,cons(X3,nil)))'(cons(_131,cons(_136,nil)),cons(_131,nil),_136).
'rev(X1,X2)'(_132,_133):-'=(X1,nil)'(_132),'=(X1,nil)'(_133).
'rev(X1,X2)'(_142,_143):-
  '=(X1,cons(X2,X3))'(_142,_136,_137),
  '=(X1,ap(X2,cons(X3,nil)))'(_143,_129,_136),'rev(X1,X2)'(_137,_129).
'rev(cons(a,cons(b,nil)),X1)'(_130):-
  '=(cons(a,cons(b,nil)),nil)','=(X1,nil)'(_130).
'rev(cons(a,cons(b,nil)),X1)'(_141):-
  '=(cons(a,cons(b,nil)),cons(X1,X2))'(_135,_136),
  '=(X1,ap(X2,cons(X3,nil)))'(_141,_129,_135),'rev(X1,X2)'(_136,_129).
'=(X1,ap(X2,cons(X3,nil)))'(_149,cons(_146,cons(_143,_144)),_151):-
  '=(X1,ap(X2,cons(X3,nil)))'(_136,_144,_151),
  '=(X1,cons(X2,X3))'(_129,_143,_136),'=(X1,cons(X2,X3))'(_149,_146,_129).
```

Appendix C

The proof of completeness

In this section we prove Theorem 5.2 on completeness of the basic folding.

We start with proving some properties of inferences by basic superposition, equality solution and resolution defined below:

Definition C.1 (resolution)
Let $\mathcal{C}_1, \mathcal{C}_2$ be closures

$$A \leftarrow T_1 \cdot \theta_1$$

and

$$A' \leftarrow T_2, B, T_3 \cdot \theta_2$$

with disjoint variables such that A is not an equation. A *resolution* of $\mathcal{C}_1$ and $\mathcal{C}_2$ is the closure

$$A' \leftarrow T_2, T_1, T_3 \cdot (\theta_1 \cup \theta_2)\sigma$$

where σ is a most general unifier of $A\theta_1$ and $B\theta_2$.

By **BR** we denote the inference system consisting of resolution, basic superposition and equality solution. Derivability in **BR** will be denoted by $\vdash_{\mathbf{BR}}$. Let S be a set of closures. A **BR**-*answer for* S is any tuple of terms $\langle t_1\theta, \ldots, t_n\theta \rangle$ such that $S \vdash_{\mathbf{BR}} query(t_1, \ldots, t_n) \cdot \theta$.

Derivations in **BR** are assumed to be in the tree-like form.

The following lemma states soundness of **BR**:

Lemma C.2 Let S be a set of closures. Then any **BR**-answer for S is a correct answer for S.

Proof. Immediate from the fact that all rules of **BR** are admissible in first-order logic, when we replace closures $C \cdot \theta$ by clauses $C\theta$. □

For the technical purposes, we also introduce *pseudo* **BR**-*derivations.*

Definition C.3 (pseudo BR-derivation)
A *pseudo* **BR**-*derivation* is defined similar to **BR**-derivations, by dropping the restrictions that the literal $u[s'] \simeq v$ in Definition 3.6 of basic left superposition and the literal $s \simeq t$ in Definition 3.7 of equality solution be the first equality literals in the bodies.

Note that any **BR**-derivation is also a pseudo **BR**-derivation.

Definition C.4 (ground version of a derivation)
Let Π and Π' be (pseudo) **BR**-derivations, $\mathcal{C}_1, \ldots, \mathcal{C}_n$ and $\mathcal{C}'_1, \ldots, \mathcal{C}'_n$ be the sets of closures, occurring in Π and Π', respectively. Then Π' is called the *ground version of* Π iff there is a one-to-one mapping f from $\{1, \ldots, n\}$ to $\{1, \ldots, n\}$ such that

1. If $\mathcal{C}_i$ is obtained by an inference rule from $\mathcal{C}_{j_1}, \ldots, \mathcal{C}_{j_k}$, then $\mathcal{C}'_{f(i)}$ is obtained by the same application[1] of an inference rule from $\mathcal{C}'_{f(j_1)}, \ldots, \mathcal{C}'_{f(j_k)}$.
2. Let the closure $\mathcal{C}_i$ have the form $C \cdot \sigma$. Then there is a substitution θ such that $\mathcal{C}'_{f(i)}$ has the form $C\sigma\theta \cdot \varepsilon$ and $C\sigma\theta$ is ground.

A **BR**-derivation is *groundable* iff it has a ground version.

A **BR**-derivation is groundable iff this derivation corresponds to the *constraint strategy* $\mathcal{CS}_{\mathcal{OB}}$ of [47] in which all equality and ordering constraints (similar to our ordering conditions in Definition 3.6) are inherited. In the proof of Theorem 5.2 given below we use the main result ([47], Theorem 6.14) on the completeness of the $\mathcal{CS}_{\mathcal{OB}}$-strategy which can be formulated as follows. Extend temporarily the set of clauses with clauses with empty heads, thus allowing closures of the form $\leftarrow A_1, \ldots, A_n \cdot \theta$.

Theorem C.1 Assume that a set of ε-closures S is inconsistent, (which means that the set of clauses $\{C \mid (C \cdot \varepsilon) \in S\}$ in inconsistent). Then there exists a groundable pseudo **BR**-derivation of $\leftarrow \cdot\varepsilon$ from S.

When writing this paper, we had to choose between the use of constraints in the style of [47] or substitutions in the style of [3]. The use of ordering constraints would simplify many proofs below based on transformations of derivations. But the definition of the basic folding which is already enough sophisticated would have become even more difficult to understand.

As the following example shows, not every **BR**-derivation is groundable.

Example C.5 Consider the following **BR**-derivation:

$$\dfrac{\dfrac{\dfrac{P(x,y) \leftarrow h(x) \simeq g(y) \cdot \{g(y_1)/x, h(x_1)/y\} \quad u \simeq g(v) \leftarrow \cdot\{h(u_1)/u\}}{P(x,y) \leftarrow g(v) \simeq g(y) \cdot \{g(y_1)/x, h(x_1)/y\}}\ (\mathcal{S})}{P(x,y) \leftarrow \cdot\{g(y_1)/x, h(x_1)/y\}}\ (\mathcal{ES}) \quad Q(z) \leftarrow P(g(z), h(z)) \cdot \varepsilon}{Q(z) \leftarrow \cdot\varepsilon}\ (\mathcal{R})$$

This is a correct **BR**-derivation for every reduction ordering $\succ$.

Every ground version of this derivation must have the following form, for a suitable ground term t.

$$\dfrac{\dfrac{\dfrac{P(g(t),h(t)) \leftarrow h(g(t)) \simeq g(h(t)) \cdot \varepsilon \quad h(g(t)) \simeq g(h(t)) \leftarrow \cdot\varepsilon}{P(g(t),h(t)) \leftarrow g(h(t)) \simeq g(h(t)) \cdot \varepsilon}\ (\mathcal{S})}{P(g(t),h(t)) \leftarrow \cdot\varepsilon}\ (\mathcal{ES}) \quad Q(t) \leftarrow P(g(t),h(t)) \cdot \varepsilon}{Q(t) \leftarrow \cdot\varepsilon}\ (\mathcal{R})$$

[1]By *the same application* we mean that the positions of terms involved in the application of the rule coincide. One can give a precise definition, but it would take too much space.

This is a **BR**-derivation iff $h(g(t)) \succ g(h(t))$. If we define $\succ$ to be the lexicographic path ordering with $g > h$, this derivation is not a **BR**-derivation because of the violation of the ordering conditions. Hence, the original derivation is not groundable w.r.t. this reduction ordering. By modifying this example, we can construct a derivation which is a correct **BR**-derivation for *any* reduction ordering and which has no ground version under any reduction ordering.

We use groundable derivations to swap applications of inference rules in the proof of the following lemma.

Lemma C.6 Let Π be a groundable pseudo **BR**-derivation of a closure $C \cdot \sigma$ from a set of closures S without applications of resolution and C does not contain equality. Then there is a groundable **BR**-derivation Π' of $C \cdot \sigma$ from S with the same number of applications of superposition.

Proof. Let $\boldsymbol{\Pi}$ be a ground version of Π.

We call *invalid* any application of basic left superposition that violates the restriction that $u[s'] \simeq v$ be the leftmost equation in the body or any application of equality solution that violates the restriction that $s \simeq t$ be the leftmost equation in the body.

We use induction on the ordinal $i\omega + j$ where i is the number of invalid applications of inference rules in Π, and j is the minimal distance between invalid applications of inference rules and the root of Π (by the distance we understand the number of applications of inference rules in between).

Obviously $j \geq 2$. If $j = 2$ then the last two applications of inference rules in Π remove two equations from the body of a closure, thus these two inference rules are both equality solutions. It is obvious that after the permutation of these two applications of equality solution the parameter i will decrease, and the end sequent of the derivation will remain the same. It is also easy to see that after the permutation of the corresponding equality solutions in $\boldsymbol{\Pi}$ we obtain the ground version of the derivation Π'.

Assume $j > 2$. Consider an application of an invalid inference rule with the minimal distance from the root. We only consider the case when it is superposition and the rule application immediately below is also a superposition (all other cases are similar). In the picture below, we display this fragment of Π and the corresponding fragment of $\boldsymbol{\Pi}$:

$$
\dfrac{\dfrac{\begin{matrix}\Pi_1 & \Pi_2\\ \mathcal{D}_1 & \mathcal{D}_2\end{matrix}}{\mathcal{D}_4}\,(\mathcal{S}) \qquad \begin{matrix}\Pi_3\\ \mathcal{D}_3\end{matrix}}{\mathcal{D}_5}\,(\mathcal{S})
\qquad\qquad
\dfrac{\dfrac{\begin{matrix}\boldsymbol{\Pi}_1 & \boldsymbol{\Pi}_2\\ \boldsymbol{\mathcal{D}}_1 & \boldsymbol{\mathcal{D}}_2\end{matrix}}{\boldsymbol{\mathcal{D}}_4}\,(\boldsymbol{\mathcal{S}}) \qquad \begin{matrix}\boldsymbol{\Pi}_3\\ \boldsymbol{\mathcal{D}}_3\end{matrix}}{\boldsymbol{\mathcal{D}}_5}\,(\boldsymbol{\mathcal{S}})
$$

where the closure $\mathcal{D}_4$ is obtained by an invalid superposition from $\mathcal{D}_2$ into $\mathcal{D}_1$, and $\mathcal{D}_5$ is obtained by a (valid) superposition between $\mathcal{D}_3$ and $\mathcal{D}_4$. Consider one of the possible 8 cases, where the superposition from $\mathcal{D}_2$ into $\mathcal{D}_1$ is left superposition and $\mathcal{D}_5$ is obtained by basic right superposition from $\mathcal{D}_4$ into $\mathcal{D}_3$. Then

$$
\begin{aligned}
\mathcal{D}_1 &= (s \simeq t \leftarrow T_1, u_1 \simeq v_1, T_2, u_2[s_2] \simeq v_2, T_3 \cdot \theta_1)\\
\mathcal{D}_2 &= (p \simeq q \leftarrow T_4 \cdot \theta_2)\\
\mathcal{D}_3 &= (u[s'] \simeq v \leftarrow T_5 \cdot \theta_3)\\
\mathcal{D}_4 &= (s \simeq t \leftarrow T_1, u_1 \simeq v_1, T_2, T_4, u_2[q] \simeq v_2, T_3 \cdot \theta_4)\\
\mathcal{D}_5 &= (u[t] \simeq v \leftarrow T_1, u_1 \simeq v_1, T_2, T_4, u_2[q] \simeq v_2, T_3, T_5 \cdot \theta_5)
\end{aligned}
$$

Here $\theta_4 = (\theta_1 \cup \theta_2)\sigma$, where σ is a most general unifier of $p\theta_2$ and $s_2\theta_1$; and $\theta_5 = (\theta_3 \cup \theta_4)\delta$, where δ is a most general unifier of $s\theta_4$ and $s'\theta_3$.

This derivation tree can be transformed to the following (on the right hand side of the picture there is a corresponding ground version):

$$\dfrac{\dfrac{\begin{matrix}\Pi_1 & \Pi_3\\ \mathcal{D}_1 & \mathcal{D}_3\end{matrix}}{\mathcal{D}'_4}\,(\mathcal{S}) \quad \begin{matrix}\Pi_2\\ \mathcal{D}_2\end{matrix}}{\mathcal{D}'_5}\,(\mathcal{S}) \qquad\qquad \dfrac{\dfrac{\begin{matrix}\boldsymbol{\Pi}_1 & \boldsymbol{\Pi}_3\\ \boldsymbol{\mathcal{D}}_1 & \boldsymbol{\mathcal{D}}_3\end{matrix}}{\boldsymbol{\mathcal{D}}'_4}\,(\boldsymbol{\mathcal{S}}) \quad \begin{matrix}\boldsymbol{\Pi}_2\\ \boldsymbol{\mathcal{D}}_2\end{matrix}}{\boldsymbol{\mathcal{D}}'_5}\,(\boldsymbol{\mathcal{S}})$$

where

$$\mathcal{D}'_4 = (u[t] \simeq v \leftarrow T_1, u_1 \simeq v_1, T_2, u_2[s_2] \simeq v_2, T_3, T_5 \cdot \theta'_4)$$
$$\mathcal{D}'_5 = (u[t] \simeq v \leftarrow T_1, u_1 \simeq v_1, T_2, T_4, u_2[q] \simeq v_2, T_3, T_5 \cdot \theta'_5)$$

Here $\theta'_4 = (\theta_1 \cup \theta_3)\delta'$, where δ' is a most general unifier of $s\theta_1$ and $s'\theta_3$; and $\theta'_5 = (\theta_2 \cup \theta'_4)\sigma'$, where σ' is a most general unifier of $s_2\theta'_4$ and $p\theta_2$.

By using standard arguments of unification theory [1], one can prove that $\mathcal{D}'_5$ is a variant of $\mathcal{D}_5$. In the same way, the ground version $\boldsymbol{\Pi}$ of Π is transformed into a ground version $\boldsymbol{\Pi}'$ of Π'. Let $\eta_1, \ldots, \eta_5$ be substitutions such that whenever $\mathcal{D}_i$ has the form $D_i \cdot \theta_i$ then $\boldsymbol{\mathcal{D}}_i$ has the form $D_i\theta_i\eta_i \cdot \varepsilon$ (and similar for η'_4, η'_5). Then

$$\boldsymbol{\mathcal{D}_4}' = (u[t] \simeq v \leftarrow T_1, u_1 \simeq v_1, T_2, u_2[s_2] \simeq v_2, T_3, T_5)(\theta_1\eta_1 \cup \theta_3\eta_3) \cdot \varepsilon$$
$$\boldsymbol{\mathcal{D}_5}' = (u[t] \simeq v \leftarrow T_1, u_1 \simeq v_1, T_2, T_4, u_2[q] \simeq v_2, T_3, T_5)(\theta_1\eta_1 \cup \theta_2\eta_2 \cup \theta_3\eta_3) \cdot \varepsilon$$

One can note that the transformed derivation has the same number of invalid rule applications as Π, but the minimal distance between the root and an invalid rule application has decreased. We have to also check that the ordering conditions on the applications of superposition are preserved in the transformed derivation, i.e.

$$v_2\theta'_4\sigma' \not\succeq u_2[s_2]\theta'_4\sigma'$$
$$q\theta_2\sigma' \not\succeq p\theta_2\sigma'$$

Note that the ground equation $u_2[s_2]\theta_1\eta_1 \simeq v_2\theta_1\eta_1$ is an instance of the equation $u_2[s_2]\theta'_4\sigma' \simeq v_2\theta'_4\sigma'$. It is easy to see that the ordering conditions are preserved in the derivation $\boldsymbol{\Pi}'$. Thus, we have $u_2[s_2]\theta_1\eta_1 \succ v_2\theta_1\eta_1$. By the definition of reduction orderings, we have $v_2\theta'_4\sigma' \not\succeq u_2[s_2]\theta'_4\sigma'$. The condition $q\theta_2\sigma' \not\succeq p\theta_2\sigma'$ is similar.

It is easy to see that the number of applications of superposition in the transformed derivation does not change.

The other cases are similar. □

The difference between **BR**-derivations and pseudo **BR**-derivations can be formalized using the selection mechanism of [3, 5]. Unfortunately, the results of [3, 5] on the completeness of basic superposition with selection are not immediately applicable to our proofs. Indeed, Lemma C.6 proves admissibility of selection w.r.t. derivations of any closure but not derivations of the empty closure as in [3, 5].

Definition C.7 (regular derivation)
A **BR**-derivation is *regular* iff all applications of superposition and equality solution precede all applications of resolution.[2]

[2] In the tree-like **BR**-derivation this means that below an application of resolution there may be only applications of resolution.

Lemma C.8 For every (groundable) **BR**-derivation Π there exists a regular (groundable) **BR**-derivation Π' with the same root closure and with the same number of applications of superposition.

Proof. Let us call a disorder any pair of nodes N_1, N_2 of the derivation tree such that N_1 is below N_2, N_1 is an application of an equality solution or a superposition, and N_2 is an application of resolution. The *degree of disorder* of the derivation Π is the number of disorders in Π. The regular derivations are exactly the derivations with the degree of disorder 0.

The proof is by induction on the degree of disorder of Π. We transform any non-regular derivation Π to a derivation with the smaller degree of disorder.

Let Π be a non-regular **BR**-derivation. Then there is a disorder formed by consequent nodes:

$$\cfrac{\cfrac{\begin{array}{cc}\Pi_1 & \Pi_2\\ \mathcal{D}_1 & \mathcal{D}_2\end{array}}{\mathcal{D}_4}\,(\mathcal{R}) \qquad \begin{array}{c}\Pi_3\\ \mathcal{D}_3\end{array}}{\mathcal{D}_5}\,(\mathcal{S})$$

where $\mathcal{S}$ denotes a superposition rule, and $\mathcal{R}$ denotes a resolution rule (the case of equality solution and resolution is similar and will not be considered). There are 4 possible different instantiations of this picture, depending of the position of equations to which superposition is applied. We only consider one such instantiation here, where

$$\begin{array}{l}
\mathcal{D}_1 = (A_1 \leftarrow T_1, u[s'] \simeq v, T_2 \cdot \theta_1)\\
\mathcal{D}_2 = (A_2 \leftarrow T_3, B, T_4 \cdot \theta_2)\\
\mathcal{D}_3 = (s \simeq t \leftarrow T \cdot \theta_3)\\
\mathcal{D}_4 = (A_2 \leftarrow T_3, T_1, u[s'] \simeq v, T_2, T_4 \cdot \theta_4)\\
\mathcal{D}_5 = (A_2 \leftarrow T_3, T_1, T, u[t] \simeq v, T_2, T_4 \cdot \theta_5)
\end{array}$$

Here $\theta_4 = (\theta_1 \cup \theta_2)\sigma$, where σ is a most general unifier of $A_1\theta_1$ and $B\theta_2$; and $\theta_5 = (\theta_3 \cup \theta_4)\delta$, where δ is a most general unifier of $s\theta_3$ and $s'\theta_4$.

This derivation tree can be transformed to

$$\cfrac{\cfrac{\begin{array}{cc}\Pi_1 & \Pi_3\\ \mathcal{D}_1 & \mathcal{D}_3\end{array}}{\mathcal{D}_4'}\,(\mathcal{S}) \qquad \begin{array}{c}\Pi_2\\ \mathcal{D}_2\end{array}}{\mathcal{D}_5'}\,(\mathcal{R})$$

where

$$\begin{array}{l}
\mathcal{D}_4' = (A_1 \leftarrow T_1, T, u[t] \simeq v, T_2 \cdot \theta_4')\\
\mathcal{D}_5' = (A_2 \leftarrow T_3, T_1, T, u[t] \simeq v, T_2, T_4 \cdot \theta_5')
\end{array}$$

Here $\theta_4' = (\theta_1 \cup \theta_3)\delta'$, where δ' is a most general unifier of $s'\theta_1$ and $s\theta_3$; and $\theta_5' = (\theta_2 \cup \theta_4')\sigma'$, where σ' is a most general unifier of $A_1\theta_4'$ and $B\theta_2$.

By using standard arguments of unification theory [1], one can prove that $\mathcal{D}_5'$ is a variant of $\mathcal{D}_5$. In order to complete the proof we note that the new derivation tree has a strictly smaller degree of disorder.

The proof that resolution is permutable with equality solution is similar. It is obvious that the number of applications of superposition in the transformed derivation does not change. It is also easy to check that if Π is groundable then the transformed derivation Π' is groundable. □

Lemma C.9 Let S be a set of closures of the form $C \cdot \varepsilon$. If there is a correct answer for S then there is a **BR**-answer for S obtained by a groundable **BR**-derivation.

Proof. Note that if we also consider closures with the empty head in **BR**-derivations, Lemmas C.6 and C.8 remain true. From Theorem C.1 it follows that there is a groundable pseudo **BR**-derivation of the empty closure $\leftarrow \cdot\varepsilon$ from $S \cup \{\leftarrow query(x_1, \ldots, x_n) \cdot \varepsilon\}$. By Lemmas C.6 and C.8 there is a groundable regular **BR**-derivation of $\leftarrow \cdot\varepsilon$ from $S \cup \{\leftarrow query(x_1, \ldots, x_n) \cdot \varepsilon\}$. Note that there is no superposition or equality solution applicable to the closure $\leftarrow query(x_1, \ldots, x_n) \cdot \varepsilon$.

It means that there is a set of closures S' derivable in **BR** from S with no applications of resolution such that the empty closure $\leftarrow \cdot\varepsilon$ is derivable from $S' \cup \{\leftarrow query(x_1, \ldots, x_n) \cdot \varepsilon\}$ be resolutions only.

By standard arguments of the theory of logic programming [1] one can prove that there are terms $t_1, \ldots, t_n$ such that a closure of the form $query(t_1, \ldots, t_n) \cdot \sigma$ is derivable from S' by resolutions only. Thus, $\langle t_1, \ldots, t_n \rangle$ is a correct **BR**-answer for S' and hence for S. □

Lemma C.10 Let S_0 be an set of initial closures. Let $\Delta = (S_0 \Rightarrow S_1 \Rightarrow \ldots)$ be a **BF**-derivation, S_ω be the limit of this derivation. If there is a correct answer for S_0 then there is a **BR**-answer for S_ω obtained by a groundable **BR**-derivation.

Proof. Note that for any application of the basic folding

$$\frac{S}{S \setminus \{\mathcal{C}\} \cup \{\mathcal{C}', Def_{t'}\}} \ \mathcal{BF}$$

if $\mathcal{C}$ is an ε-closure then both $\mathcal{C}'$ and $Def_{t'}$ are ε-closures. Consider a sequence of sets of closures S'_i defined as follows.

1. $S'_0 \rightleftharpoons S_0$;

2. If S_{i+1} is obtained from S_i by an application of the basic folding

$$\frac{S}{S \setminus \{\mathcal{C}\} \cup \{\mathcal{C}', Def_{t'}\}} \ \mathcal{BF}$$

such that $\mathcal{C} \in S'_i$, then $S'_{i+1} \rightleftharpoons S'_i \setminus \{\mathcal{C}\} \cup \{\mathcal{C}', Def_{t'}\}$. Otherwise let $S'_{i+1} \rightleftharpoons S'_i$.

Obviously, $S'_i \subseteq S_i$, for all i. Consider the sequence $S'_0 \Rightarrow S'_1 \Rightarrow \ldots$. By removing sets S'_{i+1} equal to S'_i from this sequence we obtain a **BF**-derivation Δ'. Let S'_ω be a limit of Δ'. Obviously, $S'_\omega \subseteq S_\omega$. By Lemma 5.2, S'_ω has a correct answer. Since all closures in S'_ω satisfy the conditions of Lemma C.9, S'_ω has a **BR**-answer obtained by a groundable **BR**-derivation. Thus, S_ω has a **BR**-answer obtained by a groundable **BR**-derivation. □

Lemma C.11 Let S_0 be a set of initial closures. Let $\Delta = (S_0 \Rightarrow S_1 \Rightarrow \ldots)$ be a **BF**-derivation, S_ω be the limit of this derivation. If there is a correct answer for S_0 then there is a **BR**-answer for S_ω obtained by a groundable regular derivation.

Proof. Combining Lemmas C.10 and C.8. □

Lemma C.12 Let S_0 be a set of initial closures, $\Delta = (S_0 \Rightarrow S_1 \Rightarrow \ldots)$ be any fair derivation, and S_ω be the limit of Δ. Let the closure $\mathcal{C}_1 \in S_\omega$ have the form

$$H \leftarrow T_1, s \simeq t, T_2 \cdot \theta$$

where H, T_1, T_2 do not contain equality and the substitution σ be a most general unifier of $s\theta$ and $t\theta$. Then the result of equality solution applied to C_1

$$\mathcal{C}_2 \rightleftharpoons (H \leftarrow T_1, T_2 \cdot \theta\sigma)$$

belongs to S_ω^-.

Proof. Fairness implies that there is S_j such that $\mathcal{C}_2 \in S_j$. Since $\mathcal{C}_2$ does not contain non-variable positions, the basic folding cannot be applied to $\mathcal{C}_2$. It follows that $\mathcal{C}_2 \in S_\omega$. Since $\mathcal{C}_2$ does not contain equality, $\mathcal{C}_2 \in S_\omega^-$. □

For a set of closures S, denote by $Res(S)$ the set of all closures derivable from S by (multiple) applications of resolution.

Lemma C.13 Let S_0 be a set of initial closures, $\Delta = (S_0 \Rightarrow S_1 \Rightarrow \ldots)$ be any fair derivation, and S_ω be the limit of Δ. Let the closure $\mathcal{C} \in S_j$ have the form

$$H \leftarrow T_1, s_1 \simeq t_1, T_2, s_2 \simeq t_2, \ldots, s_n \simeq t_n, T_{n+1} \cdot \theta$$

such that neither H nor any T_i contain equations. Let σ be a most general unifier of the set $\{s_1\theta \simeq t_1\theta, \ldots, s_n\theta \simeq t_n\theta\}$[3]. Then the closure

$$H \leftarrow T_1, T_2, \ldots, T_{n+1} \cdot \theta\sigma$$

belongs to $Res(S_\omega^-)$.

Proof. By induction on the ordinal $\omega^k + n$, where k is the number of non-variable positions in $\mathcal{C}$.

Let us note that since S_0 is initial, T_i do not contain non-variable positions.

$k, n = 0$ Obvious.

$k > 0$

1. Let $\mathcal{C} \in S_\omega$. By fairness, the result of equality solution

$$H \leftarrow T_1, T_2, s_2 \simeq t_2, \ldots, s_n \simeq t_n, T_{n+1} \cdot \theta\sigma'$$

   where σ' is a most general unifier of $s_1\theta, t_1\theta$, belongs to some S_i. Application of induction hypothesis to this closure and the use elementary properties of most general unifiers proves the claim.

[3]A unifier of a set of equations $\{s_1 \simeq t_1, \ldots, s_n \simeq t_n\}$ is any substitution σ such that $s_i\sigma = t_i\sigma$, for all $i \in \{1, \ldots, n\}$ (see [1]).

2. Let $\mathcal{C} \notin S_\omega$. Then there was a folding into $\mathcal{C}$ in Δ such that $\mathcal{C}$ was replaced by

$$\begin{array}{l} \mathcal{C}' = (H \leftarrow T_1, \ldots, s_i[x_0] \simeq t_i, \mathcal{E}_{t'}(x_0, \ldots, x_m), \mathcal{S}_\tau, \ldots, T_{n+1} \cdot \sigma \cup \theta) \\ Def_{t'} = (\mathcal{E}_{t'}(x_0, \ldots, x_m) \leftarrow x_0 \simeq t' \cdot \varepsilon) \end{array}$$

By Lemma C.12, $(\mathcal{E}_{t'}(x_0, x_1, \ldots, x_m) \cdot \{t'/x_0\}) \in S_\omega^-$. Applying induction hypothesis to $\mathcal{C}'$ we obtain some $\mathcal{C}'' \in Res(S_\omega^-)$. By resolving $\mathcal{C}''$ with $(\mathcal{E}_{t'}(x_0, x_1, \ldots, x_m) \cdot \{t'/x_0\}) \in S_\omega^-$ and using elementary properties of substitutions, we prove the claim.

□

The following definition is purely technical:

Definition C.14 (relations $\xrightarrow{m}$, $\xrightarrow{c}$ and $\xrightarrow{mc}$)
Introduce relations $\xrightarrow{m}$, $\xrightarrow{c}$ and $\xrightarrow{mc}$ on closures in the following way. $\mathcal{C}_1 \xrightarrow{m} \mathcal{C}_2$ iff one of the following holds:

1. $\mathcal{C}_1$ has the form $H \leftarrow T_1, A[t], T_2 \cdot \theta$ and $\mathcal{C}_2$ has the form $H \leftarrow T_1, A[x], x \simeq t, T_2 \cdot \theta$, where $x \notin var(\mathcal{C}_1)$, t is not a variable and $A[t]$ is not a variant of $x \simeq t$;

2. $\mathcal{C}_1$ has the form $A[t] \leftarrow T \cdot \theta$, t is not a variable and $\mathcal{C}_2$ has the form $A[x] \leftarrow x \simeq t, T \cdot \theta$, where $x \notin var(\mathcal{C}_1)$.

For two closures $\mathcal{C}_1, \mathcal{C}_2$ we have $\mathcal{C}_1 \xrightarrow{c} \mathcal{C}_2$ iff $\mathcal{C}_2$ can be obtained from $\mathcal{C}_1$ by a pseudo[4] application of equality solution.

Let $\xrightarrow{mc} \rightleftharpoons \xrightarrow{m} \cup \xrightarrow{c}$.

As usual $\xrightarrow{r*}$ denotes the transitive and reflexive closure of the relation $\xrightarrow{r}$, where r ranges over $\{m, c, mc\}$.

Lemma C.15 The relation $\xrightarrow{mc}$ is noetherian.

Proof. Let $\mathcal{C}_0 \rightleftharpoons (C \cdot \theta)$ and $\mathcal{C}_0 \cdots \mathcal{C}_i \xrightarrow{m} \mathcal{C}_{i+1} \cdots \mathcal{C}_j \xrightarrow{c} \mathcal{C}_{j+1} \cdots$ be a sequence of closures connected by $\xrightarrow{m}$ or $\xrightarrow{c}$. Then the number of $\xrightarrow{m}$ in this sequence is restricted by the number of non-variable positions in C, and the number of $\xrightarrow{c}$ in this sequence is restricted by the number of applications of $\xrightarrow{m}$ plus the number of equations in C. □

Lemma C.16 Let $\mathcal{C}, \mathcal{C}', \mathcal{C}''$ be closures such that $\mathcal{C} \xrightarrow{m} \mathcal{C}'$ and $\mathcal{C} \xrightarrow{c} \mathcal{C}''$. Then there is a closure $\mathcal{D}$ such that $\mathcal{C}' \xrightarrow{c*} \mathcal{D}$ and $\mathcal{C}'' \xrightarrow{m*} \mathcal{D}$, i.e. the following diagram commutes:

$$\begin{tikzcd} & \mathcal{C} \arrow[dl, "m"'] \arrow[dr, "c"] & \\ \mathcal{C}' \arrow[dr, dotted, "c^*"'] & & \mathcal{C}'' \arrow[dl, dotted, "m^*"] \\ & \mathcal{D} & \end{tikzcd}$$

[4]As before, "pseudo" means that we drop the requirement that equality solution be made into the leftmost equation in the tail.

Proof. There are two possible cases, in the first case $\xrightarrow{m}$ and $\xrightarrow{c}$ are applied to two different literals in $\mathcal{C}$, in the second case to the same literal. Both cases are simple, we shall only consider the second one.

Let $\mathcal{C}$ has the form $H \leftarrow T_1, p[t] \simeq q, T_2 \cdot \theta$ and both $\xrightarrow{m}$ and $\xrightarrow{c}$ are made in the literal $p[t] \simeq q$. Then

$$\begin{aligned}
\mathcal{C} &= (H \leftarrow T_1, p[t] \simeq q, T_2 \cdot \theta);\\
\mathcal{C}' &= (H \leftarrow T_1, p[x] \simeq q, x \simeq t, T_2 \cdot \theta);\\
\mathcal{C}'' &= (H \leftarrow T_1, T_2 \cdot \theta\sigma)
\end{aligned}$$

where σ is most general unifier of $p[t]\theta$ and $q\theta$. We define $\mathcal{D} \rightleftharpoons \mathcal{C}''$. On one hand, $\mathcal{C}'' \xrightarrow{m*} \mathcal{D}$. On the other hand, we have $\mathcal{C}' \xrightarrow{c} \mathcal{D}' \xrightarrow{c} \mathcal{D}$, where

$$\mathcal{D}' = (H \leftarrow T_1, p[x] \simeq q, T_2 \cdot \theta\{t\theta/x\})$$

□

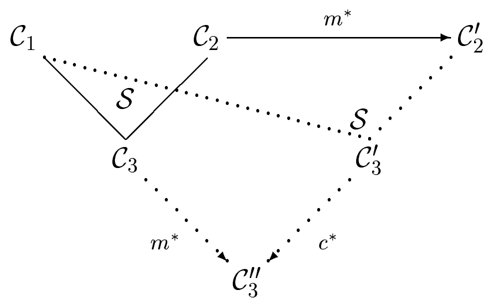
Lemma C.17 Let $\mathcal{C}, \mathcal{C}', \mathcal{C}''$ be closures such that $\mathcal{C} \xrightarrow{m*} \mathcal{C}'$ and $\mathcal{C} \xrightarrow{c*} \mathcal{C}''$. Then there is a closure $\mathcal{D}$ such that $\mathcal{C}' \xrightarrow{c*} \mathcal{D}$ and $\mathcal{C}'' \xrightarrow{m*} \mathcal{D}$, i.e. the following diagram commutes:

$$\begin{tikzcd}
 & \mathcal{C} \arrow[dl, "m^*"'] \arrow[dr, "c^*"] & \\
\mathcal{C}' \arrow[dr, dotted, "c^*"'] & & \mathcal{C}'' \arrow[dl, dotted, "m^*"] \\
 & \mathcal{D} &
\end{tikzcd}$$

Proof. Follows the scheme of the proof of [44] that any noetherian relation is confluent iff it is locally confluent, using Lemmas C.15 and C.16. □

Lemma C.18 Let $\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3, \mathcal{C}_2'$ be closures, $\mathcal{C}_3$ be obtained from $\mathcal{C}_1, \mathcal{C}_2$ by a pseudo application of superposition and $\mathcal{C}_2 \xrightarrow{m*} \mathcal{C}_2'$. Then there are closures $\mathcal{C}_3', \mathcal{C}_3''$ such that (see the picture below)

1. $\mathcal{C}_3'$ is obtained by a pseudo application of superposition to $\mathcal{C}_1, \mathcal{C}_2'$;
2. $\mathcal{C}_3' \xrightarrow{c*} \mathcal{C}_3''$;
3. $\mathcal{C}_3 \xrightarrow{m*} \mathcal{C}_3''$.



Proof. We use induction on the length n of the $\xrightarrow{m}$-chain from $\mathcal{C}_2$ to $\mathcal{C}_2'$.

Let $n = 1$. Among all possible variants of superposition between $\mathcal{C}_1, \mathcal{C}_2$ consider (the most interesting) case when it is a left superposition of $\mathcal{C}_1$ into $\mathcal{C}_2$. Then

$$\begin{aligned}
\mathcal{C}_1 &= (p \simeq q \leftarrow T_1 \cdot \theta_1);\\
\mathcal{C}_2 &= (H \leftarrow T_2, u[s] \simeq v, T_3 \cdot \theta_2);\\
\mathcal{C}_3 &= (H \leftarrow T_2, T_1, u[q] \simeq v, T_3 \cdot (\theta_1 \cup \theta_2)\rho)
\end{aligned}$$

where ρ is a most general unifier of $p\theta_1$ and $s\theta_2$. Let π_s be the position of s in $u[s]$. Let $\mathcal{C}_2 \xrightarrow{m} \mathcal{C}_2'$ by introducing the new atom $x \simeq t$. We only consider cases when the position π_t of the term t in $\mathcal{C}_2$ is either a subsequence of π_s or a supersequence of π_s. (It means that either that t occurs in s or s occurs in t.) All other cases are obvious.) Consider both cases:

1. $\pi_t \le \pi_s$, i.e. $\pi_s = \pi_t\pi$ for some π. Then $\mathcal{C}_2'$ has the form

   $$H \leftarrow T_2, u'[x] \simeq v, x \simeq t[s], T_3 \cdot \theta_2$$

   where $u'[x] \rightleftharpoons u[\pi_t \leftarrow x]$ and $t\,|_\pi = s$.

   We can apply superposition of $\mathcal{C}_1$ into $\mathcal{C}_2'$ obtaining

   $$\mathcal{C}_3' \rightleftharpoons (H \leftarrow T_2, T_1, u'[x] \simeq v, x \simeq t[q], T_3 \cdot (\theta_1 \cup \theta_2)\rho)$$

   If $t[q]$ is not a variable then define $\mathcal{C}_3'' \rightleftharpoons \mathcal{C}_3'$ and we obviously have $\mathcal{C}_3 \xrightarrow{m} \mathcal{C}_3''$. Otherwise we have $t[q] = q$ and $\pi = \varepsilon$. Define $\mathcal{C}_3'' \rightleftharpoons \mathcal{C}_3$. Then $\mathcal{C}_3' \xrightarrow{c} \mathcal{C}_3$ by the solution of equality $x \simeq t[q]$.

2. $\pi_s < \pi_t$, i.e. $\pi_t = \pi_s\pi$ for some $\pi \neq \varepsilon$. Then $\mathcal{C}_2'$ has the form

   $$H \leftarrow T_2, u[s'[x]] \simeq v, x \simeq t, T_3 \cdot \theta_2$$

   where $s' = s[\pi \leftarrow x]$. We can (pseudo) apply superposition of $\mathcal{C}_1$ into $\mathcal{C}_2'$ obtaining

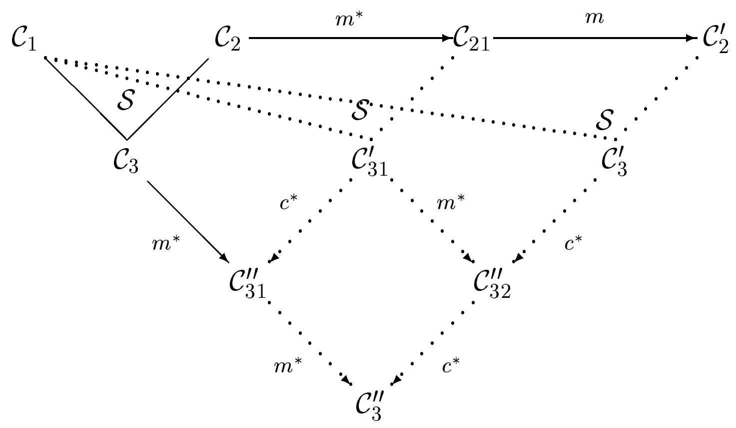
   $$\mathcal{C}_3' \rightleftharpoons (H \leftarrow T_2, T_1, u[q] \simeq v, x \simeq t, T_3 \cdot (\theta_1 \cup \theta_2)\rho')$$

   where ρ' is a most general unifier of $p\theta_1$ and $s'\theta_2$. We can obtain $\mathcal{C}_3' \xrightarrow{c} \mathcal{C}_3''$ by (pseudo) equality solution:

   $$\mathcal{C}_3'' \rightleftharpoons (H \leftarrow T_2, T_1, u[q] \simeq v, T_3 \cdot (\theta_1 \cup \theta_2)\rho'\sigma)$$

   where σ is a most general unifier of $x(\theta_1 \cup \theta_2)\rho'$ and $t(\theta_1 \cup \theta_2)\rho'$. It is easy to see that $\mathcal{C}_3''$ is a variant of $\mathcal{C}_3$.

Let $n > 1$. In this case the proof is illustrated by the following picture:



Then there is a closure $\mathcal{C}_{21}$ such that $\mathcal{C}_2 \xrightarrow{m*} \mathcal{C}_{21} \xrightarrow{m} \mathcal{C}_2'$. The induction hypothesis yields that there are $\mathcal{C}_{31}'$ and $\mathcal{C}_{31}''$ such that

1. $\mathcal{C}_{31}'$ is obtained by a pseudo application of superposition of $\mathcal{C}_1, \mathcal{C}_{21}$;
2. $\mathcal{C}_{31}' \xrightarrow{c*} \mathcal{C}_{31}''$;
3. $\mathcal{C}_3 \xrightarrow{m*} \mathcal{C}_{31}''$.

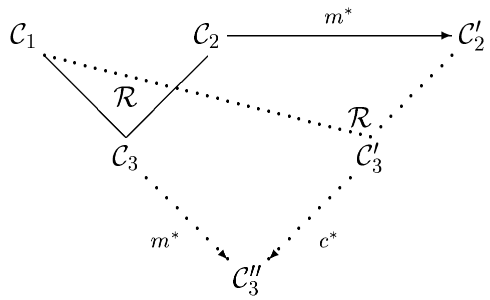
According to the proof for $n = 1$ we have that there are $\mathcal{C}_3'$ and $\mathcal{C}_{32}''$ such that

1. $\mathcal{C}_3'$ is obtained by a pseudo application of superposition to $\mathcal{C}_1, \mathcal{C}_2'$;
2. $\mathcal{C}_3' \xrightarrow{c*} \mathcal{C}_{32}''$;
3. $\mathcal{C}_{31}' \xrightarrow{m*} \mathcal{C}_{32}''$.

By Lemma C.17 there is a closure $\mathcal{C}_3''$ such that $\mathcal{C}_{31}'' \xrightarrow{m*} \mathcal{C}_3''$ and $\mathcal{C}_{32}'' \xrightarrow{c*} \mathcal{C}_3''$. This completes the proof since we have $\mathcal{C}_3 \xrightarrow{m*} \mathcal{C}_{31}'' \xrightarrow{m*} \mathcal{C}_3''$ and $\mathcal{C}_3' \xrightarrow{c*} \mathcal{C}_{32}'' \xrightarrow{c*} \mathcal{C}_3''$. □

Lemma C.19 Let $\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3, \mathcal{C}_2'$ be closures, $\mathcal{C}_3$ be obtained from $\mathcal{C}_1, \mathcal{C}_2$ by an application of resolution and $\mathcal{C}_2 \xrightarrow{m*} \mathcal{C}_2'$. Then there are closures $\mathcal{C}_3', \mathcal{C}_3''$ such that (see the picture below)

1. $\mathcal{C}_3'$ is obtained by an application of resolution to $\mathcal{C}_1, \mathcal{C}_2'$;
2. $\mathcal{C}_3' \xrightarrow{c*} \mathcal{C}_3''$;
3. $\mathcal{C}_3 \xrightarrow{m*} \mathcal{C}_3''$.



Proof. Similar to the proof of Lemma C.18. □

Lemma C.20 Let Π be a (groundable) pseudo **BR**-derivation of a closure $\mathcal{D}$ without equality from a set of closures $S \cup \mathcal{C}$ and $\mathcal{C} \xrightarrow{m*} \mathcal{C}_1$. Then there is a (groundable) pseudo **BR**-derivation Π' of $\mathcal{D}$ from $S \cup \mathcal{C}_1$ with the same number of applications of basic superposition.

Proof. Since **BR**-derivation are in a tree-like form, we can consider all closures occurring in leaves of a derivation as a multiset. We shall replace all *variants* of $\mathcal{C}$ by variants of $\mathcal{C}_1$ one by one not changing any other leaves.

Consider the case when Π consists of one node $\mathcal{C}$. Then $\mathcal{C} = \mathcal{D}$. Since $\mathcal{C}$ does not contain equality, we have $\mathcal{C} = \mathcal{C}_1$.

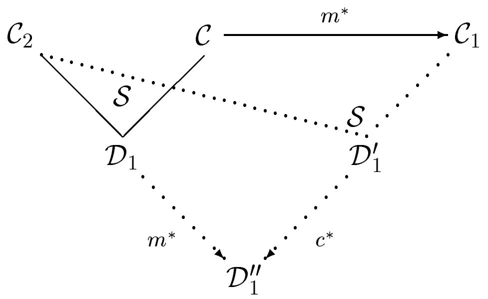
Assume now that Π consists of more than one node. Consider an occurrence of $\mathcal{C}$ in a list of Π. There are three possible inference rules applicable to $\mathcal{C}$: superposition, equality solution and resolution.

Assume that the rule applied to $\mathcal{C}$ is a superposition with a closure $\mathcal{C}_2$. Then Π has the form

$$\begin{array}{c} \Pi_2 \\ \dfrac{\mathcal{C}_2 \quad \mathcal{C}}{\mathcal{D}_1}\ (\mathcal{S}) \\ \ddots\vdots\iddots \\ \mathcal{D} \end{array}$$

By Lemma C.18 there exist closures $\mathcal{D}_1', \mathcal{D}_1''$ such that

1. $\mathcal{D}_1'$ is obtained by a pseudo application of superposition of $\mathcal{C}_1, \mathcal{C}_2$;
2. $\mathcal{D}_1' \xrightarrow{c*} \mathcal{D}_1''$;
3. $\mathcal{D}_1 \xrightarrow{m*} \mathcal{D}_1''$.



Consider the subderivation Π_1 of Π obtained by replacing the derivation of $\mathcal{D}_1$ by $\mathcal{D}_1$:

$$\begin{array}{c} \mathcal{D}_1 \\ \ddots\vdots\iddots \\ \mathcal{D} \end{array}$$

The number of nodes in this derivation is smaller than the number of nodes in Π. Thus, we can apply the induction hypothesis and obtain a derivation Π_1'' of the form

$$\begin{array}{c} \mathcal{D}_1'' \\ \ddots \vdots \cdot^{\cdot^{\cdot}} \\ \mathcal{D} \end{array}$$

which has the same multiset of closures in leaves as Π_1 except that the leaf $\mathcal{D}_1$ is replaced by $\mathcal{D}_1''$. By extending Π_1'' in the following way

$$\begin{array}{c} \Pi_2 \\ \dfrac{\mathcal{C}_2 \quad \mathcal{C}_1}{\mathcal{D}_1'}\ (\mathcal{S}) \\ \vdots \\ \mathcal{D}_1'' \\ \ddots \vdots \cdot^{\cdot^{\cdot}} \\ \mathcal{D} \end{array}$$

where the subderivation of $\mathcal{D}_1''$ from $\mathcal{D}_1'$ consists of equality solutions, we obtain the required derivation.

If the rule applied to the closure $\mathcal{C}$ is resolution then the proof is similar but using Lemma C.19 instead of Lemma C.18.

If the rule applied to the closure $\mathcal{C}$ is equality solution then the proof is similar but using Lemma C.17 instead of Lemma C.18. □

Definition C.21 (strengthening)
A closure $\mathcal{C}'$ is a *strengthening* of a closure $\mathcal{C}$ if

1. $\mathcal{R}(\mathcal{C}') = \mathcal{R}(\mathcal{C})$;
2. Every non-variable position in $\mathcal{C}$ is also a non-variable position in $\mathcal{C}'$.

Note that if $\mathcal{C}$ and its strengthening $\mathcal{C}'$ contain no equality literals, then $\mathcal{C}$ is a variant of $\mathcal{C}'$.

Lemma C.22 Let

$$\frac{\mathcal{C}_1 \quad \mathcal{C}_2}{\mathcal{C}}$$

be an application of an inference rule of **BR**. Let $\mathcal{C}_1'$ and $\mathcal{C}_2'$ be strengthenings of $\mathcal{C}_1$ and $\mathcal{C}_2$, respectively. Then there exists a strengthening $\mathcal{C}'$ of $\mathcal{C}$ obtained by the same application of the inference rule:

$$\frac{\mathcal{C}_1' \quad \mathcal{C}_2'}{\mathcal{C}'}$$

Proof. Obvious. □

Lemma C.23 Let $\mathcal{C}_i'$ be strengthenings of $\mathcal{C}_i$, for $i \in \{1, \ldots, n\}$. If there exists a groundable **BR**-derivation of $\mathcal{C}$ from $\mathcal{C}_1, \ldots, \mathcal{C}_n$, then there exists a groundable **BR**-derivation of a closure $\mathcal{C}'$ from $\mathcal{C}_1', \ldots, \mathcal{C}_n'$ with the same ground version such that $\mathcal{C}'$ is a strengthening of $\mathcal{C}$.

Proof. Straightforward from Lemma C.22. □

Lemma C.24 Let there be a (groundable) pseudo **BR**-derivation of a closure $\mathcal{D}$ without equality from a set of closures S. Let $\mathcal{C} \in S$ and $\mathcal{C}'$ be obtained by a basic folding into $\mathcal{C}$:

$$\frac{\{\mathcal{C}\}}{\{\mathcal{C}', Def_{t'}\}}\ \mathcal{BF}$$

Then there is a (groundable) pseudo **BR**-derivation of $\mathcal{D}$ from $S \setminus \{\mathcal{C}\} \cup \{\mathcal{C}', Def_{t'}\}$ with the same number of applications of basic superposition.

Proof. According to the definition of the basic folding, $\mathcal{C}, \mathcal{C}'$ and $Def_{t'}$ have the form

$$\begin{array}{l} \mathcal{C} = (H \leftarrow T_1, A[t\tau], T_2 \cdot \theta); \\ \mathcal{C}' = (H \leftarrow T_1, A[x_0], \mathcal{E}_{t'}(x_0, x_1, \ldots, x_n), \mathcal{S}_\tau, T_2 \cdot \sigma \cup \theta); \\ Def_{t'} = (\mathcal{E}_{t'}(x_0, x_1, \ldots, x_n) \leftarrow x_0 \simeq t' \cdot \varepsilon) \end{array}$$

Applying resolution to $\mathcal{C}'$ and $Def_{t'}$, we obtain the clause

$$\mathcal{C}'' \rightleftharpoons (H \leftarrow T_1, A[x_0], x_0 \simeq t', \mathcal{S}_\tau, T_2 \cdot \sigma \cup \theta)$$

Let

$$\mathcal{C}_1 \rightleftharpoons (H \leftarrow T_1, A[x_0], x_0 \simeq t, \mathcal{S}_\tau, T_2 \cdot \theta)$$

It is easy to see that $\mathcal{C} \xrightarrow{m*} \mathcal{C}_1$. By Definition 4.5 of the basic folding we have that $\mathcal{C}''$ is a strengthening of $\mathcal{C}_1$. By Lemma C.20 there is a (groundable) pseudo **BR**-derivation of $\mathcal{D}$ from $S \setminus \{\mathcal{C}\} \cup \{\mathcal{C}_1\}$ with the same number of applications of basic superposition. By Lemma C.23, $\mathcal{C}''$ to be used in the derivation instead of $\mathcal{C}_1$.

Thus, there is a (groundable) pseudo **BR**-derivation of $\mathcal{D}$ from $S \setminus \{\mathcal{C}\} \cup \{\mathcal{C}''\}$ and hence from $S \setminus \{\mathcal{C}\} \cup \{\mathcal{C}', Def_{t'}\}$ with the same number of applications of basic superposition. □

Now we can prove the completeness theorem.

Theorem 5.2 Let S_0 be a set of initial closures, $\Delta = (S_0 \Rightarrow S_1 \Rightarrow \ldots)$ be any fair derivation, and S_ω be the limit of Δ. If S_0 has a correct answer then S_ω^- has a correct answer.

Proof. By Lemma C.11, there is a groundable regular **BR**-derivation Π of $query(\bar{t})$. The proof is by induction on the number n of applications of superposition in Π.

$n = 0$ By Lemma C.13, $query(\bar{t})$ is derivable from S_ω^- by resolutions only. Hence, $query(\bar{t})$ is a logical consequence of S_ω^-.

$n > 0$ Consider a topmost application of superposition in Π, obtaining a closure $\mathcal{C}$ from $\mathcal{C}_1, \mathcal{C}_2 \in S_\omega$. By fairness of Δ we have $\mathcal{C} \in S_j$ for some j. The derivation of $query(\bar{t})$ from $S_\omega \cup \{\mathcal{C}\}$ has $n-1$ applications of superposition. By Lemmas 5.1 and C.24, there is a subset $S' \subseteq S_\omega$ obtained from $\{\mathcal{C}\}$ by a finite number of applications of the basic folding such that there is a regular groundable **BR**-derivation of $query(\bar{t})$ from $S_\omega \cup S'$ with $n-1$ applications of superposition. Note that $S_\omega \cup S' = S_\omega$ and apply induction hypothesis.

□

Bibliography

[1] K.R. Apt. Logic programming. In J. Van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Methods and Semantics, chapter 10, pages 493–574. Elsevier Science, Amsterdam, 1990.

[2] 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.

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

[4] 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.

[5] L. Bachmair, H. Ganzinger, C. Lynch, and W. Snyder. Basic paramodulation. *Information and Computation*, 121:172–192, 1995.

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

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

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

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

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

[11] A. Degtyarev and A. Voronkov. Equality control methods in machine theorem proving. *Cybernetics*, 22(3):298–307, 1986.

[12] A. Degtyarev and A. Voronkov. Equality elimination for the inverse method and extension procedures. In C.S. Mellish, editor, *Proc. International Joint Conference on Artificial Intelligence (IJCAI)*, volume 1, pages 342–347, Montréal, August 1995.

[13] A. Degtyarev and A. Voronkov. General connections via equality elimination. In M. De Glas and Z. Pawlak, editors, *Second World Conference on the Fundamentals of Artificial Intelligence (WOCFAI-95)*, pages 109–120, Paris, July 1995. Angkor.

[14] A. Degtyarev and A. Voronkov. A new procedural interpretation of Horn clauses with equality. In Leon Sterling, editor, *Proceedings of the Twelfth International Conference on Logic Programming*, pages 565–579. The MIT Press, 1995.

[15] A. Degtyarev and A. Voronkov. Equality elimination for the tableau method. In J. Calmet and C. Limongelli, editors, *Design and Implementation of Symbolic Computation Systems. International Symposium, DISCO'96*, volume 1128 of *Lecture Notes in Computer Science*, pages 46–60, Karlsruhe, Germany, September 1996.

[16] A. Degtyarev and A. Voronkov. Handling equality in logic programs via basic folding. In R. Dyckhoff, H. Herre, and P. Schroeder-Heister, editors, *Extensions of Logic Programming (5th International Workshop, ELP'96)*, volume 1050 of *Lecture Notes in Computer Science*, pages 119–136, Leipzig, Germany, March 1996.

[17] A. Degtyarev and A. Voronkov. A note on semantics of logics programs with equality based on complete sets of E-unifiers. *Journal of Logic Programming*, 28(3):207–216, September 1996.

[18] 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.

[19] E. Eder. Properties of substitutions and unifications. *Journal of Symbolic Computations*, 1(1):31–48, 1985.

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

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

[22] 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.

[23] 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.

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

[25] 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.

[26] J.C. González-Moreno, M.T. Hortalá-González, and M. Rodríguez-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.

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

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

[29] P. Hoddinott and E.W. Elcock. PROLOG: Subsumption of equality axioms by the homogeneous form. In *IEEE International Symposium on Logic Programming*, pages 115–126, 1986.

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

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

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

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

[34] 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.

[35] S. Kaplan. Conditional rewrite rules. *Theoretical Computer Science*, 33:175–193, 1984.

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

[37] 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.

[38] 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.

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

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

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

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

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

[44] M.H.A. Newman. On theories with combinatorial definition of "equivalence". *Annals of Mathematics*, 43(2):223–243, 1942.

[45] R. Nieuwenhuis. On narrowing, refutation proofs and constraints. In J. Hsiang, editor, *Rewriting Techniques and Applications*, volume 914 of *Lecture Notes in Computer Science*, pages 56–70, 1995.

[46] 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.

[47] R. Nieuwenhuis and A. Rubio. Theorem proving with ordering and equality constrained clauses. *Journal of Symbolic Computations*, 19:321–351, 1995.

[48] V.P. Orevkov. On nonlengthening applications of equality rules (in Russian). *Zapiski Nauchnyh Seminarov LOMI*, 16:152–156, 1969. English Translation in: Seminars in Mathematics: Steklov Math. Inst. 16, Consultants Bureau, NY-London, 1971, p.77-79.

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

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

[51] G. Plotkin. Building-in equational theories. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, volume 7, pages 73–90. Edinburgh University Press, 1972.

[52] G. Robinson and L. Wos. Completeness of paramodulation. *Journal of Symbolic Logic*, 34(1):159–160, 1969.

[53] 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.

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

[55] 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.

[56] 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.

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

[58] 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.

[59] S. Winker. Robbins algebra: Conditions that make a near-boolean algebra boolean. *Journal of Automated Reasoning*, 6(4):465–489, 1990.

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

[61] H. Zhang. Automatic proofs of equality problems in Overbeek's competition. *Journal of Automated Reasoning*, 11(3):333–351, 1993.

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