EXTENDING SLD RESOLUTION TO EQUATIONAL HORN CLAUSES USING *E*-UNIFICATION*

JEAN H. GALLIER AND STAN RAATZ

▷ We study the role of unification modulo a set of equations, or *E*-unification, in the context of refutation methods for sets of Horn clauses with equality. Two extensions of SLD resolution based on *E*-unification are presented, and rigorous completeness results are shown, including an analysis of the ground case for insight into the computational implications. The concept of a congruence closure generalized to sets of ground Horn clauses is central to these completeness results. The first method is general, in that it applies to arbitrary sets of equational Horn clauses, but is not practical, as it assumes a procedure which gives an explicit sequence of substitutions for each *E*-unifier. A second method uses a procedure enumerating a complete set of *E*-unifiers, and appears to be well suited to a class of "well-behaved" equational logic programs which allows a clean and natural integration of the functional and logic-programming paradigms. Using this second method, we have formalized the refutation method used in Eqlog for this class of programs, and a theorem establishes rigorously the completeness of this method. We compare these methods in detail with related work, and show that other methods either explicitly include *E*-unification or simulate it in some manner. ◁

1. INTRODUCTION

This paper presents two refutation methods for establishing the unsatisfiability of sets of Horn clauses with or without equational atomic formulae. The methods are

*This research was partially supported by the National Science Foundation under Grant DCR-86-07156.

Address correspondence to Professor J. Gallier, Department of Computer Science, University of Pennsylvania, Philadelphia, PA 19104.

Received 15 June 1987; accepted 10 September 1987

THE JOURNAL OF LOGIC PROGRAMMING

655 Avenue of the Americas, New York, NY 10010 0743-1066/89/$3.50

extensions of SLD resolution which incorporate unification modulo a set E of equations, or E-unification. We present first the SLDE-refutation method, which, given some assumptions on the E-unification procedure used, is complete for any arbitrary set of equational Horn clauses. After this, a refinement, called the SLDE†-refutation method, is presented and shown to be complete given *any* E-unification procedure that enumerates a complete set of E-unifiers for a class of equational Horn-clause programs called "well-behaved" programs. This class of programs consists of sets for which there is a procedure for enumerating a complete set of E-unifiers, and logic programs which contain clauses of the form

$$s \doteq t,$$

or

$$Q :- P_1, \ldots, P_n,$$

where s and t are first-order terms, Q is a nonequational atomic formula, and $P_1, \ldots, P_n$ are either equational or nonequational atomic formulae. This is an important class of equational Horn-clause programs (first introduced in Eqlog [17]), in that it subsumes the paradigms of functional, logic, and equational programming. The methods are defined rigorously, and the completeness issues are investigated in depth, including an analysis of the ground case, in a way that makes evident the practical implications of including equality in logic programs.

We show that the notion of E-unification comes up naturally when the Herbrand-Skolem-Gödel theorem is applied to a set of Horn clauses with equality, and that other methods for establishing the unsatisfiability of equational Horn clauses, which do not explicitly include the formalism of E-unification, can in fact be interpreted as doing so. Two consequences of an approach based on E-unification are that it allows a clean separation, and therefore handling, of the purely logical features and the equational features of a logic program, and that it makes available the fast-growing body of knowledge on the subject of E-unification in the construction of logic interpreters. These observations turn out to be valuable when considering the practical consequences of incorporating equality in Horn-clause logic programs.

Let us be very clear about what is being claimed here. We do not claim to have defined a fundamentally *new* form of SLD resolution with equality. Plotkin [44] in 1972 essentially laid out the basis for the incorporation of equality using E-unification into resolution theory, and more recently, Goguen and Meseguer [17] defined a logic-programming system, Eqlog, which incorporates E-unification into SLD resolution and specifically admits the class of well-behaved programs, among others. The claim is the following. The methods here are refutation methods for Horn clauses with or without equality whose completeness proofs rigorously and specifically illustrate the *computational* implications of including equality for certain classes of Horn clauses. In particular, we have formalized the refutation method used in Eqlog for well-behaved programs, and Theorem 7.5 establishes rigorously the completeness of this method for this class of programs.

There has been a substantial amount of work recently on defining integrations of functional and logic-programming systems by equational Horn-clause programming which either does not consider completeness issues, or considers these issues in a

manner that does not give any insight into the complexity or practical implications of the method. Our methods allow us to identify clearly certain classes of Horn-clause programs with equality that appear to be hopelessly impractical, and others, the well-behaved class for instance, that are promising. Since the prime motivation of this field of work is the *natural* and *efficient* integration of equality in logic programs in order to design languages and their interpreters allowing the use of functional and logic-programming paradigms, this identification is crucial.

The organization of the paper is now explained. We believe that the clearest way to present our results is to follow the path that led us to the discovery of the first-order refutation methods. In order to get some insight into the problem, we started by examining the ground case. Extending some previous work on the unsatisfiability of propositional Horn clauses without equality (Dowling and Gallier [7]), the first author found two fast algorithms for testing the unsatisfiability of a set of ground Horn clauses with or without equational atomic formulae. The crucial idea is that the concept of a *congruence closure* (Kozen [33, 34], Nelson and Oppen [42]) can be generalized to sets of ground Horn clauses. These algorithms are presented in Gallier [14] and are not repeated here. However, since the correctness proof of the congruence closure method is crucial to the completeness of the refutation methods, it will be presented in full. The ground case will be presented in Sections 3 and 4, after a brief section of preliminaries.

Then the SLDE-refutation method will be presented for the ground case, and its completeness will be established. In the next section, we show how the lifting of ground SLDE refutations to the first-order case via the Herbrand-Skolem-Gödel theorem gives rise naturally to the concept of E-unification. The SLDE-refutation method using E-unifiers will be defined and shown to be complete. However, the SLDE-refutation method may require the finding of arbitrary E-unifiers, which is prohibitive in practice. In order to remedy this problem, we introduce a refinement of the method, called SLDE† refutation, that requires "only" an explicit procedure for enumerating a complete set of E-unifiers. This method is complete for procedures enumerating *all* E-unifiers, but we do not know whether it is complete for all E-unification procedures (enumerating a complete set of E-unifiers). However, we prove that it is complete for all E-unification procedures (enumerating a complete set of E-unifiers) for the class of well-behaved sets of Horn clauses. Finally, we compare in detail these methods with the related work of Jaffar, Lassez, and Maher [24], Fribourg [12, 13], Dershowitz and Plaisted [6], and Goguen and Meseguer [17], and show that all of these methods embody a form of E-unification, even if it is not explicitly defined as such. We also review some of the numerous attempts to combine specific features of functional and logic-programming paradigms, but not in detail. Finally, we conclude with a discussion of the practical implications of including equality in logic programming.

2. PRELIMINARIES

This section contains a brief review of the main concepts used in this paper. As much as possible, we stick to the definitions used in the literature on the subject. More specifically, we will follow Huet and Oppen [22] and Gallier [15]. The purpose of this section is mainly to establish the terminology and the notation, and it can be

omitted by readers familiar with the literature. First, we review the basics of many-sorted languages.

Definition 2.1. A set S of *sorts* (or *types*) is any nonempty set. Typically, S consists of types in a programming language (such as *integer*, *real*, *boolean*, *character*, etc.) An *S-ranked alphabet* is a pair (Σ, ρ) consisting of a set Σ together with a function $\rho: \Sigma \to S \times S$ assigning a *rank* (u, s) to each symbol f in Σ. The string u in S^* is the *arity* of f, and s is the *sort* (or *type*) of f. If $u = s_1 \dots s_n$ $(n \geq 1)$, a symbol f of rank (u, s) is to be interpreted as an operation taking arguments, the ith argument being of type s_i and yielding a result of type s. A symbol of rank (e, s) (when u is the empty string) is called a *constant of sort s*. For simplicity, a ranked alphabet (Σ, ρ) is often denoted by Σ.

Next, we review the definition of tree domains and trees (or terms). Let $\mathbf{N}$ denote the set of natural numbers, and $\mathbf{N}_+$ the set of positive natural numbers.

Definition 2.2. A *tree domain D* is a nonempty subset of strings in $\mathbf{N}_+^*$ satisfying the conditions:

(1) For all $u, v \in \mathbf{N}_+^*$, if $uv \in D$ then $u \in D$.

(2) For all $u \in D$, for every $i \in \mathbf{N}_+$, if $ui \in D$ then, for every j, $1 \leq j \leq i$, $uj \in D$.

For every $n \in \mathbf{N}$, let $[n] = \{1, 2, \dots, n\}$, and $[0] = \emptyset$.

Definition 2.3. Given an S-sorted ranked alphabet Σ, a *Σ-tree* (or *term*) *of sort s* is any function $t: D \to \Sigma$, where D is a tree domain denoted by $\mathrm{dom}(t)$, and t satisfies the following conditions:

(1) The root of t is labeled with a symbol $t(e)$ in Σ of sort s.

(2) For every node $u \in \mathrm{dom}(t)$, if $\{i | ui \in \mathrm{dom}(t)\} = [n]$, then if $n > 0$, for each ui, $i \in [n]$, if $t(ui)$ is a symbol of sort v_i, then $t(u)$ has rank (v, s'), with $v = v_1 \dots v_n$, else if $n = 0$, then $t(u)$ has rank (e, s') for some $s' \in S$.

Given a tree t and some tree address $u \in \mathrm{dom}(t)$, the *subtree of t rooted at u* is the tree t/u whose domain is the set $\{v | uv \in \mathrm{dom}(t)\}$ and such that $t/u(v) = t(uv)$ for all v in $\mathrm{dom}(t/u)$.

The set of all finite trees of sort s is denoted by T_Σ^s, and the set of all finite trees by T_Σ.

In this paper, it is assumed that for every S-sorted alphabet Σ, there is a distinguished sort $bool \in S$. Symbols of sort *bool* are called *predicate symbols*. Terms of sort *bool* will be interpreted as logical formulae.

Given an S-indexed family $X = (X_s)_{s \in S}$, we can form the sets of trees $T_\Sigma^s(X)$ obtained by adjoining each set X_s to the set of constants of sort s. To prevent free algebras from having empty carriers, so that the Herbrand-Skolem-Gödel theorem holds, we assume that every sort is *nonvoid*. We say that a sort s is *nonvoid* iff either there is some constant of sort s, or there is some function symbol f of rank $\rho(f) = (s_1 \dots s_n, s)$ such that $s_1, \dots, s_n$ are nonvoid. Then, for every sort s, the set T_Σ^s is nonempty, and it is well known that for every set X, $T_\Sigma(X)$ is the free

Σ-algebra generated by X (see Gallier [15]). This allows us to define substitutions. Let $X = (X_s)_{s \in S}$ be an S-indexed family of countable sets of variables.

Definition 2.4. Given a term t, the set of variables occurring in t is the set $\{x \in X | \exists u \in \mathrm{dom}(t), t(u) = x\}$, and it is denoted by $\mathrm{Var}(t)$.

Definition 2.5. A *substitution* is any function $\sigma : X \to T_\Sigma(X)$ such that $\sigma(x) \neq x$ for only finitely many $x \in X$. Since $T_\Sigma(X)$ is the free Σ-algebra generated by X, every substitution $\sigma : X \to T_\Sigma(X)$ has a unique homomorphic extension $\hat{\sigma} : T_\Sigma(X) \to T_\Sigma(X)$.

Definition 2.6. Given a substitution σ, the *support* (or *domain*) of σ is the set of variables $D(\sigma) = \{x | \sigma(x) \neq x\}$. The set of variables *introduced by* σ is the set of variables $I(\sigma) = \bigcup_{x \in D(\sigma)} \mathrm{Var}(\sigma(x))$. Given a substitution σ, if its support is the set $\{x_1, \ldots, x_n\}$, and if $t_i = \sigma(x_i)$, $1 \leq i \leq n$, then σ is also denoted by $[t_1/x_1, \ldots, t_n/x_n]$.

Definition 2.7. Given two substitutions σ and θ, their *composition* is the substitution denoted by $\sigma \circ \theta$, such that, for every variable x, we have $\sigma \circ \theta(x) = \hat{\theta}(\sigma(x))$ (the composition of the functions σ and $\hat{\theta}$).

Even though the notation $\sigma \circ \theta$ is slightly misleading (since $\sigma \circ \theta$ is *not* the composition of σ and θ, but the composition of σ and $\hat{\theta}$), it is a natural consequence of the identification a substitution σ with its homomorphic extension $\hat{\sigma}$ made for notational simplicity. This convention is harmless since it is easily shown that $\widehat{\sigma \circ \hat{\theta}} = \hat{\sigma} \circ \hat{\theta}$.

The operation of tree replacement (or tree substitution) will also be needed.

Definition 2.8. Given two trees t_1 and t_2 and a tree address u in t_1, the *result of replacing* t_2 at u in t_1, denoted by $t_1[u \leftarrow t_2]$, is the function whose graph is the set of pairs

$$\{(v, t_1(v)) | u \text{ is not a prefix of } v\} \cup \{(uw, t_2(v))\},$$

and it is only defined provided that the sort of the root of t_2 is equal to the sort of $t_1(u)$.

We also review the definition of a Horn clause. For details, see Gallier [15].

Definition 2.9. An atomic formula is either a term of the form $Pt_1 \ldots t_n$, where P is a predicate symbol of rank $(s_1 \ldots s_n, bool)$ and each t_i is a term of sort s_i, or a term of the form $t_1 \doteq t_2$, where t_1 and t_2 are terms of some identical sort s. An atomic formula of the form $t_1 \doteq t_2$ is called an *equation of sort s*. A *literal* is either an atomic formula or the negation of an atomic formula. A *Horn clause* is a set of literals containing at most one positive literal (unnegated). Horn clauses are classified into two classes. A *definite clause* is a Horn clause containing some positive literal. A definite clause is denoted by

$$A :- B_1, \ldots, B_n, \quad \text{or} \quad A \quad \text{when } n = 0,$$

where A is the positive literal. A *goal clause* (or *negative clause*) only contains negative literals. A goal clause is denoted by

$$:- B_1, \ldots, B_n, \quad \text{or} \quad \square \quad \text{when} \quad n = 0.$$

The clause $\square$ is called the *empty clause*.

We will often use the term *equational Horn-clause language* to mean a system which admits sets of Horn clauses including equational atomic formulae, in order to make the distinction between such systems and more conventional logic-programming systems.

Definition 2.10. Give a clause C, the set of variables occurring in C is the union of the sets of variables occurring in the literals in C, and it is denoted by $\mathrm{Var}(C)$. A *ground term* t is a term such that $\mathrm{Var}(t) = \emptyset$, and similarly a *ground clause* C is a clause such that $\mathrm{Var}(C) = \emptyset$. A *ground substitution* σ is a substitution such that $\sigma(x)$ is a ground term for every variable x in the support of σ.

Finally, observe that the Herbrand-Skolem-Gödel theorem holds for many-sorted languages for which all sorts are nonvoid (see Gallier [15]).

3. CONGRUENCES ASSOCIATED WITH SETS OF HORN CLAUSES

We first consider how to test the unsatisfiability of a ground set of Horn clauses with equality. This section is not the main theme of this paper, but it is technically important because it contains a theorem (Theorem 3.6) that plays a crucial role in the proofs that the refutation methods presented in this paper are complete. The central concept used in the next two sections is that of a congruence closure. It will allow us to show that unsatisfiability of ground Horn clauses is decidable. Readers not interested in the details of these completeness proofs can omit Sections 3 and 4 at first reading, and proceed directly to the presentation of the ground refutation method (Section 5).

3.1. Informal Description of the Method

Testing the unsatisfiability of a set of ground Horn clauses with or without equational atomic formulae is decidable, and the first author has given two fast algorithms solving this problem [14]. If the length of the set of Horn clauses (viewed as the string obtained by concatenating the clauses in H) is n, then the first algorithm runs in time $O(n^2)$ and storage $O(n)$, and the second algorithm runs in time $O((n \log^2 n)/\log k)$ and storage $O(kn)$, for any k chosen in advance. These algorithms are obtained by combining the methods used in two other algorithms:

(1) The linear-time algorithm of Dowling and Gallier for testing the satisfiability of a set of propositional Horn clauses [7].

(2) The congruence closure algorithms of Kozen [33, 34], Nelson and Oppen [42], and Downey, Sethi, and Tarjan [8].

In this paper, we are not so much concerned with the algorithms themselves as with the underlying method and its correctness proof, because it is crucial to the

proof that the refutation methods presented in this paper are complete. The crucial idea is that the concept of a *congruence closure* can be generalized to sets of ground Horn clauses. In this generalization, two graphs are used. The first graph GT(H), similar to the graph used in the congruence closure method (Kozen [33, 34]; Nelson and Oppen [42, 43]) represents subterm dependencies. As in Gallier [15], an extra node $\top$ (the constant **true**) is added to take care of nonequational atomic formulae. The second graph GC(H) (similar to the graph used in Dowling and Gallier [7]) represents implications induced by the clauses.

Now, a set H of ground Horn clauses induces a relation E on the set of nodes of the graph GT(H) defined as follows: For every clause in H consisting of an atomic (positive) formula B

(1) if B is an atomic formula $Pt_1 \dots t_n$, then $(Pt_1 \dots t_n, \top) \in E$;
(2) if B is an equation $t_1 \doteq t_2$, then $(t_1, t_2) \in E$.

Then, a certain kind of congruence closure $\overset{*}{\leftrightarrow}_E$ of E with respect to the graph GT(H) can be defined. The crucial fact about this congruence is that H is unsatisfiable iff there is some negative clause $:- A_1, \dots, A_n \in H$ such that, for every i, $1 \leq i \leq n$, if A_i is of the form $Pt_1 \dots t_k$, then $Pt_1 \dots t_k \overset{*}{\leftrightarrow}_E \top$ else if A_i is of the form $t_1 \doteq t_2$, then $t_1 \overset{*}{\leftrightarrow}_E t_2$.

In order to compute this congruence closure, two other closures defined in terms of the graphs GT(H) and GC(H) are used. The equational congruence closure $\overset{*}{\equiv}$ is defined in terms of the graph GT(H), and it is used to propagate congruence resulting from purely equational reasons. The implicational closure $\overset{*}{\supset}$ is defined in terms of the graph GC(H), and it is used to propagate congruence resulting from purely implicational reasons. Then, the congruence closure $\overset{*}{\leftrightarrow}_E$ associated with the set H is obtained by interleaving equational congruence closure steps and implicational closure steps. We now present the method rigorously.

Let H be a set of ground Horn clauses, possibly with equational atoms. First, we make the following observation. Since our language already has the special sort *bool*, we can go a little further and add the constant $\top$ interpreted as **true**, and treat every atomic formula as an equation. Indeed, for every structure, the domain BOOL of sort *bool* is the set of truth values {**true, false**}, and every atomic formula $Pt_1 \dots t_k$ is logically equivalent to the equation $(Pt_1 \dots t_k \equiv \top)$, in the sense that $Pt_1 \dots t_k \equiv (Pt_1 \dots t_k \equiv \top)$ is valid. Since $\equiv$ behaves semantically exactly as the identity relation on BOOL, we can treat $\equiv$ as the equality symbol $\doteq_{bool}$ of sort *bool*. Hence, every set H of Horn clauses is equivalent to a set H' of Horn clauses over a many-sorted language with the special sort *bool*, in which every atomic formula $Pt_1 \dots t_k$ is replaced by the equation $Pt_1 \dots t_k \equiv \top$.

For notational simplicity, we often denote an equation $Pt_1 \dots t_k \equiv \top$ of sort *bool* as $Pt_1 \dots t_k$ and call it a nonequational atom, and we reserve the word equation for equations of sort $\neq_{bool}$. In the sequel, we assume that sets of Horn clauses have been preprocessed as explained above.

3.2. *The Graph* GT(*H*)

The graph GT(H) represents subterm dependencies, and it is used to propagate congruential information. This graph was first defined by Kozen (under a different name) to study the properties of finitely presented algebras [33–36].

Definition 3.1. Given a set H of ground Horn clauses over a many-sorted language, let TERM(H) be the set of all subterms of terms occurring in the atomic formulae in H. Let $S(H)$ be the set of sorts of all terms in TERM(H). For every sort s in $S(H)$, let TERM$(H)_s$ be the set of all terms of sort s in TERM(H). Note that by the definition, each set TERM$(H)_s$ is nonempty. Let Σ be the $S(H)$-ranked alphabet consisting of all constant and function symbols occurring in TERM(H). The graph GT(H) has the set TERM(H) as its set of nodes, and its edges and the function Λ labeling its nodes are defined as follows:

for every node t in TERM(H), if t is a constant, then $\Lambda(t) = t$, else t is of the form $fy_1 \dots y_k$ and $\Lambda(t) = f$;

for every node t in TERM(H), if t is of the form $fy_1 \dots y_k$, then t has exactly k successors $y_1, \dots, y_k$, else t is a constant and it is a terminal node of GT(H).

Given a node $u \in$ TERM(H), if $\rho(\Lambda(u)) = (s_1 \dots s_n, s)$, $n > 0$, then the ith successor of u is denoted by $u[i]$. For every $s \in S(H)$, let $E_s = \{(r, t) | r \doteq_s t \in H\}$, and let E be the $S(H)$-indexed family $(E_s)_{s \in S(H)}$.

Example 3.2. Consider the following set H of ground Horn clauses:

$$f^3a \doteq a :- fa \doteq fb, \tag{1}$$

$$a \doteq b, \tag{2}$$

$$Pa, \tag{3}$$

$$f^5a \doteq a :- Qa, \tag{4}$$

$$Qa :- f^3a \doteq a, \tag{5}$$

$$Ra :- fa \doteq a, Pfa, \tag{6}$$

$$:- Rfa. \tag{7}$$

The graph GT(H) representing the subterm dependencies of the set H is shown in Figure 1.

3.3. *The Graph* GC*(H)*

The graph GC represents implicational information, and was defined in Dowling and Gallier [7].

Definition 3.3. The nodes of the graph GC(H) are the atomic formulae occurring in all clauses in the set H, plus the special nodes $\top$ and $\bot$ (where $\bot$ is the constant interpreted as **false**). The edges and the function Δ labeling the edges of GC(H) are defined as follows:

For every clause C of the form $B :- A_1, \dots, A_n$ in H, for every i, $1 \le i \le n$, there is an edge from B to A_i labeled with C.

For every clause N of the form $:- A_1, \dots, A_n$ in H, for every i, $1 \le i \le n$, there is an edge from $\bot$ to A_i labeled with N.

For every clause C of the form B, there is one edge from B to $\top$ labeled with C.

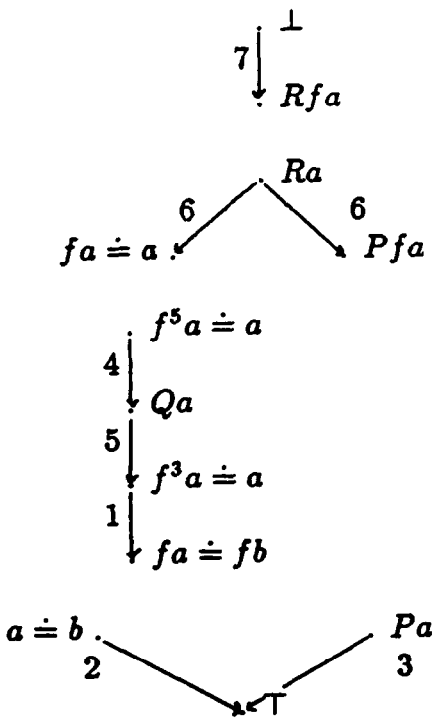

FIGURE 1. Graph $GT(H)$.

Note that since every atomic formula B is an equation $t_1 \doteq t_2$ (where t_2 may be $\top$), every node of the graph GC(H) corresponds to a unique pair of nodes in the graph GT(H).

Example 3.4. Consider again the set H of Horn clauses in the previous example. This set has the graph GC(H) shown in Figure 2.

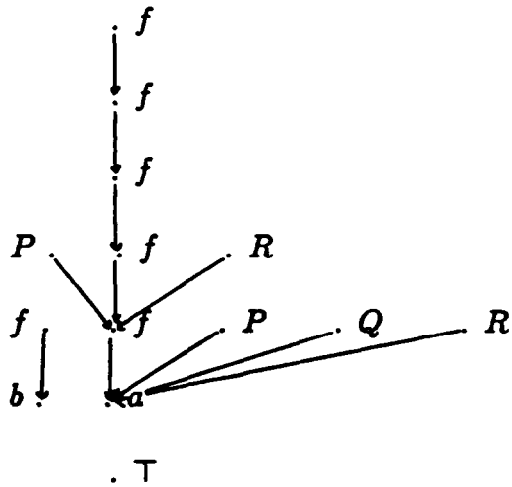

FIGURE 2. Graph $GC(H)$.

3.4. *Congruence Closure*

The crucial concept in showing the decidability of unsatisfiability for ground equational Horn clauses is a certain kind of equivalence relation on the graph GT(H) called a congruence.

Definition 3.5. Given the graph GT(H) associated with the set H of ground Horn clauses, an $S(H)$-indexed family R of relations R_s over TERM(H)$_s$ is a *congruence on* GT(H) iff:

(1) Each R_s is an equivalence relation.

(2) For every pair $(u, v) \in \text{TERM}(H)^2$, if $\Lambda(u) = \Lambda(v)$, $\rho(\Lambda(u)) = (s_1 \dots s_n, s)$, and for every i, $1 \le i \le n$, $u[i] R_{s_i} v[i]$, then $u R_s v$.

(3) For every pair (u, v) of nodes in TERM(H)^2 corresponding to a node $u \doteq_s v$ in the graph GC(H):
   (i) If $u \doteq_s v \in H$, then $u R_s v$.
   (ii) If $u \doteq_s v$ is the head of a clause $u \doteq_s v :- u_1 \doteq_{s_1} v_1, \dots, u_n \doteq_{s_n} v_n$ in H, and for every i, $1 \le i \le n$, $u_i R_{s_i} v_i$, then $u R_s v$.

In particular, note that any two nodes such that $u \doteq_s v$ is a clause are congruent.

3.5. *A Method for Testing Unsatisfiability*

The key to the method is that the least congruence on GT(H) containing E exists, and that there is an algorithm for computing it. Indeed, assume that this least congruence $\overset{*}{\leftrightarrow}_E$ containing E (called the *congruence closure of* E) exists and has been computed. Then the following result holds.

Theorem 3.6 (Soundness and completeness). Let H be a set of ground Horn clauses (with equality), let $E_s = \{(r, t) | r \doteq_s t \in H\}$, and let E be the $S(H)$-indexed family $(E_s)_{s \in S(H)}$. If $\overset{*}{\leftrightarrow}_E$ is the congruence closure on GT(H) *of E, then H is unsatisfiable iff for some clause $:- u_1 \doteq_{s_1} v_1, \dots, u_n \doteq_{s_n} v_n$ in H,*

$$\text{for every } i, \quad 1 \le i \le n, \qquad \text{we have} \quad u_i \overset{*}{\leftrightarrow}_E v_i.$$

PROOF. The proof is obtained by combining and generalizing the techniques used in Lemmas 10.6.2 and 10.6.4 of Gallier [15] (with some corrections). Let $\mathcal{D}$ be the subset of H consisting of the set of definite clauses in H. Let $\mathcal{E} = \{r \doteq_s t | (r, t) \in E_s, s \in S(H)\}$. Note that $\mathcal{E} \subseteq \mathcal{D}$.

First, we prove that the $S(H)$-indexed family R of relations R_s on TERM(H) defined such that

$$t R_s u \quad \text{iff} \quad \mathcal{D} \models t \doteq_s u$$

is a congruence on GT(H) containing E. Since $\mathcal{E} \subseteq \mathcal{D}$, it is obvious that $\mathcal{D} \models r \doteq_s t$ for every $(r, t) \in E_s$, and so $r R_s t$. Hence, R contains E. Clearly, each R_s is an equivalence relation. For every two subterms of the form $fy_1 \dots y_k$ and $fz_1 \dots z_k$ such that f is of rank $(w_1 \dots w_k, s)$, with $s \neq bool$, if for every i, $1 \le i \le k$,

$$\mathcal{D} \models y_i \doteq_{w_i} z_i,$$

then by the definition of the semantics of equality symbols,

$$\mathscr{D} \models fy_1 \dots y_k \doteq_s fz_1 \dots z_k.$$

For every two subterms of the form $Py_1 \dots y_k$ and $Pz_1 \dots z_k$ such that P is of rank $(w_1 \dots w_k, bool)$, if for every i, $1 \le i \le k$,

$$\mathscr{D} \models y_i \doteq_{w_i} z_i,$$

then by the definition of the semantics of equality symbols,

$$\mathscr{D} \models Py_1 \dots y_k \equiv Pz_1 \dots z_k.$$

For every clause $u \doteq_s v$ in $\mathscr{D}$, by the definitions of E_s and $\mathscr{E}$, we have $u \doteq_s v \in \mathscr{E}$, and since $\mathscr{E} \subseteq \mathscr{D}$, we have

$$\mathscr{D} \models u \doteq_s v.$$

For every clause $u \doteq_s v :- u_1 \doteq_{s_1} v_1, \dots, u_n \doteq_{s_n} v_n$ in $\mathscr{D}$, if for every i, $1 \le i \le n$, we have $\mathscr{D} \models u_i \doteq_{s_i} v_i$, then $\mathscr{D} \models u \doteq_s v$. Hence, R is a congruence on $\mathrm{GT}(H)$ containing E. Since $\overset{*}{\leftrightarrow}_E$ is the least congruence on $\mathrm{GT}(H)$ containing E, for any terms $r, t \in \mathrm{TERM}(H)_s$,

$$\text{if} \quad r \overset{*}{\leftrightarrow}_E t \quad \text{then} \quad \mathscr{D} \models r \doteq_s t.$$

Then, if for some negative clause $:- u_1 \doteq_{s_1} v_1, \dots, u_n \doteq_{s_n} v_n$ in H we have $u_i \overset{*}{\leftrightarrow}_E v_i$ for every i, $1 \le i \le n$, then $\mathscr{D} \models u_1 \doteq_{s_1} v_1 \wedge \cdots \wedge u_n \doteq_{s_n} v_n$ holds, which implies that the set $\mathscr{D} \cup \{ :- u_1 \doteq_{s_1} v_1, \dots, u_n \doteq_{s_n} v_n \}$ is unsatisfiable. Consequently, H is unsatisfiable.

Conversely, assume that there is no negative clause $:- u_1 \doteq_{s_1} v_1, \dots, u_n \doteq_{s_n} v_n$ in H such that $u_i \overset{*}{\leftrightarrow}_E v_i$ for every i, $1 \le i \le n$. We shall construct a model $\mathbf{M}$ of H.

First, we make the $S(H)$-indexed family $\mathrm{TERM}(H)$ into a many-sorted Σ-algebra $\mathbf{H}$. The difficulty involved in choosing the right algebra structure is that $\overset{*}{\leftrightarrow}_E$ must be a congruence on this algebra. This is not obvious, because $\mathrm{TERM}(H)$ is not closed under the term constructors, that is, for some terms $t_1, \dots, t_n \in \mathrm{TERM}(H)$ and some function symbols f, $ft_1 \dots t_n \notin \mathrm{TERM}(H)$. Hence, we have to be careful in defining the term value of $f_{\mathbf{H}}(t_1, \dots, t_n)$. If there exist other terms $r_1, \dots, r_n \in \mathrm{TERM}(H)$ such that $fr_1 \dots r_n \in \mathrm{TERM}(H)$, and $t_i \overset{*}{\leftrightarrow}_E r_i$ for every i, $1 \le i \le n$, then the value of $f_{\mathbf{H}}(t_1, \dots, t_n)$ cannot be defined arbitrarily. If we want $\overset{*}{\leftrightarrow}_E$ to be a congruence on $\mathbf{H}$, we must define $f_{\mathbf{H}}(t_1, \dots, t_n)$ so that $f_{\mathbf{H}}(t_1, \dots, t_n) \overset{*}{\leftrightarrow}_E f_{\mathbf{H}}(r_1, \dots, r_n)$. The same difficulty exists for predicate symbols. These difficulties are overcome in the following two definitions.

For each sort $s \neq bool$ in $S(H)$, each constant t of sort s is interpreted as the term t itself. For every function symbol f in Σ of rank $(w_1 \dots w_k, s)$, with $s \neq bool$, for every k terms $y_1, \dots, y_k$ in $\mathrm{TERM}(H)$, each y_i being of sort w_i, $1 \le i \le k$,

$$f_{\mathbf{H}}(y_1, \dots, y_k) = \begin{cases} fy_1 \dots y_k & \text{if } fy_1 \dots y_k \in \mathrm{TERM}(H)_s, \\ fz_1 \dots z_k & \text{if } fy_1 \dots y_k \notin \mathrm{TERM}(H)_s \text{ and there are terms} \\ & \quad z_1, \dots, z_k \text{ such that } y_1 \overset{*}{\leftrightarrow}_E z_i \text{ and} \\ & \quad fz_1 \dots z_k \in \mathrm{TERM}(H)_s, \\ t_0 & \text{otherwise, where } t_0 \text{ is some arbitrary term} \\ & \quad \text{chosen in } \mathrm{TERM}(H)_s. \end{cases}$$

For every predicate symbol P of rank $(w_1 \dots w_k, \mathit{bool})$, for every k terms $y_1, \dots, y_k \in \text{TERM}(H)$, each y_i being of sort w_i, $1 \le i \le k$,

$$P_{\mathbf{H}}(y_1, \dots, y_k) = \begin{cases} \mathbf{T} & \text{if } Py_1 \dots y_k \in \text{TERM}(H) \text{ and } Py_1 \dots y_k \overset{\cdot}{\leftrightarrow}_E \top, \\ \mathbf{T} & \text{if } Py_1 \dots y_k \notin \text{TERM}(H) \text{ and there are terms } z_1, \dots, z_k \\ & \quad \text{such that } y_i \overset{\cdot}{\leftrightarrow}_E z_i,\ Pz_1 \dots z_k \in \text{TERM}(H) \\ & \quad \text{and } Pz_1 \dots z_k \overset{\cdot}{\leftrightarrow}_E \top, \\ \mathbf{F} & \text{otherwise.} \end{cases}$$

Next, we prove that $\overset{\cdot}{\leftrightarrow}_E$ is an algebra congruence on $\mathbf{H}$. There are two main cases:

Case 1: For every function symbol f in Σ of rank $(w_1 \dots w_k, s)$ with $s \neq \mathit{bool}$, for every k pairs of terms $(y_1, z_1), \dots, (y_k, z_k)$ with y_i, z_i of sort w_i, $1 \le i \le k$, if $y_i \overset{\cdot}{\leftrightarrow}_E z_i$, then:

(i) If $fy_1 \dots y_k$ and $fz_1 \dots z_k$ are both in $\text{TERM}(H)$, then

$$f_{\mathbf{H}}(y_1, \dots, y_k) = fy_1 \dots y_k \quad \text{and} \quad f_{\mathbf{H}}(z_1, \dots, z_k) = fz_1 \dots z_k,$$

and since $\overset{\cdot}{\leftrightarrow}_E$ is a congruence on $\text{GT}(H)$, we have $fy_1 \dots y_k \overset{\cdot}{\leftrightarrow}_E fz_1 \dots z_k$. Hence, $f_{\mathbf{H}}(y_1, \dots, y_k) \overset{\cdot}{\leftrightarrow}_E f_{\mathbf{H}}(z_1, \dots, z_k)$.

(ii) $fy_1 \dots y_k \notin \text{TERM}(H)$, or $fz_1 \dots z_k \notin \text{TERM}(H)$, but there are some terms $z_1', \dots, z_k' \in \text{TERM}(H)$ such that $y_i \overset{\cdot}{\leftrightarrow}_E z_i'$ and $fz_1' \dots z_k' \in \text{TERM}(H)$. Since $y_i \overset{\cdot}{\leftrightarrow}_E z_i$, there are also terms $z_1'', \dots, z_k'' \in \text{TERM}(H)$ such that $z_i \overset{\cdot}{\leftrightarrow}_E z_i''$ and $fz_1'' \dots z_k'' \in \text{TERM}(H)$. Then,

$$f_{\mathbf{H}}(y_1, \dots, y_k) = fz_1' \dots z_k' \quad \text{and} \quad f_{\mathbf{H}}(z_1, \dots, z_k) = fz_1'' \dots z_k''.$$

Since $y_i \overset{\cdot}{\leftrightarrow}_E z_i$, we have, $z_i' \overset{\cdot}{\leftrightarrow}_E z_i''$, and so $fz_1' \dots z_k' \overset{\cdot}{\leftrightarrow}_E fz_1'' \dots z_k''$, that is,

$$f_{\mathbf{H}}(y_1, \dots, y_k) \overset{\cdot}{\leftrightarrow}_E f_{\mathbf{H}}(z_1, \dots, z_k).$$

(iii) If neither $fy_1 \dots y_k$ nor $fz_1 \dots z_k$ is in $\text{TERM}(H)$ and (ii) does not hold, then

$$f_{\mathbf{H}}(y_1, \dots, y_k) = f_{\mathbf{H}}(z_1, \dots, z_k) = t_0$$

for some chosen term t_0 in $\text{TERM}(H)$, and we conclude using the reflexivity of $\overset{\cdot}{\leftrightarrow}_E$.

Case 2: For every predicate symbol P of rank $(w_1 \dots w_k, \mathit{bool})$, for every k pairs of terms $(y_1, z_1), \dots, (y_k, z_k)$, with y_i, z_i of sort w_i, $1 \le i \le k$, if $y_i \overset{\cdot}{\leftrightarrow}_E z_i$, then:

(i) $Py_1 \dots y_k \in \text{TERM}(H)$ and $Pz_1 \dots z_k \in \text{TERM}(H)$. Since $y_i \overset{\cdot}{\leftrightarrow}_E z_i$ and $\overset{\cdot}{\leftrightarrow}_E$ is a graph congruence, $Py_1 \dots y_k \overset{\cdot}{\leftrightarrow}_E Pz_1 \dots z_k$. Hence, $Py_1 \dots y_k \overset{\cdot}{\leftrightarrow}_E \top$ iff $Pz_1 \dots z_k \overset{\cdot}{\leftrightarrow}_E \top$, that is, $P_{\mathbf{H}}(y_1, \dots, y_k) = \mathbf{T}$ iff $P_{\mathbf{H}}(z_1, \dots, z_k) = \mathbf{T}$.

(ii) $Py_1 \dots y_k \notin \text{TERM}(H)$ or $Pz_1 \dots z_k \notin \text{TERM}(H)$. In this case, $P_{\mathbf{H}}(y_1, \dots, y_k) = \mathbf{T}$ implies that there are terms $z_1' \dots z_k' \in \text{TERM}(H)$ such that $y_i \overset{\cdot}{\leftrightarrow}_E z_i'$, $Pz_1' \dots z_k' \in \text{TERM}(H)$, and $Pz_1' \dots z_k' \overset{\cdot}{\leftrightarrow}_E \top$. Since $y_i \overset{\cdot}{\leftrightarrow}_E z_i$, we also have $z_i \overset{\cdot}{\leftrightarrow}_E z_i'$. Since $Pz_1' \dots z_k' \in \text{TERM}(H)$, and $Pz_1' \dots z_k \overset{\cdot}{\leftrightarrow}_E \top$, we have, $P_{\mathbf{H}}(z_1, \dots, z_k) = \mathbf{T}$. The same argument shows that if $P_{\mathbf{H}}(z_1, \dots, z_k) = \mathbf{T}$, then $P_{\mathbf{H}}(y_1, \dots, y_k) = \mathbf{T}$. Hence, we have shown that $P_{\mathbf{H}}(y_1, \dots, y_k) = \mathbf{T}$ iff $P_{\mathbf{H}}(z_1, \dots, z_k) = \mathbf{T}$.

This concludes the proof that $\overset{*}{\leftrightarrow}_E$ is a congruence. □

Let **M** be the quotient of the algebra **H** by the congruence $\overset{*}{\leftrightarrow}_E$. We claim that for every term $t \in \text{TERM}(H)$, $t_{\mathbf{M}} = [t]$, the congruence class of t. This is easily shown by induction and is left as an exercise. By the definition of **M** as the quotient of **H** by $\overset{*}{\leftrightarrow}_E$, we also have the following property: For any two terms $u, v \in \text{TERM}(H)_s$,

$$\mathbf{M} \models u \doteq_s v \quad \text{iff} \quad u \overset{*}{\leftrightarrow}_E v. \tag{*}$$

We now prove that **M** is a model of H.

For every clause $u \doteq_s v \in H$, we have $(u, v) \in E_s$, and since $\overset{*}{\leftrightarrow}_E$ is a congruence containing E, we have $u \overset{*}{\leftrightarrow}_E v$. But then, by (*), we have $\mathbf{M} \models u \doteq_s v$.

For every clause $u \doteq_s v :- u_1 \doteq_{s_1} v_1, \ldots, u_n \doteq_{s_n} v_n$ in H, if $\mathbf{M} \models u_i \doteq_{s_i} v_i$ for every i, $1 \le i \le n$, by (*), we have $u_i \overset{*}{\leftrightarrow}_E v_i$ for every i, $1 \le i \le n$. Since $\overset{*}{\leftrightarrow}_E$ is a congruence on GT(H), we have $u \overset{*}{\leftrightarrow}_E v$. By (*), this is equivalent to $\mathbf{M} \models u \doteq_s v$. Hence,

$$\mathbf{M} \models u \doteq_s v :- u_1 \doteq_{s_1} v_1, \ldots, u_n \doteq_{s_n} v_n.$$

Finally, given any negative clause $:- u_1 \doteq_{s_1} v_1, \ldots, u_n \doteq_{s_n} v_n$ in H, recall that it is assumed that we cannot have $u_i \overset{*}{\leftrightarrow}_E v_i$ for every i, $1 \le i \le n$. Then, for some i, $1 \le i \le n$, u_i and v_i are not congruent modulo $\overset{*}{\leftrightarrow}_E$, and by (*), this implies that $\mathbf{M} \not\models u_i \doteq_{s_i} v_i$, that is, $\mathbf{M} \models \neg u_i \doteq_{s_i} v_i$. But this implies that

$$\mathbf{M} \models :- u_1 \doteq_{s_1} v_1, \ldots, u_n \doteq_{s_n} v_n.$$

Hence, **M** is a model of every clause in H. This concludes the proof. □

It is interesting to note that the soundness part of Theorem 3.6 follows from the fact that $\overset{*}{\leftrightarrow}_E$ is the *least* congruence on GT(H) containing E, and that the completeness part follows from the fact that $\overset{*}{\leftrightarrow}_E$ is a graph congruence. It only remains to prove that $\overset{*}{\leftrightarrow}_E$ exists.

4. EXISTENCE OF THE CONGRUENCE CLOSURE

We now prove that the congruence closure of a relation R on the graph GT(H) exists. This can be done by interleaving steps in which a purely equational congruence closure is computed, and steps in which a purely implicational kind of closure is computed. The advantage of this method (even though it is not the most direct) is that it can be used for showing the completeness of an extension of SLD resolution.

First, we define the concept of equational congruence closure.

4.1. Equational Congruence Closure

The notion of equational congruence closure was first introduced (under a different name) by Kozen [33, 34]. In fact, Kozen appears to have given an $O(n^2)$-time algorithm solving the word problem for finitely presented algebras before everybody else [33]. Independently, the concept of congruence closure was defined in Nelson and Oppen [42]. We have added the qualifier *equational* in order to distinguish it from the more general notion defined in Section 3.4 that applies to Horn clauses.

For our purpose, we only need to consider the concept of equational closure on the graph GT(H) induced by some (fixed) set H of ground Horn clauses. In the rest of this section, it is assumed that a fixed set H of ground Horn clauses is given.

Definition 4.1. An $S(H)$-indexed family R of relations R_s over $\text{TERM}(H)_s$ is an *equational congruence on* GT(H) iff:

(1) each R_s is an equivalence relation;

(2) for every pair $(u, v) \in \text{TERM}(H)^2$, if $\Lambda(u) = \Lambda(u)$, $\rho(\Lambda(u)) = (s_1 \dots s_n, s)$, and for every i, $1 \le i \le n$, $u[i]\, R_{s_i}\, v[i]$, then $u R_s v$.

The following lemma was first shown by Kozen [33, 34]. For the sake of completeness we present the proof given in Gallier [15].

Lemma 4.2. Given any $S(H)$-indexed family R of relations on $\text{TERM}(H)$, *there is a smallest equational congruence* $\stackrel{*}{=}_R$ *on the graph* GT(H) *containing* R.

PROOF. We define the sequence R^i of $S(H)$-indexed families of relations inductively as follows: For every sort $s \in S(H)$, for every $i \ge 0$,

$$R_s^0 = R_s \cup \{(u, u) \mid u \in \text{TERM}(H)_s\},$$

$$\begin{aligned} R_s^{i+1} = R_s^i \cup \{(v, u) \in \text{TERM}(H)^2 \mid (u, v) \in R_s^i\} \\ \cup \{(u, w) \in \text{TERM}(H)^2 \mid \exists v \in \text{TERM}(H), (u, v) \in R_s^i \text{ and } (v, w) \in R_s^i\} \\ \cup \{(u, v) \in \text{TERM}(H)^2 \mid \Lambda(u) = \Lambda(v), \rho(\Lambda(u)) = (s_1 \dots s_n, s), \\ \text{and } u[j]\, R_{s_j}^i\, v[j], 1 \le j \le n\}. \end{aligned}$$

Let $(\stackrel{*}{=}_R)_s = \bigcup_{i \ge 0} R_s^i$. It is easily shown by induction that every equational congruence on GT(H) containing R contains every R^i, and that $\stackrel{*}{=}_R$ is an equational congruence on GT(H). Hence, $\stackrel{*}{=}_R$ is the least equational congruence on GT(H) containing R. □

Since the graph GT(H) is finite, there must exist some integer i such that $R^i = R^{i+1}$. Hence, the equational congruence closure $\stackrel{*}{=}_R$ of R is computable.

We now define the concept of implicational closure.

4.2. *Implicational Closure*

Let H be a set of equational ground Horn clauses.

Definition 4.3. An $S(H)$-indexed family R of relations R_s over $\text{TERM}(H)_s$ is an *implicational relation* on GT(H) iff for every pair (u, v) of nodes in $\text{TERM}(H)^2$ corresponding to a node $u \doteq_s v$ in the graph GC(H):

(1) If $u \doteq_s v \in H$, then $u R_s v$.

(2) If $u \doteq_s v$ is the head of a clause $u \doteq_s v :- u_1 \doteq_{s_1} v_1, \dots, u_n \doteq_{s_n} v_n$ in H, and for every i, $1 \le i \le n$, $u_i R_{s_i} v_i$, then $u R_s v$.

The following result holds.

Lemma 4.4. Given a set H of equational ground Horn clauses, given any $S(H)$-indexed family R of relations on TERM(H), *there is a smallest implicational relation $\overset{*}{\supset}_R$ on the graph* GT(H) *containing R. The relation $\overset{*}{\supset}_R$ is called the implicational closure of R on* GT(H).

PROOF. We define the sequence R^i of $S(H)$-indexed families of relations inductively as follows: For every sort $s \in S(H)$, for every $i \geq 0$,

$$R_s^0 = R_s \cup \left\{(u, v) \in \text{TERM}(H)^2 \middle| u \doteq_s v \in H\right\},$$

$$R_s^{i+1} = R_s^i \cup \left\{(u, v) \in \text{TERM}(H)^2 \middle| u \doteq_s v \text{ is a node in GC}(H),\right.$$

and there is some clause

$u \doteq_s v :- u_1 \doteq_{s_1} v_1, \ldots, u_n \doteq_{s_n} v_n$ in H

such that $u_j R_{s_j}^i v_j$, $1 \leq j \leq n$\}.

Let $(\overset{*}{\supset}_R)_s = \bigcup_{i \geq 0} R_s^i$. As in the previous proof, it is easily shown that $\overset{*}{\supset}_R$ is the implicational closure of R.

Since GT(H) is finite, there is a least integer i such that $R^i = R^{i+1}$. Hence, the implicational closure $\overset{*}{\supset}_R$ of R is computable.

Note that $\overset{*}{\supset}_R$ is not necessarily an equivalence relation, but this does not matter, because we are going to interleave implicational closure steps and equational congruence closure steps.

4.3. *Congruence Closure for Horn Clauses*

The idea is to interleave steps in which the implicational closure is computed, and steps in which the equational congruence closure is computed.

Theorem 4.5. Given a set H of equational ground Horn clauses, and given any $S(H)$-indexed family R of relations on TERM(H), *there is a smallest congruence closure $\overset{*}{\leftrightarrow}_R$ on the graph* GT(H) *containing R.*

PROOF. We define the sequence R^i of $S(H)$-indexed families of relations inductively as follows: For every sort $s \in S(H)$, for every $j \geq 0$,

$$R_s^0 = R_s,$$

$$R_s^{2j+1} = \overset{*}{\supset}_{R_s^{2j}},$$

$$R_s^{2j+2} = \overset{*}{\cong}_{R_s^{2j+1}}.$$

Let $(\overset{*}{\leftrightarrow}_R)_s = \bigcup_{i \geq 0} R_s^i$.

Since the graph GT(H) is finite, there is some integer $i \geq 2$ such that $R^i = R^{i+1}$. If $i = 2j$, since $R_s^{2j+1} = \overset{*}{\supset}_{R_s^{2j}}$ and $j \geq 1$, then R_s^{2j} is an equational congruence, and R_s^{2j+1} is a congruence on GT(H). If $i = 2j+1$, since $R_s^{2j+2} = \overset{*}{\cong}_{R_s^{2j+1}}$ and $j \geq 1$, then R_s^{2j+1} is an implicational relation, and R_s^{2j+2} is a congruence on GT(H). It can also easily be shown by induction that any congruence on GT(H) containing R contains every R^i. Hence, $\overset{*}{\leftrightarrow}_R$ is the congruence of closure R on GT(H). □

The above theorem gives a method for computing $\overset{*}{\leftrightarrow}_R$. This method is not efficient, but it is possible to give fast algorithms based on the equational-congruence closure algorithm for ground equations [8, 15, 33, 34, 42, 43] and Dowling and Gallier's algorithm [7] for computing an implicational closure. Such algorithms are presented in Gallier [14].

5. A REFUTATION SYSTEM FOR GROUND EQUATIONAL HORN CLAUSES

We now show how the results of Sections 3 and 4 can be used to prove the completeness of a refutation system for ground equational Horn clauses. First, as in Kozen [33, 34], we show that equational congruence can be expressed using the notion of term rewriting. Technically, this is an important step, because the notion of term rewriting can be generalized to nonground terms, whereas it is not known how to generalize the congruence-closure concept to nonground terms. Hence, the reader should not be too surprised if the concept of congruence closure is not used in the rest of this paper, but rather the concept of term rewriting. The main role of the congruence closure concept is to establish the decidability of unsatisfiability for ground Horn clauses, and it also plays a crucial role in the proof of Theorem 5.7.

Definition 5.1. Let E be a finite set of equations. We define the relation $\Rightarrow_E$ on the set of terms as follows. Let t_1, t_2 be any two terms; then $t_1 \Rightarrow_E t_2$ iff there is some equation $s \doteq t \in E$, some trees address α in t_1, and some substitution σ such that if t_1/α denotes the subterm of t_1 rooted at α, we have

$$t_1/\alpha = \sigma(s) \quad \text{and} \quad t_2 = t_1[\alpha \leftarrow \sigma(t)].$$

When $t_1 \Rightarrow_E t_2$, we say that t_1 *rewrites to* t_2. In words, t_1 rewrites to t_2 iff t_2 is obtained from t_1 by finding a subterm of t_1 which is equal to a substitution instance $\sigma(s)$ of the left-hand side s of some equation $s \doteq t \in E$, and replacing it by the substitution instance $\sigma(t)$ of the right-hand side t of the equation.
Let $\overset{*}{\Rightarrow}_E$ be the reflexive and transitive closure of $\Rightarrow_E$. The relation $\Leftrightarrow_E$ is defined as follows: for every pair of terms s, t,

$$s \Leftrightarrow_E t \quad \text{iff} \quad s \Rightarrow_E t \text{ or } t \Rightarrow_E s.$$

Let $\overset{*}{\Leftrightarrow}_E$ be the transitive and reflexive closure of $\Leftrightarrow_E$. When we want to fully specify a rewrite step, we use the notation $t_1 \Rightarrow_{[\alpha, s \doteq t, \sigma]} t_2$, or more simply $t_1 \Rightarrow_{[s \doteq t, \sigma]} t_2$, and similarly for $\Leftrightarrow$. When E consists of ground equations and the terms s and t are ground, the substitution σ is the identity and is omitted.

The following easy lemma will be needed in the next section.

Lemma 5.2. Let E be a finite set of equations, and s and t two ground terms. Then $s \overset{*}{\Leftrightarrow}_E t$ iff there is a finite set E' of ground instances of equations in E such that $s \overset{*}{\Leftrightarrow}_{E'} t$.

PROOF. Both directions are easy inductions on the number of rewrite steps. The details are left to the reader. □

The following lemma, analogous to Kozen's first theorem [33, p. 8] formalizes the equivalence of the congruence-closure method and ground-term rewriting.

Lemma 5.3. Let E be a finite set of ground equations, and $s \doteq t$ be any arbitrary ground equation. Let H be the set of Horn clauses $E \cup \{ :- s \doteq t \}$, and let $\dot{\cong}_E$ be the equational-congruence closure of E on $\mathrm{GT}(H)$. Then,

$$s \dot{\cong}_E t \quad \text{iff} \quad s \dot{\leftrightarrow}_E^* t.$$

PROOF. The proof proceeds by induction on the number of rewrite steps and on the number of congruence-closure steps. The details are straightforward and are left to the reader. □

We are now in a position to define our refutation system. This system is an extension of SLD resolution in which both standard SLD resolution steps and *conditional rewrite steps* are performed.

Definition 5.4. Let H be a set of ground Horn clauses with equality consisting of a set D of definite clauses and a set $\{G_1, \ldots, G_q\}$ of goals. A *ground SLDE derivation* for H is a sequence $\langle N_0, N_1, \ldots, N_p \rangle$ of negative clauses satisfying the following properties:

(1) $N_0 = G_j$, where G_j is one of the goals;

(2) For every N_i, $0 \leq i < p$, if $N_i = :- A_1, \ldots, A_{k-1} A_k, A_{k+1}, \ldots, A_n$ (where A_k is of any sort, including *bool*), then either

(i) there is some definite clause $C_i = A_k :- B_1, \ldots, B_m$ in D such that, if $m > 0$, then

$$N_{i+1} = :- A_1, \ldots, A_{k-1}, B_1, \ldots, B_m, A_{k+1}, \ldots, A_n,$$

else if $m = 0$ then

$$N_{i+1} = :- A_1, \ldots, A_{k-1}, A_{k+1}, \ldots, A_n;$$

or

(ii) Letting $A_k = s \doteq t$ (where $s \doteq t$ is of any sort, including *bool*), there is some finite set

$$\mathscr{D} = \{ s_1 \doteq t_1 :- \Gamma_1, \ldots, s_m \doteq t_m :- \Gamma_m \}$$

of definite clauses in D, such that, letting $\mathscr{E} = \{ s_1 \doteq t_1, \ldots, s_m \doteq t_m \}$, we have

$$s \dot{\leftrightarrow}_{\mathscr{E}}^* t \quad \text{and} \quad N_{i+1} = :- A_1, \ldots, A_{k-1}, \Gamma_1, \ldots, \Gamma_m, A_{k+1}, \ldots, A_n$$

(where any of the Γ_j's may be empty).

A ground SLDE derivation is a *ground SLDE-refutation* iff $N_p = \square$ (the empty clause).

A step as in (i) is called an *SLD step*, and a step as in (ii) is called a *conditional rewrite step*. It is in step (ii) that A_k is treated as an equation, possibly of sort *bool*. Note that cases (i) and (ii) are not mutually exclusive, that is, one may have the choice of applying a standard SLD step or a conditional rewrite step to the same

atom $A_k = s \doteq t$. Also, if H does not have any equations, then a ground SLDE derivation is an SLD refutation, and if all definite clauses in H are equations, a ground SLDE refutation consists of rewrite steps only. Such a refutation method is a form of linear input refutation, because it resolves the current clause N_k with some clause(s) in the input set D. Our method can be viewed as a special version of the paramodulation method of Robinson and Wos [49], or of the E-resolution method of Anderson [1]. The crucial difference is that many paramodulation steps can be performed in one SLDE step. Before showing the completeness of this method, we give an extended example.

Example 5.5. Consider the following set H of ground equational Horn clauses:

$$
\begin{aligned}
& f^3a \doteq \ :- fa \doteq fb, \\
& a \doteq b, \\
& \quad Pa, \\
& f^5a \doteq a :- Qa, \\
& \quad Qa :- f^3a \doteq a, \\
& \quad Ra :- fa \doteq a, Pfa, \\
& \quad\quad :- Rfa.
\end{aligned}
$$

Recall that Pa, Qa, Ra, Pfa, and Rfa are abbreviations for $Pa \doteq \top$, $Qa \doteq \top$, $Ra \doteq \top$, $Pfa \doteq \top$, and $Rfa \doteq \top$. Since there is no definite clause of the form $Rfa \doteq \top :- B_1, \ldots, B_m$ in H, an SLD step is not applicable. However, if we let

$$
\begin{aligned}
D_1 = \{ & Ra \doteq \top :- fa \doteq a, Pfa, \\
& f^3a \doteq a :- fa \doteq fb, \\
& f^5a \doteq a :- Qa \}
\end{aligned}
$$

and thus $\mathscr{E}_1 = \{ Ra \doteq \top, f^3a \doteq a, f^5a \doteq a \}$, a conditional rewrite step is applicable, and Rfa can be rewritten to $\top$ by the following sequence of rewrite steps:

$$\top \overset{\cdot}{\leftrightarrow}_{\mathscr{E}_1} Ra \overset{\cdot}{\leftrightarrow}_{\mathscr{E}_1} Rf^3a \overset{\cdot}{\leftrightarrow}_{\mathscr{E}_1} Rf^6a \overset{\cdot}{\leftrightarrow}_{\mathscr{E}_1} Rfa.$$

Note that the equations in $\mathscr{E}_1$ have been used as two-way rewrite rules (and that Rfa is an abbreviation for $Rfa \doteq \top$). Collecting the premises used in these steps, we have the following step in the derivation, where N_i stands for the ith step:

$$
\begin{aligned}
N_1 \quad & :- Rfa, \\
N_2 \quad & :- fa \doteq a, Pfa, fa \doteq fb, Qa.
\end{aligned}
$$

Let us choose $fa \doteq fb$ as the next current subgoal. Since there is no definite clause with $fa \doteq fb$ as its head, an SLD step is again not applicable. Letting $\mathscr{E}_2 = D_2 = \{ a \doteq b \}$ allows $fa \overset{\cdot}{\leftrightarrow}_{\mathscr{E}_2} fb$ immediately, and thus

$$N_3 \quad :- fa \doteq a, Pfa, Qa$$

Again, no SLD step applies. Letting

$$
\begin{aligned}
D_3 = \{ & f^3a \doteq a :- fa \doteq fb, \\
& f^5a \doteq a :- Qa \}
\end{aligned}
$$

and $\mathscr{E}_3 = \{ f^3a \doteq a, f^5a \doteq a \}$, rewriting, we have

$$a \overset{\cdot}{\Leftrightarrow}_{\mathscr{E}_3} f^3a \overset{\cdot}{\Leftrightarrow}_{\mathscr{E}_3} f^6a \overset{\cdot}{\Leftrightarrow}_{\mathscr{E}_3} fa$$

and thus

$$N_4 \quad :- Qa, fa \doteq fb, Pfa, Qa,$$

$$N_{4'} \quad :- Qa, fa \doteq fb, Pfa.$$

$N_{4'}$ is derived from N_4 by simplification, i.e., dropping the duplicate Qa. Similarly to the derivation of N_2 from N_1, $fa \doteq fb$ can be eliminated, yielding

$$N_5 \quad :- Qa, Pfa.$$

Continuing, by letting

$$D_4 = \{ f^3a \doteq a :- fa \doteq fb,$$
$$f^5a \doteq a :- Qa,$$
$$Pa \doteq \top \}$$

and $\mathscr{E}_4 = \{ f^3a \doteq a, f^5a \doteq a, Pa \doteq \top \}$, rewriting

$$\top \overset{\cdot}{\Leftrightarrow}_{\mathscr{E}_4} Pa \overset{\cdot}{\Leftrightarrow}_{\mathscr{E}_4} Pf^3a \overset{\cdot}{\Leftrightarrow}_{\mathscr{E}_4} Pf^6a \overset{\cdot}{\Leftrightarrow}_{\mathscr{E}_4} Pfa$$

yields

$$N_6 \quad :- Qa, fa \doteq fb.$$

Eliminating $fa \doteq fb$ is again immediate, and thus the next step in the derivation is

$$N_7 \quad :- Qa.$$

An SLD step is now applicable using the definite clause $Qa :- f^3a \doteq a$, and yields

$$N_8 \quad :- f^3a \doteq a.$$

Finally, using the clause $f^3a \doteq a :- fa \doteq fb$, the negative clause $:- fa \doteq fb$ is derived by another SLD step, which as above can be eliminated, yielding a refutation

$$N_9 :- fa \doteq fb,$$

$$N_{10} \quad \square.$$

The reader has probably noticed that there has been a tremendous amount of redundant computation in this refutation; in particular, the implication

$$f^3a \doteq a \wedge f^5a \doteq a \supset fa \doteq a$$

has occurred three times. This is an issue that would have to be addressed in an implementation.

In order to prove the completeness of the above method, we need the following lemma establishing the completeness of ground SLD refutations.

Lemma 5.6. Let $H = D \cup \{ :- u \doteq v \}$ be a set of ground Horn clauses, where D is a set of definite clauses with equality, let $E = \{(u', v') | u' \doteq v' \in D\}$, and let $\overset{\cdot}{\supset}_E$ be the implicational closure of E on $\mathrm{GC}(H)$. If $u \overset{\cdot}{\supset}_E v$, then there is an SLD refutation for the set $D \cup \{ :- u \doteq v \}$.

PROOF. By Lemma 4.4, for every sort s, $(\dot{\supset}_E)_s = \bigcup_{i \geq 0} R^i_s$, where the R^i_s are defined inductively. We proceed by induction on the least i such that $u R^i v$.

If $i = 0$, then $u \doteq v \in E$, and we have an obvious SLD refutation from $:- u \doteq v$ and $u \doteq v$.

If $i > 0$, then there is some definite clause $u \doteq v :- u_1 \doteq v_1, \ldots, u_n \doteq v_n$ in H such that, for every j, $1 \leq j \leq n$, we have $u_j R^{i-1} v_j$. By the induction hypothesis, there is an SLD refutation for each set $D \cup \{ :- u_j \doteq v_j \}$, $1 \leq j \leq n$. Using the definite clause $u \doteq v :- u_1 \doteq v_1, \ldots, u_n \doteq v_n$, it is obvious that an SLD refutation for the set $D \cup \{ :- u \doteq v \}$ can be constructed from these SLD refutations. □

We now prove the completeness of the ground SLDE-refutation method.

Theorem 5.7 (Completeness of ground SLDE refutations). Let H be a finite set of ground Horn clauses with equality, and let D be the set of definite clauses in H. If H is unsatisfiable, then H contains some negative clause $:- u_1 \doteq v_1, \ldots, u_m \doteq v_m$, and there is some ground SLDE refutation for the set $D \cup \{ :- u_1 \doteq v_1, \ldots, u_m \doteq v_m \}$.

PROOF. From the completeness theorem 3.6, H is unsatisfiable iff there is some negative clause $:- u_1 \doteq v_1, \ldots, u_m \doteq v_m$ in H such that, for every i, $1 \leq i \leq m$, we have $u_i \dot{\leftrightarrow}_E v_i$. Then, for every i, $1 \leq i \leq m$, the set $D \cup \{ :- u_1 \doteq v_i \}$ is unsatisfiable.

Let $E = \{(u', v') | u' \doteq v' \in H\}$. By Lemma 4.5, for every sort s, $(\dot{\leftrightarrow}_E)_s = \bigcup_{i \geq 0} R^i_s$, where

$$R^0_s = E_s,$$

$$R^{2j+1}_s = \dot{\supset}_{R^{2j}_s},$$

$$R^{2j+2}_s = \dot{\cong}_{R^{2j+1}_s}.$$

Since $u_i \dot{\leftrightarrow}_E v_i$, we prove that there is an SLDE refutation for the set $D \cup \{ :- u_i \doteq v_i \}$ by induction on the least k such that $u_i R^k v_i$.

Case 1: If $k = 0$, then $u_i \doteq v_i \in H$, and so we have an SLD-refutation for the set $\{ :- u_i \doteq v_i, u_i \doteq v_i \}$.

Case 2: If $k = 2j + 1$, then $R^{2j+1}_s = \dot{\supset}_{R^{2j}_s}$. By Lemma 5.6, there is an SLD-refutation $\mathscr{R}$ for the set $D \cup \{ u \doteq v | u R^{2j} v \} \cup \{ :- u_i \doteq v_i \}$. Let $\{ x_1 \doteq y_1, \ldots, x_r \doteq y_r \}$ be set of equations used in the SLD refutation $\mathscr{R}$. Since $x_l R^{2j} y_l$, by the induction hypothesis there is an SLDE-refutation for each set $D \cup \{ :- x_l \doteq y_l \}$, $1 \leq l \leq r$. By combining these refutations and the SLD-refutation $\mathscr{R}$, we obtain a SLDE-refutation for the set $D \cup \{ :- u_i \doteq v_i \}$.

Case 3: If $k = 2j + 2$, then $R^{2j+2}_s = \dot{\cong}_{R^{2j+1}_s}$. By Lemma 5.3, there is a sequence of rewrite steps $u_i \dot{\leftrightarrow}_{R^{2j+1}} v_i$. Let $\{ x_1 \doteq y_1, \ldots, x_r \doteq y_r \}$ be the set of equations used in the sequence of rewrite steps. Since $x_l R^{2j+1} y_l$, by the induction hypothesis there is an SLDE-refutation for each set $D \cup \{ :- x_l \doteq y_l \}$, $1 \leq l \leq r$. But since R^{2j+1} is an implicational closure, by case (2) this implies that for every $:- x_l \doteq y_l$, the first step of the SLDE refutation for the set $D \cup \{ :- x_l \doteq y_l \}$ uses some definite clause $x_l \doteq y_l :- \Gamma_l$ (where Γ_l may be empty). Hence, performing conditional rewriting steps using the set $\{ x_1 \doteq y_1 :- \Gamma_1, \ldots, x_l \doteq y_l :- \Gamma_l \}$ and the above SLDE refutations, it is possible to construct an SLDE-refutation for the set $D \cup \{ :- u_i \doteq v_i \}$, whose first

step is

$$:- u_i \doteq v_i,$$

$$:- \Gamma_1, \ldots, \Gamma_l.$$

From the SLDE refutations for the sets $D \cup \{ :- u_i \doteq v_i \}$, we obtain an SLDE refutation for the set $D \cup \{ :- u_1 \doteq v_1, \ldots, u_m \doteq v_m \}$. □

We leave the soundness of the ground SLDE-refutation method as an exercise.

6. A GENERAL REFUTATION METHOD USING *E*-UNIFICATION

In order to generalize the results of the previous section to the first-order case, we shall need the notion of unification modulo a set of equations. This concept comes up naturally when we apply the Herbrand-Skolem-Gödel theorem (see Gallier [15], or Shoenfield [51]) to a set of universal Horn clauses with equality, in order to reduce unsatisfiability in the first-order case to unsatisfiability in the ground case. Recall that this theorem states that a set of universal prenex sentences is unsatisfiable if and only if some set of ground substitution instances of the matrices of these sentences is unsatisfiable.

Applying the Herbrand-Skolem-Gödel theorem to a set H of first-order universal Horn clauses with equality, by Theorem 5.7, there is a ground SLDE refutation of a certain set H' of ground instances of the clauses in H. It is easily seen by induction on the number of refutation steps that every goal clause in this refutation is of the form $\sigma(:- A_1, \ldots, A_n)$ for some ground substitution σ and some goal clause $:- A_1, \ldots, A_n$ (*not* necessarily in H). Hence, every refutation step is in one of the following two forms:

(i) There is some goal $N_i = \sigma(:- A_1, \ldots, A_{k-1}, A_k, A_{k+1}, \ldots, A_n)$, and some definite clause $C_i = \theta(A :- B_1, \ldots, B_m)$, where $A :- B_1, \ldots, B_m$ is some definite clause in H and θ is some ground substitution, and

$$\sigma(A_k) = \theta(A). \qquad (*)$$

(ii) $N_i = \sigma(:- A_1, \ldots, A_{k-1}, A_k, A_{k+1}, \ldots, A_n)$, and letting $A_k = s \doteq t$, there is some finite set

$$\mathscr{D}' = \{ s_1' \doteq t_1' :- \Gamma_1', \ldots, s_m' \doteq t_m' :- \Gamma_m' \}$$

of ground definite clauses, some finite set $\mathscr{S} = \{\theta_1, \ldots, \theta_m\}$ of ground substitutions, and some finite set

$$\mathscr{D} = \{ s_i \doteq t_i :- \Gamma_1, \ldots, s_r \doteq t_r :- \Gamma_r \}$$

of clauses in H, such that, for every $s_j' \doteq t_j' :- \Gamma_j' \in \mathscr{D}'$, there is some substitution $\theta_j \in \mathscr{S}$ and some clause $s \doteq t :- \Gamma \in \mathscr{D}$ such that $s_j' \doteq t_j' :- \Gamma_j' = \theta_j(s \doteq t :- \Gamma)$, and, letting $\mathscr{E}' = \{ s_1' \doteq t_1', \ldots, s_m' \doteq t_m' \}$, we have $\sigma(s) \overset{\cdot}{\Leftrightarrow}_{\mathscr{E}'} \sigma(t)$. By Lemma 5.2, this is equivalent to

$$\sigma(s) \overset{\cdot}{\Leftrightarrow}_{\mathscr{E}} \sigma(t), \qquad (**)$$

where $\mathscr{E} = \{ s_i \doteq t_i, \ldots, s_r \doteq t_r \}$.

In case (*), A and A_k are unifiable, and since it can be assumed that A and A_k have no variables in common, it is well known that A and A_k have a most general unifier (see [15]). In case (**), we have a generalization of the concept of a unifier. We say that σ is a unifier of s and t *modulo the set of equations* $\mathscr{E}$. Unfortunately, unification modulo a set of equations does not enjoy the nice properties of standard unification. In particular, there is in general no algorithm for deciding whether two terms are unifiable modulo $\mathscr{E}$, and if two terms are unifiable modulo $\mathscr{E}$, there is in general no single most general unifier, but instead, a possibly infinite set. Nevertheless, there is always a procedure for enumerating a complete set of unifiers modulo $\mathscr{E}$. The problem is that such a procedure may generate a highly redundant set of $\mathscr{E}$-unifiers. First, we give a precise definition of a $\mathscr{E}$-unifier, and then we generalize Definition 5.4 to the first-order case. Some auxiliary definitions are also needed.

Definition 6.1. Given a Horn clause C, a Horn clause C' is a *variant* of C iff there is an injective substitution σ such that $\sigma(x)$ is a variable for every $x \in D(\sigma)$, $C' = \sigma(C)$, and $\mathrm{Var}(C) \cap \mathrm{Var}(\sigma(C)) = \varnothing$. We say that σ is a *renaming*.

Definition 6.2. Let E be a finite set of equations. Given two terms s and t, we say that a substitution σ is a *unifier of s and t modulo E*—for short, an *E-unifier of s and t*—iff $\sigma(s) \overset{\cdot}{\Leftrightarrow}_E \sigma(t)$.

Note that when $E = \varnothing$, an E-unifier is just a standard unifier. In the following definition, E-unification is integrated into SLD resolution.

Definition 6.3. Let H be a set of first-order Horn clauses with equality consisting of a set D of definite clauses and a set $\{G_1, \ldots, G_q\}$ of goals. An *SLDE-derivation* for H is a sequence $\langle N_0, N_1, \ldots, N_p \rangle$ of negative clauses satisfying the following properties:

(1) $N_0 = G_j$, where G_j is one of the goals.

(2) For every N_i, $0 \le i < p$, if $N_i = :- A_1, \ldots, A_{k-1}, A_k, A_{k+1}, \ldots, A_n$ (where A_k is of any sort, including *bool*), then either

(i) there is some definite clause $C_i = A :- B_1, \ldots, B_m$ in D and some most general unifier σ_i of A and A_k (assuming that the variables in C_i have been renamed so that they do not occur in N_i, which is always possible), such that, if $m > 0$, then

$$N_{i+1} = \sigma_i(:- A_1, \ldots, A_{k-1}, B_1, \ldots, B_m, A_{k+1}, \ldots, A_n),$$

else if $m = 0$ then

$$N_{i+1} = \sigma_i(:- A_1, \ldots, A_{k-1}, A_{k+1}, \ldots, A_n);$$

or

(ii) letting $A_k = s \doteq t$ (where $s \doteq t$ is of any sort, including *bool*), there is some finite set

$$\mathscr{D} = \{s_1 \doteq t_1 :- \Gamma_1, \ldots, s_m \doteq t_m :- \Gamma_m\}$$

of variants of definite clauses in D (such that any two clauses in this set have disjoint sets of variables, also disjoint from the set of variables in

N_i) and some finite set $\mathscr{S} = \{\theta_1, \ldots, \theta_m\}$ of substitutions, and, letting

$$\mathscr{E} = \{s_1 \doteq t_1, \ldots, s_m \doteq t_m\}$$

for some $\mathscr{E}$-unifier σ_i of s and t, we have

$$\sigma_i(s) \Leftrightarrow_{[s_1 \doteq t_1, \theta_1]} \cdots \Leftrightarrow_{[s_m \doteq t_m, \theta_m]} \sigma_i(t)$$

and

$$N_{i+1} = :- \sigma_i(A_1), \ldots, \sigma_i(A_{k-1}), \theta_1(\Gamma_1), \ldots, \theta_m(\Gamma_m), \sigma_i(A_{k+1}), \ldots, \sigma_i(A_n)$$

(where any of the Γ_j's may be empty).

An SLDE-derivation is an *SLDE refutation* iff $N_p = \square$ (the empty clause).

Again, note that cases (i) and (ii) are not mutually exclusive. Also, note what is involved in step (ii). It is necessary to find:

(1) some finite set $\mathscr{D} = \{s_1 \doteq t_1 :- \Gamma_1, \ldots, s_m \doteq t_m :- \Gamma_m\}$ of variants of clauses in D, and some finite set $\mathscr{S} = \{\theta_1, \ldots, \theta_m\}$ of substitutions;

(2) an $\mathscr{E}$-unifier σ_i of s and t such that

$$\sigma_i(s) \Leftrightarrow_{[s_1 \doteq t_1, \theta_1]} \cdots \Leftrightarrow_{[s_m \doteq t_m, \theta_m]} \sigma_i(t),$$

where $\mathscr{E} = \{s_1 \doteq t_1, \ldots, s_m \doteq t_m\}$.

The point is that it is not sufficient to simply find some set $\mathscr{D}$ and some $\mathscr{E}$-unifier σ_i of s and t. In order to identify the premises $\theta_1(\Gamma_1), \ldots, \theta_m(\Gamma_m)$, we also need to find the set $\mathscr{S} = \{\theta_1, \ldots, \theta_m\}$ of substitutions applied in some sequence of rewrite steps $\sigma_i(s) \overset{*}{\Leftrightarrow}_{\mathscr{E}} \sigma_i(t)$.

Hence, the method requires not just a procedure for enumerating E-unifiers, but also one for producing an explicit sequence of substitutions for every $\mathscr{E}$-unifier, which is prohibitive in practice. In the next section, we shall consider subcases for which it is not necessary to produce the sets $\mathscr{E}$. First, we give an example and establish the completeness of the above method.

Example 6.4. Consider the following set H of equational Horn clauses, where x, y, z denote variables:

$$f^3 y \doteq y :- fy \doteq fb,$$
$$a \doteq b,$$
$$Pa,$$
$$f^5 x \doteq x :- Qx,$$
$$Qa :- f^3 a \doteq a,$$
$$Ra :- fa \doteq a, Pfa,$$
$$:- Rfz, Pz.$$

Recall that Pa, Qx, Qa, Ra, Pfa, Rfz, and Pz are abbreviations for $Pa \doteq \top, Qx \doteq \top, Qa \doteq \top, Ra \doteq \top, Pfa \doteq \top, Rfz \doteq \top$, and $Pz \doteq \top$. Now $Rfz \doteq \top$ is not unifiable with

the head of any clause in H. However, if we let

$$D = \{\, Ra \doteq \top :- fa \doteq a,\ Pfa,$$
$$f^3y \doteq y :- fy \doteq fb,$$
$$f^5x \doteq x :- Qx \,\};$$

$\mathscr{S} = \{\theta_1 = \mathrm{Id}, \theta_2 = [a/y], \theta_3 = [a/x]\}$ where Id stands for the identity substitution; and $\mathscr{E} = \{Ra \doteq \top, f^3y \doteq y, f^5x \doteq x\}$, then a conditional rewriting step is applicable, since $\sigma_1 = [a/z]$ is an $\mathscr{E}$-unifier of $\top$ and Rfz by the following sequence of rewrite steps:

$$\top \Leftrightarrow_{[Ra \doteq \top, \theta_1]} Ra \Leftrightarrow_{[f^3y \doteq y, \theta_2]} Rf^3a \Leftrightarrow_{[f^3y \doteq y, \theta_2]} Rf^6a \Leftrightarrow_{[f^5x \doteq x, \theta_3]} Rfa.$$

Thus $\sigma_1(Rfz) \overset{\cdot}{\Leftrightarrow}_{\mathscr{E}} \sigma_1(\top)$. Again, as in the ground case, the equations have been used as two-way rewrite rules. Collecting the premises used in these steps, we have the following step in the derivation:

$$N_1 \quad :- Rfz, Pz$$
$$N_2 \quad :- \theta_1(fa \doteq a), \theta_1(Pfa), \theta_2(fy \doteq fb), \theta_3(Qx), \sigma_1(Pz).$$

This simplifies by substitution to

$$N_{2'} \quad :- fa \doteq a, Pfa, fa \doteq fb, Qa, Pa.$$

Since the derivation step illustrated shows case (ii) of the definition in its full generality, and case (i) is the standard SLD-resolution step, we will not complete the derivation beyond this point.

The completeness of the above method is now established.

Theorem 6.5 (Completeness of SLDE refutations). *Let H be a finite set of first-order Horn clauses with equality. If H is unsatisfiable, then there is an SLDE refutation for H.*

PROOF. As indicated at the beginning of Section 6, we apply the Herbrand-Skolem-Gödel theorem (see Gallier [15] or Shoenfield [51]) to H. According to this theorem, H is unsatisfiable iff some set H' of ground-substitution instances of the matrices of the clauses in H is unsatisfiable. Since H' is unsatisfiable by Theorem 5.7 there is some ground SLDE refutation for the set H' from some goal N_0' in H'. We prove the following claim.

Claim. For every ground SLDE derivation $\langle N_0', N_1', \ldots, N_p' \rangle$ for the set of ground instances H', there is an SLDE derivation $\langle N_0, N_1, \ldots, N_p \rangle$ for the set H, and some sequence $\langle \eta_0, \ldots, \eta_p \rangle$ of ground substitutions, such that, $N_j' = \eta_j(N_j)$ for every j, $0 \le j \le p$.

PROOF OF CLAIM. We proceed by induction on the length of derivations. The claim is trivial for $p = 0$. Next, we prove the claim for $p + 1$. In the rest of this proof, "derivation" will mean SLDE derivation (and similarly for refutation). By the induction hypothesis, there is a derivation $\langle N_0, \ldots, N_p \rangle$ satisfying the claim, and in particular, $N_p' = \eta_p(N_p)$, where N_p is some goal clause $:- A_1, \ldots, A_{k-1}, A_k, A_{k+1}, \ldots, A_n$. There are two cases:

Case 1: $N'_p = \eta_p(N_p)$ as above, $C_p = \theta(A :- B_1, \ldots, B_m)$ for some ground substitution θ and some definite clause $A :- B_1, \ldots, B_m$ in D, and

$$N'_{p+1} = :- \eta_p(A_1), \ldots, \eta_p(A_{k-1}), \theta(B_1), \ldots, \theta(B_m), \eta_p(A_{k+1}), \ldots, \eta_p(A_n),$$

where $\eta_p(A_k) = \theta(A)$. It can be assumed by renaming variables if necessary that η_p and θ have disjoint support. Then, we can let σ_{p+1} denote the union of η_p and θ, so that $N'_{p+1} = \sigma_{p+1}(:- A_1, \ldots, A_{k-1}, B_1, \ldots, B_m, A_{k+1}, \ldots, A_n)$ and $\sigma_{p+1}(A_k) = \sigma_{p+1}(A)$. If σ is a most general unifier of A and A_k, there is a substitution η_{p+1} such that $\sigma_{p+1} = \sigma \circ \eta_{p+1}$. Then, there is an SLD-resolution step from $N_p = :- A_1, \ldots, A_{k-1}, A_k, A_{k+1}, \ldots, A_n$ to $N_{p+1} = \sigma(:- A_1, \ldots, A_{k-1}, B_1, \ldots, B_m, A_{k+1}, \ldots, A_n)$ and $N'_{p+1} = \eta_{p+1}(N_{p+1})$. Hence, $\langle N_0, \ldots, N_{p+1} \rangle$ and $\langle \eta_0, \ldots, \eta_{p+1} \rangle$ also satisfy the claim.

Case 2: $N'_p = \eta_p(N_p)$ as above, and letting $A_k = s \doteq t$, there is some finite set

$$\mathscr{D} = \{ s_1 \doteq t_1 :- \Gamma_1, \ldots, s_m \doteq t_m :- \Gamma_m \}$$

of variants of definite clauses in D (such that any two clauses in this set have disjoint sets of variables, also disjoint from the set of variables in N_p), and some finite set $\mathscr{S} = \{\theta_1, \ldots, \theta_m\}$ of ground substitutions; and, letting

$$\mathscr{E} = \{ s_1 \doteq t_1, \ldots, s_m \doteq t_m \},$$

we have

$$\eta_p(s) \Leftrightarrow_{[s_1 \doteq t_1, \theta_1]} \cdots \Leftrightarrow_{[s_m \doteq t_m, \theta_m]} \eta_p(t),$$

and

$$N'_{p+1} = :- \eta_p(A_1), \ldots, \eta_p(A_{k-1}), \theta_1(\Gamma_1), \ldots, \theta_m(\Gamma_m), \eta_p(A_{k+1}), \ldots, \eta_p(A_n).$$

Then, note that η_p is a $\mathscr{E}$-unifier of s and t, and, letting $N_{p+1} = N'_{p+1}$ and $\eta_{p+1} = \mathrm{Id}$ (where Id denotes the identity substitution), $\langle N_0, \ldots, N_{p+1} \rangle$ and $\langle \eta_0, \ldots, \eta_{p+1} \rangle$ also satisfy the claim. This concludes the proof of the claim. □

Applying the claim to a refutation, the theorem is provided. □

The soundness of the SLDE-refutation method is left as an exercise.

7. A REFUTATION METHOD USING *E*-UNIFICATION PROCEDURES

In this section, we present a refinement of the SLDE-refutation method that uses an explicit procedure for enumerating a complete set of E-unifiers. Stimulated mostly by work on the Knuth-Bendix procedure, E-unification has been investigated extensively in the past few years. Siekmann [50] contains an excellent survey, and the state of the art in this domain is described in Kirchner [27]. The main problem is to generalize the concept of a most general unifier. To this effect, we need some definitions, most of which are taken from Kirchner and Kirchner [29]. First, we need to define when two substitutions are equal modulo E.

Definition 7.1. Given a finite set E of equations, and given any set W of variables, we say that two substitutions σ and θ are *equal modulo E over W*, denoted by $\sigma =_E \theta[W]$, iff for every variable $x \in W$, $\sigma(x) \overset{*}{\Leftrightarrow}_E \theta(x)$.

We say that σ *is more general than* θ *over* W, denoted by $\sigma \leq_E \theta[W]$, iff there is some substitution η such that $\theta =_E \sigma \circ \eta[W]$.

We say that σ and θ are *congruent modulo* E *over* W, denoted by $\sigma \equiv_E \theta[W]$, iff $\sigma \leq_E \theta[W]$ and $\theta \leq_E \sigma[W]$.

When either $\sigma \not\leq_E \theta[W]$ or $\theta \not\leq_E \sigma[W]$, we use the notation $\sigma \not\equiv_E \theta[W]$, and we say that σ and θ are *noncongruent modulo* E *over* W. When W is the set of all variables, it is omitted, and similarly when $E = \varnothing$.

Note that in general, $=_E$ and $\equiv_E$ are distinct relations, as shown by Fages and Huet [10]. In the next definition, the concept of a most general unifier is generalized to E-unifiers. Unlike standard unification, it is necessary to consider a set of substitutions.

Definition 7.2. Given a finite set E of equations, for any two terms s and t, and for any finite set W of variables such that $\mathrm{Var}(s) \cup \mathrm{Var}(t) \subseteq W$, a set S of substitutions is a *complete set of* E*-unifiers for* s *and* t *away from* W iff:

(i) for every $\sigma \in S$, one has $D(\sigma) \subseteq \mathrm{Var}(s) \cup \mathrm{Var}(t)$ and $I(\sigma) \cap W = \emptyset$;

(ii) for every $\sigma \in S$, one has $\sigma(s) \stackrel{\cdot}{\leftrightarrow}_E \sigma(t)$;

(iii) for every E-unifier θ of s and t, there is some $\sigma \in S$ such that $\sigma \leq_E \theta[W]$.

Condition (i) is the *purity condition*, condition (ii) the *coherence condition*, and condition (iii) the *completeness condition*. Condition (i) will be needed in the proof of the completeness theorem for SLDE† refutations. It is needed to ensure that the variables appearing in a resolvent clause are disjoint from the variables in the original literals being resolved on.

Unfortunately, it is undecidable whether two terms are unifiable modulo a set of equations. This can be shown by choosing the set of equations to contain the axioms for monoids. Then, the word problem for monoids is as an instance of the E-unification problem. Since the word problem for monoids is undecidable [39], so is E-unification. However, using a simple dovetailing argument, it can be shown that for every finite set E of equations and any two terms s and t, the set $U_E(s, t)$ of all E-unifiers of s and t is recursively enumerable. Note that for any finite set W containing $\mathrm{Var}(s) \cup \mathrm{Var}(t)$, the subset of $U_E(s, t)$ satisfying the purity condition with respect to W is a complete set of E-unifiers away from W. However, even though such a set is recursively enumerable, it may be highly redundant. It would be desirable, as in the case where $E = \emptyset$, to show the existence of complete sets of unifiers from which $U_E(s, t)$ can be generated by instantiations, and even better, complete sets of E-unifiers satisfying some minimality conditions. Such conditions were proposed by Huet in the framework of higher-order unification [20].

Let S be a complete set of E-unifiers of s and t away from W. Two minimality conditions can be defined.

Minimality: For any two substitutions $\sigma, \theta \in S$, if $\sigma \leq_E \theta[W]$, then $\sigma = \theta$.

Noncongruence: For any two substitutions $\sigma, \theta \in S$, if $\sigma \equiv_E \theta[W]$, then $\sigma = \theta$.

Note that minimality implies noncongruence. Unfortunately, there are difficulties with both concepts. Minimality cannot always be achieved, and noncongruence may

not be recursively enumerable. Fages and Huet [10] have shown that there exists a set of equations E and two terms s and t such that there is *no* complete and minimal set of E-unifiers for s and t. The reason why complete minimal sets of E-unifiers do not always exist is that the ordering induced by $\leq_E$ on the set of equivalence classes of $U_E(s,t)$ modulo $\equiv_E$ may not be well founded. When $\leq_E$ is well founded (that is, every strictly decreasing chain is finite), for every $\theta \in U_E(s,t)$ there is some element $\sigma \in U_E(s,t)$ minimal with respect to $\leq_E$, and such that $\sigma \leq_E \theta[W]$. In this case, a complete and minimal set of E-unifiers exists. More generally, if every decreasing chain (with respect to $\leq_E$, and even infinite), has a lower bound (the ordering $\leq_E$ is inductive), then using Zorn's lemma, it is not difficult to see that for every $\theta \in U_E(s,t)$, the set $\{\sigma \in U_E(s,t) | \sigma \leq_E \theta[W]\}$ has a minimal element. Hence, in this case, a complete minimal set of E-unifiers also exists.

The above discussion suggests to relax the condition of minimality as follows:

Weak minimality: For any two substitutions $\sigma, \theta \in S$, if $\sigma \leq_E \theta[W]$, then either $\sigma = \theta$ (θ is minimal), or there is no minimal element $\rho \in U_E(s,t)$ such that $\rho \leq_E \theta[W]$.

By "weeding out" elements of $U_E(s,t)$ greater than some minimal element, we can always show that a complete and weakly minimal set of E-unifiers exists. However, the argument below implies that some complete and weakly minimal sets of E-unifiers are not recursively enumerable.

For any finite set W containing $\mathrm{Var}(s) \cup \mathrm{Var}(t)$, by considering any set satisfying the purity condition with respect to W and obtained by selecting some substitution in each equivalence class of $U_E(s,t)$ modulo $\equiv_E$, a complete and noncongruent set of E-unifiers of s and t away from W is shown to exist. However, such a set may not be recursively enumerable. This is because if the word problem for E is undecidable, the restriction of $\equiv_E$ to $U_E(s,t)$ may not be recursive (but it is recursively enumerable). For example, we can choose E such that $E = \mathscr{E} \cup \{\text{axioms for monoids}\}$, where $\mathscr{E}$ is a set of equations such that deciding whether any two strings are congruent modulo $\stackrel{*}{\equiv}_{\mathscr{E}}$ is undecidable. Such a set $\mathscr{E}$ is given in Machtey and Young [39], where it is shown that for some fixed term v_0, the set of all terms u such that $u \stackrel{*}{\equiv}_{\mathscr{E}} v_0$ is not recursive. Then, for the terms $s = x$ and $t = v_0$ (where x is a variable), s and t are E-unifiable iff for some substitution σ, $\sigma(x) \stackrel{*}{\equiv}_E v_0$. Hence, $\stackrel{*}{\equiv}_E$ is not recursive. Observe that every substitution σ with support $\{x\}$ can be identified with the term $\sigma(x)$. Using this identification, note that $U_E(x, v_0)$ contains (among other things) the set of all ground terms u such that $u \stackrel{*}{\equiv}_E v_0$. Hence, $U_E(x, v_0)$ is not recursive. But then, given any two ground substitutions σ_1 and σ_2 with support $\{x\}$, $\sigma_1 \equiv_E \sigma_2$ iff $\sigma_1(x) \stackrel{*}{\equiv}_E \sigma_2(x)$. If the restriction of $\equiv_E$ to $U_E(x, v_0)$ were recursive, then by choosing σ_2 such that $\sigma_2(x) = v_0$, we could show that $U_E(x, v_0)$ is recursive, a contradiction. Hence, the restriction of $\equiv_E$ to $U_E(x, v_0)$ is not recursive. But since $\equiv_E$ is recursively enumerable, the restriction of its complement $\not\equiv_E$ to $U_E(x, v_0)$ is not recursively enumerable.

When E consists of ground equations, it has been shown by Kozen that E-unification is NP-complete [33, 34]. Kozen's proof consists in showing that if two terms are E-unifiable (with E ground), then there is some E-unifier σ satisfying the following property: There is a function $\varphi : \mathrm{Var}(s) \cup \mathrm{Var}(t) \to \mathrm{Subterms}(E \cup \{s, t\})$, where $\mathrm{Subterms}(E \cup \{s, t\})$ denotes the set of subterms occurring in terms in

$E \cup \{s, t\}$ such that for every $x \in \text{Var}(s) \cup \text{Var}(t)$, φ defines a (unique) substitution σ such that $\sigma(x) = \varphi \circ \sigma(x)$. Hence, σ is completely determined by the function $\varphi : \text{Var}(s) \cup \text{Var}(t) \to \text{Subterms}(E \cup \{s, t\})$. The above yields a nondeterministic polynomial-time algorithm for deciding E-unifiability: "Guess" φ, and check that $\sigma(s) \stackrel{\cdot}{\Leftrightarrow}_E \sigma(t)$, using the congruence closure algorithm. NP-completeness is easily shown because one can reduce the satisfiability problem to E-unifiability by choosing the equations in E to be an encoding of the truth tables for the logical connectives $\wedge$, $\vee$, and $\neg$ (for details, see Kozen [33, 34]). However, Kozen's proof does not show that a finite complete set of E-unifiers exists (for a ground set E). It is possible to prove this stronger result, and the first author has, in fact, extracted a complete E-unification procedure based on this proof [16].

We now consider the case in which an explicit procedure for enumerating complete sets of E-unifiers is available. The following definitions will be needed.

Definition 7.3.

(1) Given a set H of Horn clauses, let E_H be the set of equations occurring as the head of some clause in H. We say that H is *acceptable* iff we have some procedure UNIF(E_H) such that for any two terms s and t, and for any finite set W such that $\text{Var}(s) \cup \text{Var}(t) \subseteq W$, the procedure UNIF($E_H$)($s$, t, W) enumerates a complete set of E_H-unifiers for s, t away from W. [As noted earlier, such a procedure always exists, but in practice, UNIF(E_H) generates complete sets of unifiers having some special properties.]

(2) Given a set H of Horn clauses, we say that H is *well-behaved* iff
   (i) $H = E \cup C$ and $E \cap C = \emptyset$, where E is a set of equations (that is, atoms of sort $\neq$ *bool*) and C is a set of Horn clauses such that the head of each such clause is *not* an equation (that is, an atom of sort *bool*).
   (ii) H is acceptable. Note that $E_H = E$.

The class of well-behaved Horn clauses was introduced by Goguen and Meseguer, who have also investigated some of its properties [17]. For this class, only equations in E_H can be used in step (ii) of Definition 6.3. Hence, the search space required for constructing refutations is reduced. Actually, it is possible to define a refinement of the SLDE-refutation method applying to arbitrary acceptable sets of Horn clauses.

Definition 7.4. Let H be an acceptable set of first-order Horn clauses with equality consisting of a set D of definite clauses and a set $\{G_1, \ldots, G_q\}$ of goals. An SLDE† *derivation* for H is a sequence $\langle N_0, N_1, \ldots, N_p \rangle$ of negative clauses satisfying the following properties:

(1) $N_0 = G_j$, where G_j is one of the goals.

(2) For every N_i, $0 \leq i < p$, if $N_i = :- A_1, \ldots, A_{k-1}, A_k, A_{k+1}, \ldots, A_n$ (where A_k is of any sort, including *bool*), then either
   (i) there is some definite clause $C_i = A :- B_1, \ldots, B_m$ in D, some finite set

$$\mathcal{D} = \{s_1 \doteq t_1 :- \Gamma_1, \ldots, s_r \doteq t_r :- \Gamma_r\}$$

   of variants of definite clauses in D (such that any two clauses in this set have disjoint sets of variables, also disjoint from the set of variables in

N_i), and some finite set $\mathscr{S} = \{\theta_1, \ldots, \theta_r\}$ of substitutions; and, letting

$$\mathscr{E} = \{s_1 \doteq t_1, \ldots, s_r \doteq t_r\},$$

for some E_H-unifier σ_i of A_k and A given by the procedure UNIF(E_H), we have

$$\sigma_i(A_k) \Leftrightarrow_{[s_1 \doteq t_1, \theta_1]} \cdots \Leftrightarrow_{[s_r \doteq t_r, \theta_r]} \sigma_i(A)$$

and

$$N_{i+1} = :- \sigma_i(A_1), \ldots, \sigma_i(A_{k-1}), \sigma_i(B_1), \ldots, \sigma_i(B_m), \theta_1(\Gamma_1), \ldots, \theta_r(\Gamma_r), \sigma_i(A_{k+1}), \ldots, \sigma_i(A_n)$$

(where any of the Γ_j's or $B_1, \ldots, B_m$ may be empty); or

(ii) Letting $A_k = s \doteq t$ (where $s \doteq t$ is of any sort, including *bool*), there is some finite set

$$\mathscr{D} = \{s_1 \doteq t_1 :- \Gamma_1, \ldots, s_r \doteq t_r :- \Gamma_r\}$$

of variants of definite clauses in D (such that any two clauses in this set have disjoint sets of variables, also disjoint from the set of variables in N_i), and some finite set $\mathscr{S} = \{\theta_1, \ldots, \theta_r\}$ of substitutions; and, letting

$$\mathscr{E} = \{s_1 \doteq t_1, \ldots, s_r \doteq t_r\},$$

for some E_H-unifier σ_i of s and t given by the procedure UNIF(E_H), we have

$$\sigma_i(s) \Leftrightarrow_{[s_1 \doteq t_1, \theta_1]} \cdots \Leftrightarrow_{[s_r \doteq t_r, \theta_r]} \sigma_i(t)$$

and

$$N_{i+1} = :- \sigma_i(A_1), \ldots, \sigma_i(A_{k-1}), \theta_1(\Gamma_1), \ldots, \theta_r(\Gamma_r), \sigma_i(A_{k+1}), \ldots, \sigma_i(A_n)$$

(where any of the Γ_j's may be empty).

An SLDE† derivation is an *SLDE†-refutation* iff $N_p = \square$ (the empty clause).

Note that when H is well behaved, $\mathscr{D}$ is a set of equations, and there is no need for the set $\mathscr{S}$, since all the Γ_j's are empty.

In the case of arbitrary acceptable sets, the completeness of the SLDE† method depends on the procedure UNIF(E_H) used. This means that completeness cannot be guaranteed for *all* procedures enumerating a complete set of E_H-unifiers [but it is complete for the systematic procedures enumerating each set $U_{E_H}(s, t)$; see below]. The problem is the following. Assume that for some terms s, t and some set W, we have UNIF(E_H)$(s, t, W) \neq U_{E_H}(s, t)$ [of course, UNIF(E_H)$(s, t, W) \subseteq U_{E_H}(s, t)$]. Since UNIF($E_H$) enumerates complete sets of E_H-unifiers, for every E_H-unifier θ of s and t, there is some E_H-unifier $\sigma \in$ UNIF(E_H)(s, t, W) and some substitution η such that $\theta =_{E_H} \sigma \circ \eta$. However, the proof that UNIF(E_H) enumerates complete sets of E_H-unifiers may not yield enough information about the substitutions η to establish the completeness of the SLDE†-refutation method.

If we choose the procedure UNIF(E_H) to be the systematic procedure enumerating $U_{E_H}(s, t)$ for every pair of terms s and t, the proof of Theorem 6.5 goes through.

Hence, the SLDE†-refutation method is *complete for unification procedures enumerating all E-unifiers*. Unfortunately, this is not an improvement over the previous method, since such an enumeration procedure is a "brute force" procedure with no minimality properties at all.

However, we can show that the SLDE†-refutation method is complete for all well-behaved sets of Horn clauses, for *all* procedures UNIF(E_H) enumerating complete sets of E_H-unifiers. Hence, in this case, one can use procedures generating complete sets having some minimality conditions.

Theorem 7.5 (Completeness of SLDE† refutations for well-behaved sets of Horn clauses). Let H be a finite well-behaved set of first-order Horn clauses with equality. If H is unsatisfiable, then there is an SLDE† refutation for H.

PROOF. The proof is similar to that of Theorem 6.5, but that the following claim is used.

Claim. For every ground SLDE derivation $\langle N_0', N_1', \ldots, N_p' \rangle$ for the set of ground instances H', there is a SLDE† derivation $\langle N_0, N_1, \ldots, N_p \rangle$ for the set H, and some sequence $\langle \eta_0, \ldots, \eta_p \rangle$ of ground substitutions, such that $N_j' \stackrel{\cdot}{\Leftrightarrow}_{E_H} \eta_j(N_j)$ for every j, $0 \le j \le p$.

PROOF OF CLAIM. We proceed by induction on the length of derivations. The claim is trivial for $p = 0$. Next, we prove the claim for $p + 1$. In this proof, "derivation" means SLDE† derivation. By the induction hypothesis, there is a derivation $\langle N_0, \ldots, N_p \rangle$ satisfying the claim, and in particular, $N_p' \stackrel{\cdot}{\Leftrightarrow}_{E_H} \eta_p(N_p)$. Let $N_p = :- A_1, \ldots, A_{k-1}, A_k, A_{k+1}, \ldots, A_n$, and $N_p' = :- A_1', \ldots, A_{k-1}', A_k', A_{k+1}', \ldots, A_n'$. There are two cases:

Case 1: $N_p' = :- A_1', \ldots, A_{k-1}', A_k', A_{k+1}', \ldots, A_n'$, there is some definite clause $C_p = \theta(A :- B_1, \ldots, B_m)$ for some definite clause $C = A :- B_1, \ldots, B_m$ in D and some ground substitution θ, and

$$N_{p+1}' = :- A_1', \ldots, A_{k-1}', \theta(B_1), \ldots, \theta(B_m), A_{k+1}', \ldots, A_n',$$

where $A_k' = \theta(A)$. It can be assumed by renaming variables if necessary that η_p and θ have disjoint support. Then, we can let σ denote the union of η_p and θ, and since $A_k' \stackrel{\cdot}{\Leftrightarrow}_{E_H} \eta_p(A_k)$, we have

$$\sigma(A) \stackrel{\cdot}{\Leftrightarrow}_{E_H} \sigma(A_k).$$

Hence, σ is an E_H-unifier of A and A_k. Since H is well behaved, letting $W_{p+1} = \mathrm{Var}(N_p) \cup \mathrm{Var}(C)$, there is some E_H-unifier σ_{p+1} of A and A_k away from W_{p+1} given by the procedure UNIF(E_H) and some substitution η_{p+1}, such that $\sigma =_{E_H} \sigma_{p+1} \circ \eta_{p+1}[W_{p+1}]$. Then,

$$N_{p+1} = \sigma_{p+1}(:- A_1, \ldots, A_{k-1}, B_1, \ldots, B_m, A_{k+1}, \ldots, A_n)$$

and

$$N_{p+1}' \stackrel{\cdot}{\Leftrightarrow}_{E_H} \eta_{p+1}(N_{p+1}).$$

Hence, $\langle N_0, \ldots, N_{p+1} \rangle$ and $\langle \eta_0, \ldots, \eta_{p+1} \rangle$ also satisfy the claim.

Case 2: $N_p' \overset{\cdot}{\Leftrightarrow}_{E_H} \eta_p(N_p)$ as above, $A_k' = s' \doteq t'$, there is some finite set

$$\mathscr{E} = \{s_1 \doteq t_1, \ldots, s_m \doteq t_m\}$$

of variants of equations in D (such that any two equations in this set have disjoint sets of variables, also disjoint from the set of variables in N_i) and some finite set $\mathscr{S} = \{\theta_1, \ldots, \theta_m\}$ of ground substitutions, and we have

$$s' \overset{\cdot}{\Leftrightarrow}_{\mathscr{E}} t' \quad \text{and} \quad N_{p+1}' = :- A_1', \ldots, A_{k-1}, A_{k+1}, \ldots, A_n.$$

Since $s' \overset{\cdot}{\Leftrightarrow}_{E_H} \eta_p(s)$, $t' \overset{\cdot}{\Leftrightarrow}_{E_H} \eta_p(t)$, $s' \overset{\cdot}{\Leftrightarrow}_{\mathscr{E}} t'$, and $\mathscr{E} \subseteq E_H$, we have

$$\eta_p(s) \overset{\cdot}{\Leftrightarrow}_{E_H} \eta_p(t).$$

Hence, η_p is an E_H-unifier of s and t. Since H is well behaved, letting $W_{p+1} = \mathrm{Var}(N_p)$, there is some E_H-unifier σ_{p+1} of s and t away from W_{p+1} given by the procedure UNIF(E_H) and some substitution η_{p+1}, such that $\eta_p =_{E_H} \sigma_{p+1} \circ \eta_{p+1}[W_{p+1}]$. Then,

$$N_{p+1} = \sigma_{p+1}(:- A_1, \ldots, A_{k-1}, A_{k+1}, \ldots, A_n)$$

and

$$N_{p+1}' \overset{\cdot}{\Leftrightarrow}_{E_H} \eta_{p+1}(N_{p+1}).$$

Hence, $\langle N_0, \ldots, N_{p+1} \rangle$ and $\langle \eta_0, \ldots, \eta_{p+1} \rangle$ also satisfy the claim. This concludes the proof of the claim. □

Applying the claim to a refutation, the theorem is proved. □

The soundness of the SLDE†-refutation method is left as an exercise.

When the procedure UNIF(E_H) is an algorithm, and any two terms have a finite complete set of unifiers, the search space for the SLDE†-refutation method is further reduced. Fay [11] and Hullot [23] have given well-known algorithms to compute E-unifiers based on the concept of a narrowing substitution [52], and generalizations of these algorithms have been given by Kirchner and Kirchner [27–29]. The procedure of Martelli, Moiso, and Rossi [40] can also be nicely integrated with the SLDE†-refutation method in the case of canonical sets of equations.

8. COMPARISON WITH RELATED WORK

We have been able to identify four other approaches to handling equational Horn clauses which include rigorous completeness results. We now point out the main differences between these approaches and the methods defined in this paper. In particular, it is shown that E-unification is either incorporated or simulated in some form in each of these four approaches.

Jaffar, Lassez, and Maher [24] define a logic program as a pair (P, E) where P is the usual set of (nonequational) definite clauses and E is a set of *definite equality clauses* of the form

$$e \leftarrow e_1, e_2, \ldots, e_m$$

$m \geq 0$, where each e is an equation of the form $s \doteq t$ for terms s and t (but e cannot be a nonequational atom). A *(P, E)-derivation sequence* is a (finite or infinite)

sequence of triples $\langle G_i, \hat{C}_i, \theta_i \rangle$, $i = 0, 1, \ldots$, such that

(a) G_i is of the form $B_i, \ldots, B_m$ where $m \geq 0$ and each B_j is an atom, for all j, $0 \leq j \leq m$;

(b) $\hat{C}_i$ is a list of m clauses

$$A^{(1)} \leftarrow D_1^{(1)}, \ldots, D_{n_1}^{(1)},$$
$$A^{(2)} \leftarrow D_1^{(2)}, \ldots, D_{n_2}^{(2)},$$
$$\vdots$$
$$A^{(m)} \leftarrow D_1^{(m)}, \ldots, D_{n_m}^{(m)},$$

where each clause above is a clause in P with variables renamed;

(c) θ_i is an E-unifier of $(B_1, \ldots, B_m)$ and $(A^{(1)}, \ldots, A^{(m)})$;

(d) G_{i+1} is $(D_1^{(1)}, \ldots, D_{n_1}^{(1)}, D_1^{(2)}, \ldots, D_{n_2}^{(2)}, \ldots, D_1^{(m)}, \ldots, D_{n_m}^{(m)})\theta_i$.

A derivation sequence is *finitely failed* with length i if θ_i cannot be formed, and is *successful* if some G_i is empty (i.e, $m = 0$). The authors are able to show that the classic soundness and completeness results associated with standard logic programs [2] also hold in the above more general framework—in particular, that the least model of (P, E) is equal to the least fixpoint of $T_{(P,E)}$, where T is an operator defined analogously to the T-operator in Apt and van Emden [2].

It is important to point out that this method assumes a *very* powerful form of E-unification, namely unification modulo equational theories consisting of *arbitrary* definite equality clauses. Also, although they partition a logic program into definite clauses and definite equality clauses, it is easy to see that adding the distinguished term $\top$ to the term algebra over E and allowing equations of the form $s \doteq \top$ collapses this partition, and results in a system that admits Horn clauses with equality in their full generality. Such a system is closely related to the SLDE-refutation method presented in this paper. As the authors themselves point out, and as we observed in Sections 6 and 7 when discussing the SLDE-refutation system, these definitions do not shed much insight on corresponding *computational* methods implementing them. The problems are hidden in the complexities of the E-unification step. Finally, we note that the authors extend their results to include the soundness and completeness of the negation-by-failure rule for completed logic programs in the manner of Clark [5].

Goguen and Meseguer [17] define Eqlog, a logic-programming language that includes equality, types, and generic modules. This important work appears to represent the current state of the art in defining and implementing a logic-programming system with modern language features. The authors give a rigorous semantics for Eqlog, but their approach does not show the completeness of the *inference procedure*. To clarify this point, the authors prove that for predicate symbol Q and terms $t_1, \ldots, t_n$ containing free variables $Y_1, \ldots, Y_m$,

$$C \vdash \exists Y_1 \cdots \exists Y_m Q(t_1, \ldots, t_n)$$

iff there is a substitution σ sending Y_i to ground terms such that $Q([\sigma(t_1)], \ldots, [\sigma(t_n)])$ is true in an initial model for C. This version of Herbrand's theorem characterizes the abstract properties of the underlying logic. However, it is independent of the

mechanics of the inference method used to compute the operational semantics; the proof does not include a lifting of a ground case to the first-order case. In fact, the actual inference method used in Eqlog, as reported in [17], is described (Section 4) *in words*. After quoting a passage by Warren [56] on the computation algorithm or ordinary PROLOG, and defining the concept of an E-unification algorithm SOLN, the following paragraph occurs:

> The assumption that the set C of clauses decomposes into disjoint sets E of equations and P of predicate-headed clauses has the desirable effect of isolating the solution of equations into a separate E-unification algorithm SOLN, which is then called by the PROLOG search algorithm described above. Of course, SOLN must be called in a way that can be backtracked and is fair, in the sense that every substitution gets tried. This gives a semidecision procedure that may not halt; but if *soln* is r.e. complete, then a general proof of correctness of the algorithm can be given along standard lines.

As the proofs in Sections 6 and 7 illustrate, such a general proof must be handled very carefully if it is to result in insight for subsequent implementations. We also note that the authors subsequently extend by stages the definition of logic programs from the base case of $C = E \cup P$ (what we have called the class of well-behaved logic programs) to Horn-clause logic with equality in its full generality. They also give examples in Eqlog of these extensions. As in the case of Jaffar, Lassez, and Maher, and in our own analysis of acceptable logic programs in their full generality, such extensions seem to require forms of E-unification which appear to make completeness and tractability results problematical.

Dershowitz and Plaisted [6] define a system of conditional directed equations of the form

$$l[\bar{X}] :- p[\bar{X}, \bar{Y}] \rightarrow r[\bar{X}, \bar{Y}],$$

where l and r are first-order terms, p a predicate, and $\bar{X}$ and $\bar{Y}$ are sets of variables. The interpretation here is $p \supset l \rightarrow r$. Summarizing from [6], computation is performed by using an equation either (1) to simplify a subterm that matches its left-hand side, or (2) to narrow a subterm that unifies with its left-hand side. A computation begins with a goal rule of the form

$$g[\bar{x}, \bar{Z}] \rightarrow answer(\bar{Z}),$$

where g contains irreducible input terms x and output variables $\bar{Z}$. At each step in the computation, if the current subgoal is

$$h :- q_1, \ldots, q_n \rightarrow answer(\bar{s})$$

and a rule $l :- p \rightarrow r$ whose left-hand side can be unified with a nonvariable subterm of q_1 via a most general unifier σ at some context t (i.e., term address), then the subgoal q_1 is *conditionally* narrowed to

$$\sigma(h) :- \sigma(p), t[\sigma(r)], \sigma(q_2), \ldots, \sigma(q_n) \rightarrow answer(\sigma(\bar{s})).$$

This goal is then simplified by term rewriting as much as is possible. Only when all the conditions become *true*, and the subgoal is of the conditional form $h' \rightarrow answer(\bar{s})$, are narrowing substitutions attempted for h'. The computation ends when a *solution rule* of the form $true \rightarrow answer(\bar{t})$ is derived.

This system can be viewed as a restriction of case (ii) of the SLDE-refutation method in which equational Horn clauses are restricted to clauses of the form

$$l(\bar{x}) \doteq r(\bar{x}, \bar{Y}) :- p(\bar{x}, \bar{Y})$$

whose heads are *oriented* left to right, i.e., they represent one-way rewrite rules rather than two-way rewrite rules as in the SLDE-refutation method. The system tries to E-unify the head of the goal rule with $\top$, and all the narrowing substitutions in the computation sequence are steps on the way to doing this. Indeed, the authors' description (p. 58) of the method of computing narrowing substitutions is very reminiscent of Fay's [11] algorithm for finding E-unifiers: "Since, in general, there may be many ways to achieve a subgoal, alternative narrowing computations must be attempted, either in parallel (until one succeeds) or sequentially (by backtracking upon failure)." Since the set E upon which E-unification takes place can change, this is again as in the Jaffar case, a very powerful form of E-unification. This work represents an intelligent and practical restriction of equational Horn clauses.

Fribourg in [12,13] describes SLOG, an equational Horn-clause interpreter based on a form of clausal superposition and term rewriting. The method is general in that it applies to arbitrary equational Horn clauses, but he concentrates his discussion on programs containing clauses of the form

$$L \doteq R :- Q_1 \doteq \top, \ldots, Q_n \doteq \top,$$

where $n \geq 0$. A goal in this system is a clause of the above form in which the head is empty. For simplicity, we will abbreviate a term $Q_i \doteq \top$ with the usual form Q_i. We note also that Fribourg actually gives a series of definitions, each assuming specific restrictions or properties on the input equational Horn clauses or on the superposition operation. We have extracted a simplified but "representative" definition and refer the reader to the Fribourg's papers for actual definitions.

Summarizing from [13], let G be a goal of the form $:- Q_1, \ldots, Q_n$, and P be a set of definite equational clauses of the form $L \doteq R :- B_1, \ldots, B_m$. Then, G' is an *innermost goal superposant* of P into G at address α using most general unifier σ iff Q_1 has a nonvariable subterm M "which itself contains no matchable proper subterm" such that $\sigma(M) = \sigma(L)$, and G' is

$$:- \sigma(B_1, \ldots, B_m, Q_1[\alpha \leftarrow R], Q_2, \ldots, Q_n).$$

A substitution σ is defined as a GC-*substitution* if it substitutes ground terms defined only on the constructors in $P \cup G$. The constructors [21] are (loosely) the set of non-user-defined symbols. C is an *inductive consequence* of P iff for any GC-substitution σ, $P \cup E \models \sigma(C)$, where E is the set of equality axioms. Finally, if R denotes the rewrite system composed of the inductive consequences of P, an SLOG program is a pair $\langle P, R \rangle$. R is usually taken to be canonical via completion by the Knuth-Bendix algorithm.

Let $A : \langle P, R \rangle$ be an SLOG program. An *S-derivation* of G' from $P \cup G$ via a superposant selection function ϕ consists of a finite sequence $G_0, G_1, \ldots, G_n$ of goals and a sequence $\sigma_1, \sigma_2, \ldots, \sigma_n$ of most general unifiers such that

(1) G_0 is the R-normal form of G, and G_n is G';

(2) for all i, $1 \leq i \leq n$, G_i is the R-normal form of a ϕ-superposant of a clause in P into G_{i-1}.

Finally, an *S-refutation* is an S-derivation of the empty clause. Note that in this system, the clauses in P are used not only in superposition on the "leftmost-innermost" literal of the goal, but also as rewrite rules to simplify all literals in the goal as well.

Fribourg gives a rigorous and stepwise development of this system, and gives completeness results, including an analysis of the ground case with lifting lemmas, for each step. Note that as in the case with the Dershowitz-Plaisted system, it is possible to interpret this system as including a form of E-unification by viewing goal superposition as an instance of the narrowing operation, and describing the aim of the system as attempting to find an E-unifier for each literal of the goal and $\top$. It is again a very powerful form of E-unification, since this system applies to arbitrary equational Horn clauses. Finally, Fribourg extends his results by formalizing the closed-world assumption [47] in the above framework and considers implementation methods.

We also mention the work of Kaplan [25], who presents a conditional term-rewriting system which allows the simulation of conditional equational Horn theories. This system consists of rules of the form

$$G \to D \leftarrow u_1 = v_1, \ldots, u_n = v_n$$

with the restriction that $\mathrm{Var}(D) \subset \mathrm{Var}(G)$, for every i, $1 \le i \le n$, $\mathrm{Var}(u_i) \cup \mathrm{Var}(v_i) \subset \mathrm{Var}(G)$. He shows that this formalism is too general, in the sense that for any given set of conditional rules, the rewriting of any term is in general an undecidable problem, even if the system is canonical. The paper also contains a review of related efforts to cope with the undecidability phenomenon by restrictions to the above formalism. This work is relevant to the work involving E-unification in that the standard E-unification algorithm is based on the canonical *unconditional* term-rewriting systems. It is possible that canonical conditional systems could form the basis for more efficient or general E-unification algorithms, and in fact there has been some encouraging recent progress on checking confluence for conditional rewrite rules [46].

It is worth noting that there are important semantic and computational differences between the use of equational Horn-clause languages (as in [17] and [6]), and the use of conditional rewrite systems (as in Kaplan [25])—even though the problems addressed in both cases are undecidable for the general case. In an equational Horn-clause language, one attempts to show

$$\models E \supset \exists z_1 \cdots \exists z_n (s \doteq t),$$

where $\{z_1, \ldots, z_n\}$ is set of free variables in $(s \doteq t)$, and find *explicit* terms $t_1, \ldots, t_n$ such that

$$\models E \supset (s \doteq t)[t_1/z_1, \ldots, t_n/z_n].$$

This can be accomplished by refutation by showing that

$$E \wedge \forall z_1 \cdots \forall z_n \neg (s \doteq t)$$

is unsatisfiable, which implies, by the Herbrand-Skolem-Gödel theorem and theorem 3.6 that there exists a substitution $\sigma = [t_1/z_1, \ldots, t_n/z_n]$ and a set E' of ground instances of E such that

$$\begin{aligned} & E' \wedge \neg\sigma(s \doteq t) \text{ is unsatisfiable} \\ \text{iff} \quad & E' \supset \sigma(s \doteq t) \text{ is valid} \\ \text{iff} \quad & E \models \sigma(s \doteq t), \end{aligned}$$

which implies that $\sigma(s)$ rewrites to $\sigma(t)$ for some sequence of rewrite steps using the equations in E. (Note: σ is an E-unifier of s and t.)

The concept of narrowing occurs naturally when we attempt to convert a sequence of rewrite steps from $\sigma(s)$ to $\sigma(t)$ into a sequence of extended rewrite steps between the original terms s and t. Such extended rewrite steps, called *narrowing steps*, must incorporate unification, because the term $\sigma(s)$ can be rewritten using a rule $l \to r$ iff some instance $\theta(l)$ of l is equal to some subterm $\sigma(s)/u$ of $\sigma(s)$. If $u \in \mathrm{dom}(s)$, then $\sigma(s)/u = \sigma(s/u)$, and we have $\theta(l) = \sigma(s/u)$, that is, s/u and l are unifiable. [The case where $u \notin \mathrm{dom}(s)$ is more difficult, and will not be discussed here. Note that a way to prevent this case from happening is to assume that E is in fact a set of confluent and Noetherian rewrite rules. Then, we can assume that the substitution σ is reduced, that is, every $\sigma(x)$ is irreducible for every $x \in D(\sigma)$. In such a case, if we had $u \notin \mathrm{dom}(s)$, then if $\sigma(s)$ were reducible, $\sigma(x)$ would be reducible for some variable $x \in \mathrm{dom}(s)$, which contradicts the fact that σ is reduced.] A narrowing substitution is in fact a most general unifier of s/u and l. Such a substitution allows the substitution instance $\sigma(s)$ of a term s to be further reduced. Since any term may have multiple narrowing substitutions, the computation of an E-unifier $\sigma = [t_1/z_1, \ldots, t_n/z_n]$ such that $\models E \supset \sigma(s \doteq t)$ involves finding a specific sequence of narrowing steps (and reductions).

In a conditional rewrite system, one wants to know whether an equation $s \doteq t$ is a *logical consequence* of a theory E, that is, for terms s and t, whether

$$\models E \supset \forall z_1 \cdots \forall z_n (s \doteq t),$$

where $\{z_1, \ldots, z_n\}$ is a set of variables free in s and t. A refutation thus verifies that

$$E \wedge \exists z_1 \cdots \exists z_n \neg (s \doteq t)$$

is unsatisfiable, which after skolemizing is equivalent to verifying that

$$E \wedge \neg (s' \doteq t')$$

is unsatisfiable, where s' and t' are ground terms obtained by substituting constant symbols for the variables ("freezing the free variables"). Again, applying the Herbrand-Skolem-Gödel theorem, there is a set E' of ground instances of E such that

$$\begin{aligned} & E' \wedge \neg (s' \doteq t') \text{ is unsatisfiable} \\ \text{iff} \quad & E' \supset s' \doteq t' \text{ is valid} \\ \text{iff} \quad & E \models s' \doteq t', \end{aligned}$$

which as before, implies that s' rewrites to t' for some sequence of rewrite steps using the equations in E. But since s' and t' are ground (since they are the result of substituting constants for the variables in s and t), there is no need to extend the rewrite steps from s' to t' to narrowing steps from s to t (provided that we always rename the variables occurring in the equations in E used as rewrite rules away from the variables in $s \doteq t$). This observation, together with the completeness of the rewrite-rule method in the ground case, can be used to give an alternate proof of the completeness of the rewrite-rule method for equational logic.

Another interesting approach worth noting is that of Miller and Nadathur. In their paper [41], Miller and Nadathur present a higher-order extension of PROLOG

that includes predicate and function variables, and typed λ-terms. Although their language does not include equality, their proof procedure is similar in spirit to our method, because the presence of typed λ-terms requires the use of a procedure for enumerating unifiers. Hence, their proof procedure also mixes backchaining steps and enumerations of (higher-order) unifiers. Technically, the main difference is the use of Huet's unification procedure for typed λ-terms [20], instead of an E-unification procedure. In some sense, they are dealing with a fixed set E of equations corresponding to the rules of λ-conversion. This approach appears quite promising, since Huet showed that his procedure enumerates a complete and minimal set of *preunifiers* [20].

Finally, we note that there has also been substantial work relevant to this paper which attempts to combine features of logic programming with functional programming, or to include facilities for solving equations. We distinguish this work from the methods above (and our own methods), in that the emphasis is not on defining and showing completeness results of a proof system for the underlying equational Horn-clause logic.[1] Kornfeld [32], who appears to have been the first to explicitly consider incorporating equality into logic programming, extends PROLOG by allowing the inclusion of assertions about equality. He does not attempt to provide a theoretical foundation for this extension, and in fact, the method does not appear to be complete. Reddy [45] subsequently gave a correct computation method for assertions about equality, reduction by narrowing, and the related semantics in the context of including logical variables in functional languages. Lindstrom [37] describes FLG + LV, a functional language that includes logical variables, in which terms are simplified before unification. Funlog [53], defined without formal semantics, also combines the functional- and logic-programming paradigms. Hansson, Haridi, and Tarnlund [18] define a superset of Horn-clause logic which includes negation, equality, and explicit universal quantifiers based on a natural deduction system; however, again no formal semantics is given. Also based on natural deduction semantics is the language LEAF [3], which extends the logic programming to provide functional notation. Tamaki [54] describes a system requiring the addition of the equality axioms which includes a reducibility predicate which can simulate narrowing. FPL [4] is essentially a logic-programming notation for a functional programming language, in that it does not support logical variables. LOGLISP [48] and QLOG [31] define essentially embeddings of PROLOG in LISP. Hoffmann and O'Donnell [19] define a purely equational language which supports user-defined abstract data types by regarding the equations as rewrite rules. Finally, in promising recent work, Kieburtz [26] defines a functional language, F + L, with interpreted equality which has both an efficient implementation by compiled graph reduction and a well-defined semantics.

9. CONCLUSIONS

We have presented two methods based on SLD resolution with E-unification for establishing the unsatisfiability of equational Horn clauses. The completeness proofs for these methods are based in the ground case on a generalization of the idea of a

[1] Because of the volume of work on this topic, our list can only be representative.

congruence closure to sets of ground Horn clauses. The SLDE-refutation method applies to arbitrary sets of equational Horn clauses, but is not practical in that it assumes a procedure which gives an explicit sequence of substitutions for each E-unifier. The SLDE†-refutation method applies to sets of equational Horn clauses which admit a procedure enumerating a complete set of E-unifiers, and is shown to be complete for sets of well-behaved equational Horn clauses.

The above methods and their completeness proofs illustrate the *computational* implications of including equality for specific classes of equational Horn-clause logic programs. Specifically:

(1) For the class of well-behaved sets of equational Horn clauses, i.e. sets which admit a procedure enumerating a complete set of E-unifiers, and contain clauses of the form

$s \doteq t$ not of sort *bool*

or

$Q :- P_1, \ldots, P_n,$

where s and t are first-order terms, Q is a nonequational atomic formula, and $P_1, \ldots, P_n$ are either equational or nonequational atomic formulae, the E-unification algorithm due to Fay [11] or Hullot [23] coupled with SLD resolution can be used as a relatively efficient interpreter. This is an important observation, since this class of equational Horn-clause programs subsumes the paradigms of functional, logic, and equational programming.

(2) For larger classes, the issues relating to efficient implementations are complex. A simple extension of the above approach to *conditionally* well-behaved sets of equational Horn clauses, which include clauses of the form

$s \doteq t :- Q_1, \ldots, Q_n,$

where $Q_1, \ldots, Q_n$ are nonequational atomic formula, seems to demand a SLD interpreter which must backtrack over calls to the E-unification algorithm on *different* sets of equations. Stated in another way, if E_H is the set of equations occurring in the head of some clause, a system using the E-unification algorithm of Fay would be forced to consider different canonical subsets of E_H.

It is clear that logic programming can benefit from the inclusion of equality. However, this inclusion must not compromise the inherent efficiency that allows this paradigm to be used as a programming language. Our results indicate that for systems involving E-unification, including the SLDE- and SLDE†-refutation methods, and the related work reviewed in the last section, the inclusion of equality for classes of programs larger than the well-behaved class appears to compromise this efficiency. It remains to be seen how applying intelligent restrictions to these larger classes, as in Dershowitz and Plaisted's work, or "distributing" the E-unification process across other operations, for instance superposition with term rewriting as in Fribourg, affects this issue.

We wish to thank Gopalan Nadathur for reading the manuscript very carefully, and for some incisive comments, particularly regarding the discussion on minimality conditions.

REFERENCES

1. Anderson, R., Completeness Results for E-Resolution, in *Proceedings AFIPS 1970,* Spring Joint Computer Conference, Vol. 36, AFIPS Press, Montvale, NJ; 1970, pp. 653–656.
2. Apt, K. and van Emden, M., Contributions to the Theory of Logic Programming, *J. Assoc. Comput. Mach.* 29:841–862 (July 1982).
3. Barbutti, R., Bellia, M., Levi, G., and Martelli, M., LEAF: A Language Which Integrates Logic, Equations, and Functions, in: D. Degroot and G. Lindstrom (eds.), *Functional and Logic Programming,* Prentice-Hall, 1985.
4. Bellia, M., Degano, P., and Levi, G., Call by Name Semantics of a Clause Language with Functions, in: K. L. Clark and S.-A. Tarlund (eds.), *Logic Programming*, Academic, New York, 1982.
5. Clark, K. L., Negation as Failure, in H. Gallaire and J. Minker (eds.), *Logic and Databases*, Plenum Press, New York, 1978, pp. 293–322.
6. Dershowitz, N. and Plaisted, D. A., Logic Programming cum Applicative Programming, in: *1985 IEEE Symposium on Logic Programming*, Boston, pp. 54–67.
7. Dowling, W. F., and Gallier, J. H., Linear-Time Algorithms for Testing the Satisfiability of Propositional Horn Formulae, *J. Logic Programming* 3:267–284 (1984).
8. Downey, P. J., Sethi, R., and Tarjan, E. R., Variations on the Common Subexpressions Problem, *J. Assoc. Comput. Mach.* 27(4):758–771 (1980).
9. Fages, F., Associate-Commutative Unification, in: *Proceedings of CADE-7,* Napa, 1984, pp. 194–208.
10. Fages, F. and Huet, G., Unification and Matching in Equational Theories, in: *Proceedings of CAAP 83,* Vol. 159, Springer, l'Aquilla, Italy, 1983, pp. 205–220.
11. Fay, M., First-Order Unification in an Equational Theory, in: *Proceedings of the 4th Workshop on Automated Deduction,* Austin, Texas, 1979.
12. Fribourg, L., Oriented Equational Clauses as a Programming Language, *J. Logic Programming* 2:165–177 (1984).
13. Fribourg, L., SLOG: A Logic Programming Language Interpreter Based on Clausal Superposition and Rewriting, in: *1985 IEEE Symposium on Logic Programming,* Boston, pp. 172–184.
14. Gallier, J. H., Fast Algorithms for Testing Unsatisfiability of Ground Horn Clauses with Equations, *J. Symbolic Comput.* to appear.
15. Gallier, J. H. *Logic for Computer Science: Foundations of Automatic Theorem Proving,* Harper and Row, New York, 1986.
16. Gallier, J. H., and Snyder, W., A General, Complete *E*-unification Procedure, presented at RTA '87, Bordeaux, France, May 1987.
17. Goguen, J. A. and Meseguer, J., Eqlog: Equality, Types, and Generic Modules for Logic Programming, in: D. Degroot and G. Lindstrom (eds.), *Functional and Logic Programming,* Prentice-Hall, 1985; short version, *J. Logic Programming* 2:179–210 (1984).
18. Hansson, A., Haridi, S., and Tarnlund, S.-A., Properties of a Logic Programming Language, in: K. L. Clark and S.-A. Tarnlund (eds.), *Logic Programming,* Academic, New York, 1982.
19. Hoffman, C. M. and O'Donnell, M. J., Programming with Equations, *TOPLAS* 4:83–112 (1982).
20. Huet, G. Résolution d'Equations dans les Langages d'Ordre $1, 2, \ldots, \omega$, Thèse d'Etat, Univ. de Paris VII, 1976.
21. Huet, G. and Hullot, J. M. Proofs by Induction in Equational Theories with Constructors, *J. Comput. System Sci.* 25:239–266 (1982).
22. Huet, G. and Oppen, D. C., Equations and Rewrite Rules: A Survey, in: R. V. Book (ed.), *Formal Languages: Perspectives and Open Problems*, Academic, New York, 1982.

23. Hullot, J.-M., Canonical Forms and Unification, in: *Proceedings of CADE-5*, 1980, pp. 318–334.
24. Jaffar, J., Lassez, J.-L., and Maher, M. J., A Theory of Complete Logic Programs with Equality, *J. Logic Programming* 3:211–223 (1984).
25. Kaplan, S., Conditional Rewrite Rules, *Theoret. Comput. Sci.* 33:175–193 (1984).
26. Kieburtz, R., Functions + Logic in Theory and Practice, Tech. Rep., Oregon Graduate Center, Feb. 1987.
27. Kirchner, C., Méthodes et Outils de Conception Systematique d'Algorithmes d'Unification dans les Theories Equationnelles, Thèse d'Etat, Univ. de Nancy I, 1985.
28. Kirchner, H., Preuves Par Completion dans les Variétés d'Algèbres, Thèse d'Etat, Univ. de Nancy I, 1985.
29. Kirchner, C. and Kirchner, H., Contribution à la Resolution d'Equations dans les Algèbres Libres et les Variétés Equationnelles d'Algèbres, Thèse de 3^e cycle, Univ. de Nancy I, 1982.
30. Knuth, D. E. and Bendix, P. B., Simple word problems in universal algebras, in: J. Leech (ed.), *Computational Problems in Abstract Algebra*, Pergamon, 1970, pp. 263–297.
31. Komorowski, H. J., QLOG—The Programming Environment for PROLOG in LISP, in: K. L. Clark and S.-A. Tarnlund (eds.), *Logic Programming*, Academic, 1982, pp. 315–322.
32. Kornfeld, W. A. Equality for Prolog, in: *Proceedings of 8th IJCAI*, Karlsruhe, 1983, pp. 514–519.
33. Kozen, D., Complexity of Finitely Presented Algebras, Tech. Rep. TR 76–294, Dept. of Computer Science, Cornell Univ., Ithaca, NY, 1976.
34. Kozen, D., Complexity of Finitely Presented Algebras, in: *9th STOC Symposium*, Boulder, CO, May 1977, pp. 164–177.
35. Kozen, D., Finitely Presented Algebras and the Polynomial Time Hierarchy, Technical Report TR 77–303, Dept. of Computer Science, Cornell Univ., Ithaca, NY, 1977.
36. Kozen, D., First-Order Predicate Logic Without Negation is NP-Complete, Technical Report TR 77–307, Dept. of Computer Science, Cornell Univ., Ithaca, NY, 1977.
37. Lindstrom, G., Functional Programming and the Logical Variable in: *Proceedings of the ACM Symposium on Principles of Programming Languages*, 1985.
38. Lloyd, J. W. *Foundations of Logic Programming*, Springer, New York, 1984.
39. Machtey, M. and Young, P. R., *An Introduction to the General Theory of Algorithms*, Elsevier North-Holland, New York, 1977.
40. Martelli, A., Moiso, C., and Rossi, G. F., An Algorithm for Unification in Equational Theories, in: *Third IEEE Symposium on Logic Programming*, Salt Lake City, Sept. 1986, pp. 180–186.
41. Miller, D., and Nadathur, G., Higher-Order Logic Programming, in: *Proceedings of the Third International Conference on Logic Programmming*, London, July 1986.
42. Nelson G., and Oppen, D. C., Fast Decision Procedures Based on Congruence Closure, *J. Assoc. Comput. Mach.* 27(2):356–364 (1980).
43. Oppen, D. C., Reasoning about Recursively Defined Data Structures, *J. Assoc. Comput. Mach.* 27(3):403–411 (1980).
44. Plotkin, G., Building in Equational Theories, *Mach. Intell.* 7:73–90 (1972).
45. Reddy, U.S., Narrowing as the Operational Semantics of Functional Languages, in: *1985 IEEE Symposium on Logic Programming*, Boston, pp. 138–155.
46. Remy, J. L., Etude des Systemes de Réécriture Conditionnels et Application aux Types Abstraits Algebriques, Ph.D. Thesis, Centre de Recherche en Informatique de Nancy, 1982.
47. Reiter, R., On Closed World Data Bases, in: H. Gallaire and J. Minker (eds.), *Logic and Databases*, Plenum, New York, 1980, pp. 55–76.

48. Robinson, J. A. and Sibert, E. E., LOGLISP: An Alternative to PROLOG, in: *Machine Intelligence* 10, Ellis Horwood, 1982, pp. 399–419.
49. Robinson, G. A. and Wos, L., Paramodulation and Theorem-Proving in First-order Logic with Equality, *Mach. Intell.* 4:135–150 (1969).
50. Siekmann, J. H., Universal Unification, in: *Proceedings CADE-7*, Napa, 1984, pp. 1–42.
51. Shoenfield, J. R., *Mathematical Logic*, Addison-Wesley, Reading, MA, 1967.
52. Slagle, J. R., Automated Theorem Proving for Theories with Simplifiers, Commutativity, and Associativity, *J. Assoc. Comput. Mach.* 21:622–642 (1974).
53. Subrahmanyam, P. A. and You, J.-H., Conceptual Basis and Evaluation Strategies for Integrating Functional and Logic Programming, in: *1984 IEEE Symposium on Logic Programming*, Atlantic City, pp. 144–153.
54. Tamaki, H., Semantics of a Logic Programming Language with a Reducibility Predicate, in: *1984 IEEE Symposium on Logic Programming*, Atlantic City, pp. 259–264.
55. Tarjan, R. E., Efficiency of a Good but not Linear Set Union Algorithm, *J. Assoc. Comput. Mach.* 22(2):215–225 (1975).
56. Warren, D., Logic Programming and Compiler Writing, *Software Practice and Experience* 10:97–125 (1980).