Journal of Automated Reasoning **17**: 199–214, 1996.
© 1996 *Kluwer Academic Publishers. Printed in the Netherlands.*

On the Completeness of SLDENF-Resolution

MICHAEL THIELSCHER
FG Intellektik, TH Darmstadt, Alexanderstraße 10, D-64283 Darmstadt, Germany
e-mail: mit@intellektik.informatik.th-darmstadt.de

(Received: 1 September 1994)

Abstract. SLDENF-resolution combines the negation-as-failure principle for logic programs involving negation, and SLDE-resolution for logic programs with an underlying equational theory. Recently, J. Shepherdson proved the soundness of this resolution principle wrt. an extended completion semantics. In this note, we investigate the particular problems of obtaining completeness which are caused by adding equational theories. As a concrete result we show to what extent the classical result for hierarchical and allowed nonequational programs can be generalized.

Key words: logic programming, negation as failure, unification theory.

1. Introduction

The notion of equality undoubtedly plays a fundamental role in most areas of mathematics and is therefore widely used, especially in mathematical applications of automated deduction. It emerged early that the simplest way of treating equality in automated theorem proving, namely, adding the standard equality axioms to the premises, is intractable in practice. Following a vision of Robinson [14], Plotkin [13] showed that the presence of equational theories can adequately and much more efficiently be modeled by integrating equational theories into the unification procedure. Plotkin's pioneering work was the signal for the formation of a subfield in automated deduction called *unification theory*, which has established itself today and is still growing (a survey is given in [22] or [3], e.g.). Among others, the field of logic programming profits from these developments insofar as the standard SLD-resolution for logic programs was also extended to handle equational theories. The resulting computation mechanism, called SLDE-resolution, was developed in the past decade [9, 5, 6]. Though less expressive than a general constraint-based approach [10], incorporating equational theories into logic programs offers the advantage of being a straightforward extension of standard logic programming, thus allowing for adapting many concepts and results from the special case (see [3] for a more detailed comparison).

Meanwhile, but independently, semantics for the treatment of negation in logic programs were developed and thoroughly analyzed. The first basic idea was introduced in [4], namely, the notion of *completion* as a uniform semantics for the negation-as-failure principle, where negation is regarded as failure to prove. Since then, most work concerning the so-called SLDNF-resolution concentrated on the one hand on aspects of consistency by defining classes of programs that have a consistent completion, and on the other hand on completeness of this computation mechanism (e.g., [8, 16, 18, 1, 15, 23]; a survey is given in [2]).

A first step towards the combination of proving with equality and employing negation-as-failure was made in [9], where the concept of failure in case of equational theories was investigated by extending the notion of Clark's completion semantics [4] to handle equality. However, this work concentrated on failure in so-called definite logic programs, where no negation occurs in the program clauses. Recently, resolution in equational logic programs involving negation was investigated in [20], where Shepherdson proved the soundness of what we call *SLDENF-resolution* wrt. this extended completion semantics. In this note, we investigate the particular problems of obtaining completeness which are caused by the additional equational theories. As a concrete result we show to what extent the classical result for hierarchical and allowed nonequational programs can be generalized.

In the following section, we briefly repeat the fundamental concepts of equational logic programs with negation, some important notions from unification theory, and the extended completion semantics required in case of equational theories. We also define SLDENF-resolution (in a slightly modified way compared with [20]). In Section 3, we discuss the obstacles for a completeness result that are caused by additional equational theories. As a consequence, we establish a completeness result for hierarchical and allowed equational logic programs provided the underlying equational theory meets two important restrictions.

2. Foundations

2.1. NORMAL EQUATIONAL LOGIC PROGRAMS

We adopt the *universal language approach* of [11] in this paper; that is, we assume the programs and goal clauses under consideration be built up from a countable set of predicates and functions given once and forever. The set of predicate symbols should include the special equality predicate $\doteq$. A *normal program clause* is an expression

$$A \leftarrow B_1, \ldots, B_l,$$

where the *head* A is an atom, but not an instance of the equality predicate, and the elements of the *body* $B_1, \ldots, B_l$ are literals, possibly the equality predicate being among them ($l \geqslant 0$). If p is the predicate symbol occurring in the head of a clause, this clause is said to *define* p. The reason for restricting the head of a program clause to nonequational atoms is that equality relations are defined separately: A *normal equational logic program* (P, E) consists of a finite set P of normal program clauses along with an *equational theory* E. Such a theory consists of a number of axioms $s \doteq t$ that are implicitly assumed to be universally closed. For example, the equational theory

$$E_C = \{h(x,y) \doteq h(y,x)\} \tag{1}$$

describes the law of commutativity for the function h. The following classification is required later in this paper: An equational theory E is called *regular* iff for each $s \doteq t \in E$ the set of variables occurring in s equals the set of variables occurring in t. For instance, E_C is regular, while

$$E_0 = \{h(x,0) \doteq 0\} \tag{2}$$

describing a neutralizing element 0 is not.[1]

A program (P, E) is called *hierarchical* [4, 12] if we can find a *level mapping* that assigns a natural number to each predicate symbol such that the following holds for each clause in P: The level of the predicate symbol occurring in the head is greater than the level of each predicate occurring in the body.

2.2. UNIFICATION THEORY

By $Var(e)$ we denote the variables occurring in the expression e. The *restriction* of a substitution σ to a set V of variables – written $\sigma|_V$ – is defined by $x(\sigma|_V) = x\sigma$ if $x \in V$ and $x(\sigma|_V) = x$ otherwise. Given an equational theory E, by $E \models \Phi$ we denote that the formula Φ is a logical consequence of the axioms in E plus the standard axioms of equality, viz. $\forall(x \doteq x)$ (*reflexivity*), $\forall(x \doteq y \rightarrow y \doteq x)$ (*symmetry*), $\forall(x \doteq y \wedge y \doteq z \rightarrow x \doteq z)$ (*transitivity*), $\forall(x_i \doteq y \rightarrow f(x_1,\ldots,x_i,\ldots,x_n) \doteq f(x_1,\ldots,y,\ldots,x_n))$ (*substitutivity for functions*), $\forall(x_i \doteq y \rightarrow [p(x_1,\ldots,x_i,\ldots,x_n) \leftrightarrow p(x_1,\ldots,y,\ldots,x_n)])$ (*substitutivity for predicates*). We call two terms s and t *equal wrt. E* – written $s =_E t$ – iff $E \models s \doteq t$.

A substitution σ is called an *E-unifier* of s and t iff $s\sigma =_E t\sigma$. E.g., $\{x \mapsto b, y \mapsto a\}$ is an E_C-unifier of $h(x,a)$ and $h(y,b)$ (cf. (1)). An *E-unification problem* consists of two terms s and t and is the problem of establishing whether there exists an E-unifier of s and t, i.e., whether s and t are *E-unifiable*. The set of all E-unifiers of s and t is denoted by $U_E(s,t)$.

As usual, we can define a subsumption ordering on substitutions: Two substitutions σ and θ are *E-equivalent* wrt. a set V of variables – written $(\sigma =_E \theta)|_V$ – iff $\forall x \in V \cdot x\sigma =_E x\theta$. A substitution σ is called *more general* than a substitution θ wrt. a set V of variables – written $(\sigma \leqslant_E \theta)|_V$ – iff there is a substitution τ

[1] For sake of clarity, our definition differs from the one used in [20] insofar as we adopt the (restricted) notion of equational theories used in unification theory instead of including the case where equality is defined through (more general) Horn formulas based on the equality predicate. We chose this notion because it allows to adopt the concept of regularity, which is of importance in our analysis. However, our results are also valid in the general setting where the notion of a regular theory is replaced by the general property that whenever two terms are equal under the theory, they contain an identical set of variables.

such that $(\sigma\tau =_E \theta)|_{\mathcal{V}}$. A set $cU_E(s,t)$ is then called a *complete set of E-unifiers* iff $cU_E(s,t) \subseteq U_E(s,t)$ (correctness) and $\forall\theta \in U_E(s,t).\ \exists\sigma \in cU_E(s,t).\ (\sigma \leqslant_E \theta)|_{\mathcal{V}ar(s)\cup\mathcal{V}ar(t)}$ (completeness). A set $\mu U_E(s,t)$ is called a *minimal set of E-unifiers* iff it is complete and $\forall\sigma,\theta \in \mu U_E(s,t)[(\sigma \leqslant_E \theta)|_{\mathcal{V}ar(s)\cup\mathcal{V}ar(t)} \Rightarrow \sigma = \theta]$ (minimally). The unification problem wrt. an equational theory E is called *finitary* if for any pair of terms s and t a set $\mu U_E(s,t)$ exists that contains at most finitely many elements. It is called *infinitary* if for any pair of terms s and t a set $\mu U_E(s,t)$ exists and there are at least two terms s and t such that there is no finite $\mu U_E(s,t)$. For instance, E_C is known to be finitary [21]; an example for an infinitary theory is considered in Section 2.

The notions of E-unifier, set of E-unifiers, etc. are extended to atoms in the obvious way. An *E-unification procedure* is a procedure that takes two terms s and t (resp. two atoms A and B) as input and generates a subset of $U_E(s,t)$ (resp. $U_E(A,B)$). It is called *complete* iff it generates a complete set of unifiers and *minimal* iff it generates a minimal set of unifiers.

2.3. THE COMPLETION SEMANTICS

Following the direction of [9] and [20], we use a generalization of Clark's completion procedure [4] to define a semantics of normal equational programs. The idea is to consider the set of program clauses that define a predicate p as a complete description of the positive information regarding p. Formally, the completion procedure applied to a set of clauses P is as follows.

DEFINITION 1. Let $p(t_1,\dots,t_n) \leftarrow L_1,\dots,L_m$ be a program clause in P, and let $\bar{y}$ denote a sequence of all variables that occur in this clause. Let $x_1,\dots,x_n$ be pairwise different variables not in $\bar{y}$. Then the *rectified form* of this clause is the formula

$$p(x_1,\dots,x_n) \leftarrow \exists\bar{y}(x_1 \doteq t_1 \wedge\cdots\wedge x_n \doteq t_n \wedge L_1 \wedge\cdots\wedge L_m).$$

Let p be an arbitrary predicate symbol and

$$\begin{aligned} p(x_1,\dots,x_n) &\leftarrow D_1 \\ &\ \vdots \\ p(x_1,\dots,x_n) &\leftarrow D_k \end{aligned}$$

be all clauses in P defining p in rectified form $(k \geqslant 0)$. The *completed definition* of p in P is the formula

$$\forall x_1,\dots,x_n(p(x_1,\dots,x_n) \leftrightarrow D_1 \vee\cdots\vee D_k).$$

(In case $k = 0$ this reduces to $\forall(\neg p(x_1,\dots,x_n))$.) The *completion* P^* of P is the conjunction of the completed definitions of all predicate symbols occurring in the alphabet except for the equality predicate $\doteq$.

As an example, consider the two clauses

$$\begin{array}{l} p(h(a,b)) \\ q(h(z,b)) \leftarrow \neg p(h(z,b)). \end{array} \tag{3}$$

Their completion P^* is the conjunction of the formulas

$$\begin{array}{l} \forall x(p(x) \leftrightarrow x \doteq h(a,b)) \\ \forall x(q(x) \leftrightarrow \exists z(x \doteq h(z,b) \wedge \neg p(h(z,b)))) \end{array} \tag{4}$$

together with all formulas $\forall \bar{x}. \neg r(\bar{x})$ where r is a predicate symbol that does not occur in $\{p, q \doteq\}$.

In order to derive negative information using the completion, it is necessary to extend the standard equality axioms by axioms that allow for proving inequalities. For instance, given the completed formula (4), it is intended one can conclude $\neg p(c)$ be logical consequence, say. This, however, cannot be obtained unless $c \not\doteq h(a,b)$ is provable. Clark added some axiom schemata to the completed formula that allow for proving inequality of two terms whenever these are not unifiable under the empty equational theory [4].

In case of equational theories the original method has to be modified because the direct use of the axiom schemata may lead to undesired results. For instance, Clark's axioms imply $h(a,b) \not\doteq h(b,a)$, which we clearly do not expect on the basis of our equational theory E_C (1). The following generalization, called *unification completeness*, was introduced in [9] and improved in [20]. As in [20], given a substitution $\theta = \{x_1 \mapsto t_1, \dots, x_n \mapsto t_n\}$, we use $eqn(\theta)$ to denote the formula $x_1 \doteq t_1 \wedge \dots \wedge x_n \doteq t_n$:

DEFINITION 2. Let E be an equational theory. A consistent set of formulas E^* is called *unification complete* wrt. E if it consists of the axioms in E, the standard equality axioms, and a number of equational formulas, that is, formulas with $\doteq$ as the only predicate, such that for any two terms s and t with variables $\bar{x}$ the following holds:

1. If s and t are not E-unifiable, then $E^* \models \neg\exists\bar{x}.\ s \doteq t$.
2. If s and t are E-unifiable, then for each complete set of unifiers $cU_E(s,t)$ we have

$$E^* \models \forall\bar{x}\Big(s \doteq t \rightarrow \bigvee_{\theta \in cU_E(s,t)} \exists\bar{y}.\ eqn(\theta)\Big) \tag{5}$$

where $\bar{y}$ denotes the variables that occur in $eqn(\theta)$ but not in $\bar{x}$.[2]

For example, the following extension of Clark's axioms along with the axiom in (1) and the standard equality axioms are unification complete wrt. E_C:

[2] Note that in case of infinitary equational theories the disjunct in (5) may contain infinitely many elements.

$$\begin{array}{ll} f(x_1,\dots,x_n) \neq g(y_1,\dots,y_m) & f,\ g \text{ different function symbols;} \\ & m,n \geqslant 0 \\ f(x_1,\dots,x_n) \doteq f(y_1,\dots,y_n) & f \text{ function symbol } (f \neq h);\ n \geqslant 1 \\ \quad \rightarrow x_1 \doteq y_1 \wedge \dots \wedge x_n \doteq y_n & \\ h(x_1,x_2) \doteq h(y_1,y_2) & \\ \quad \rightarrow x_1 \doteq y_1 \wedge x_2 \doteq y_2 & \\ \quad \vee x_1 \doteq y_2 \wedge x_2 \doteq y_1 & \\ \tau[x] \neq x & \tau[x] \text{ nonvariable term containing } x. \end{array} \tag{6}$$

As pointed out in [20] the use of $\bar{y}$ in Definition 2, which was missing in [9], is necessary due to the fact that E-unifiers might introduce new variables.

It can be easily proved that Clark's axioms form a unification complete theory for the empty equational theory, where standard unification is used. In general, it is not at all clear how to find an appropriate unification complete theory for a given set of equations. On the other hand, we know that such a theory exists whenever it is possible to compute complete sets of E-unifiers wrt. the given equational theory. A formal proof of the following proposition can be found in [7].

PROPOSITION 3. *If E is an equational theory and $\mathcal{P}$ a complete E-unification procedure, there is a unification complete theory wrt. E.*

Throughout this paper we assume the equational theories under consideration to admit a unification complete theory. Given a normal equational logic program (P, E), we call (P^*, E^*) its *completion* whenever P^* completes P as in Definition 1 and E^* is unification complete wrt. E.

2.4. SLDENF-RESOLUTION

Let (P, E) be a normal equational program and G a *normal goal clause*, namely, an expression

$$\leftarrow L_1,\dots,L_m,$$

where each *subgoal* $L_1,\dots,L_m$ is a literal $(m \geqslant 0)$. The *empty goal* (where $m = 0$) is usually denoted by $\square$. In the sequel we define SLDENF-refutations, finitely failed SLDENF-trees, and SLDENF-derivation trees analogously to [20] by simultaneous induction on the *rank*, but with a slight simplifications: To deal with equality subgoals of the form $s \doteq t$, we assume the clause $\doteq (x, x)$ to be implicitly available if the equality predicate occurs in a program clause or the current goal. As a second, more substantial difference to [20], we concentrate on *finitely* failed trees, which consist of finitely many nodes, instead of *generally* failed trees, which consist of (possibly infinitely many) finite branches (see also [9]). We chose the former notion as in general it is impossible to finitely compute a generally failed tree. We will raise this problem later, in Section 3. A *selection* *rule* determines for each goal which literal is selected to proceed. As usual, we assume that negative subgoals are not selected until they are ground. If a goal consists of only nonground negative literals, the derivation is said to *flounder* [12].

DEFINITION 4. An *SLDENF-refutation of rank* r ($r \geqslant 0$) for $(P, E) \cup \{G\}$ via a selection rule R consists of a sequence $G_0, \ldots, G_n$ of normal goals such that $G = G_0$ and $G_n = \Box$, and for each $i = 1, \ldots, n$ the following holds:

1. If the selected literal L_k of $G_{i-1} = \leftarrow L_1, \ldots, L_m$ is positive, there is a new variant $A \leftarrow B_1, \ldots, B_l$ of a program clause in P and an E-unifier θ_i of L_k and A, and G_i is $\leftarrow (L_1, \ldots, L_{k-1}, B_1, \ldots, B_l, L_{k+1}, \ldots, L_m)\theta_i$.
2. If the selected literal L_k of $G_{i-1} = \leftarrow L_1, \ldots, L_m$ is a negative ground literal $\neg A$, there is a finitely failed SLDENF-tree of rank less than r for $(P, E) \cup \{\leftarrow A\}$ via R, and G_i is as G_{i-1} except that it does not contain L_k.

The number n is called the *length* of the SLDENF-refutation. The composition of the substitutions $\theta_1, \ldots, \theta_n$ restricted to the variables in the original goal (i.e., $(\theta_1 \cdots \theta_n)|_{Var(G)}$) is called *computed answer substitution.*

A *finitely failed SLDENF- tree of rank* r ($r \geqslant 0$) for $(P, E) \cup \{G\}$ via a selection rule R is a finite tree such that

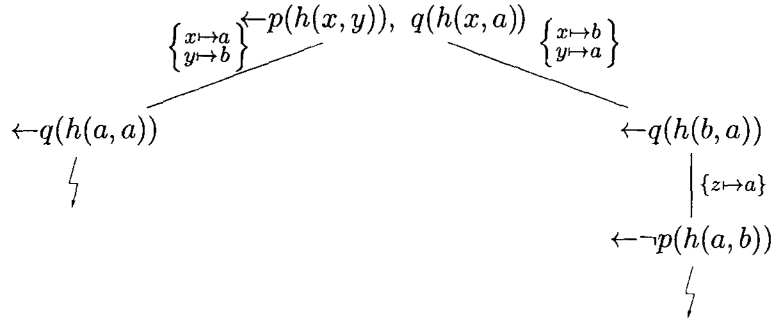
$\leftarrow p(h(x,y)),\ q(h(x,a))$
$\{x \mapsto a, y \mapsto b\}$
$\{x \mapsto b, y \mapsto a\}$
$\leftarrow q(h(a,a))$
$\leftarrow q(h(b,a))$
$\{z \mapsto a\}$
$\leftarrow \neg p(h(a,b))$

Figure 1. A complete SLDENF-derivation tree of rank 1 wrt. the equational program $((3), (1))$, which is finitely failed. The root has two successor nodes corresponding to the two most general E_C-unifiers of $h(x, y)$ and $h(a, b)$. The tree's leftmost branch fails because $q(h(a, a))$ does not E_C-unify with $q(h(z, b))$, while its rightmost branch fails because of the existence of an SLDENF-refutation of rank 0 for the goal $\leftarrow p(h(a, b))$. The reader is invited to verify that indeed $\neg\exists x, y(p(h(x, y)) \wedge q(h(x, a)))$ is a logical consequence of the completion $((3)^*, (1)^*)$ (see (4) and (6)).

1. Each node is labeled with a nonempty normal goal, and the root is labeled with G.
2. For each leaf node $\leftarrow L_1, \ldots, L_k, \ldots, L_m$ such that L_k is selected,
   (a) If L_k is a positive literal, it does not E-unify with the head of a new variant of any program clause in P.

   (b) If L_k is a negative ground literal $\neg A$, there exists an SLDENF-refutation of rank less than r for $(P, E) \cup \{\leftarrow A\}$ via R.
3. For each inner node $\leftarrow L_1, \ldots, L_k, \ldots, L_m$ such that L_k is selected,
   (a) If L_k is a positive literal, for each program clause in P let $A \leftarrow B_1, \ldots, B_l$ be a new variant and $cU(L_k, A)$ a complete set of E-unifiers of L_k and A.[3] For each $\theta \in cU(L_k, A)$, the node labeled with $\leftarrow(L_1, \ldots, L_{k-1}, B_1, \ldots, B_l, L_{k+1}, \ldots, L_m)\theta$ is a child of this inner node.
   (b) If L_k is a negative ground literal $\neg A$, there exists a finitely failed SLDENF -tree of rank less than r for $(P, E) \cup \{\leftarrow A\}$ via R, and the only child of the inner node is labeled with $\leftarrow L_1, \ldots, L_{k-1}, L_{k+1}, \ldots, L_m$.

A *complete SLDENF-derivation tree* is a finitely failed SLDENF-tree but may consist of infinitely many branches, possibly of infinite length, and a leaf can also be labeled with the empty goal (denoting a success branch) or with a goal containing only nonground negative literals (denoting a branch that flounders).

As usual, the *depth* of a finitely failed SLDENF-tree (resp. an SLDENF-derivation tree) is defined as the length of the longest path from the root node to a leaf. □

To illustrate these definitions, Figure 1 depicts an example based on the equational program that consists of the clauses (3) along with the theory of commutativity E_C (1). It is obvious that Shepherdson's soundness result [20] can be adapted to our definition:

THEOREM 5. *Let (P, E) be a normal equational program and (P^*, E^*) its completion. Furthermore, let $G = \leftarrow L_1, \ldots, L_m$ be a normal goal $(m \geqslant 0)$. Then*

1. *if $(P, E) \cup \{G\}$ has an SLDENF-refutation with computed answer substitution θ, then $P^* \cup E^* \models \forall((L_1 \wedge \cdots \wedge L_m)\theta)$, and*
2. *if $(P, E) \cup \{G\}$ has a finitely failed SLDENF-tree, then $P^* \cup E^* \models \neg\exists(L_1 \wedge \cdots \wedge L_m)$.*

3. A Completeness Result

Completeness of SLDENF-resolution wrt. the extended completion semantics of Subsection 2.3 means that every *correct answer* can be computed by this calculus: A substitution θ is a correct answer to a goal $G = \leftarrow L_1, \ldots, L_m$ in conjunction with a program (P, E) iff the domain of θ is a subset of $\mathcal{V}ar(G)$ and $P^* \cup E^* \models \forall((L_1 \wedge \cdots \wedge L_m)\theta)$. We already know from nonequational logic programming that a main obstacle for obtaining completeness of SLDENF-resolution is that the completion P^* of a set of normal program clauses P might

[3] Although not necessary it is usually desirable to use a minimal set of unifiers whenever one exists. Especially in case of finitary equational theories we assume $cU(L_k, A)$ to be finite in any case.

be inconsistent. Inconsistency of the completion implies that any substitution is correct for any goal; this, of course, cannot be achieved by applying SLDENF-resolution. Hence, in view of our intended completeness theorem, which concerns hierarchical equational programs, we have to ensure that their completion is always consistent.

LEMMA 6. *Let (P, E) be a hierarchical equational program and (P^*, E^*) its completion. Then $P^* \cup E^*$ is satisfiable.*

Proof. We prove this by an inductive construction of a model $\mathcal{I}$ whose universe consists of the congruence classes $[t]_E$ determined by the finest E-congruence relation on the set of ground terms. Let $\mathcal{I}_0 = \{[s]_E \doteq [t]_E \mid s, t$ ground terms$\}$. Then it is easy to see that $\mathcal{I}_0$ is a model of E^*. Now, let *lev* be an adequate level mapping, that is, one that satisfies the hierarchy property as regards P. Let $n \in \mathbb{N}$ be the maximal level, and let $P^*|_k \subseteq P^*$ contain the completed definitions for all predicates with a level not greater than k $(1 \leqslant k \leqslant n)$; hence, $P^*|_k = \{\forall \bar{x}(p(\bar{x}) \leftrightarrow \Phi) \in P^* \mid lev(p) \leqslant k\}$. For each $k = 1, \dots, n$ construct a model $\mathcal{I}_k$ as follows:

$$\begin{aligned}\mathcal{I}_k := \mathcal{I}_{k-1} \cup \Big\{&p([t_1]_E, \dots, [t_n]_E) \mid lev(p) = k,\\ &\forall x_1, \dots, x_n(p(x_1, \dots, x_n) \leftrightarrow \Phi) \in P^*, \text{ and}\\ &\Phi\{x_1 \mapsto t_1, \dots, x_n \mapsto t_n\} \text{ is true in } \mathcal{I}_{k-1}\Big\}.\end{aligned}$$

It is easy to verify that each $\mathcal{I}_k$ is a model of the corresponding set of formulas $P^*|_k \cup E^*$; hence, $\mathcal{I}_n$ is a model of $P^* \cup E^*$. □

Aside from requiring the programs under consideration to have a consistent completion, from the special case of nonequational logic programming we know two further conditions that are necessary to obtain completeness. In what follows, we discuss how equational theories cause additional problems in meeting these conditions.

Finite derivation trees. To obtain a general completeness result for SLD(E)NF-resolution wrt. the (extended) completion semantics, one has to ensure finiteness of any complete derivation tree [4, 1]. From nonequational logic programming we know that this is guaranteed in the case of hierarchical programs. In the case of underlying equational theories, however, the theory itself may lead to infinite complete derivation trees. As an example, consider the clauses

$$\begin{aligned}&p(b)\\ &q(x, x) \leftarrow p(x)\\ &r \leftarrow g(h(a, y), h(y, a))\end{aligned} \tag{7}$$

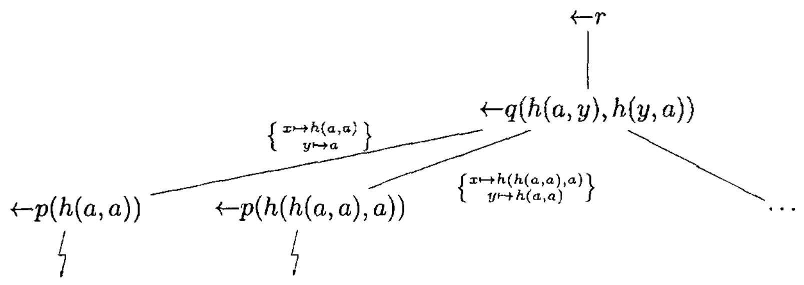
$\leftarrow r$
$\leftarrow q(h(a,y),h(y,a))$
$\{x \mapsto h(a,a), y \mapsto a\}$
$\{x \mapsto h(h(a,a),a), y \mapsto h(a,a)\}$
$\leftarrow p(h(a,a))$
$\leftarrow p(h(h(a,a),a))$
$\ldots$

Figure 2. An infinite SLDENF-tree for $(P, E_A) \cup \{\leftarrow r\}$ – where P consists of the clauses (7) and E_A describes the law of associativity.

along with the equational theory $E_A = \{h(h(x,y),z) \doteq h(x,h(y,z))\}$ defining associativity, which is known to be infinitary [13]. This program is hierarchical. Clearly, its completion implies that q is true only in case $q(b,b)$ and, hence, $\neg r$ is a consequence of the completion. However, we cannot find an SLDENF-refutation for the goal $\leftarrow \neg r$ since unifying the literals $q(h(a,y),h(y,a))$ and $q(x,x)$ wrt. associativity yields the infinite minimal set of unifiers $\{\{x \mapsto h(a,a), y \mapsto a\}, \{x \mapsto h(h(a,a),a), y \mapsto h(a,a)\}, \ldots\}$ (see Fig. 2). Hence, besides the logic program being hierarchical, we have to require that the underlying equational theory is finitary when finiteness of derivation trees is guaranteed.[4]

Nonfloundering goals. A second condition towards completeness of SLDENF-resolution concerns the problem of floundering, which has to be avoided [4, 17]. Again, from nonequational logic programming we know a sufficient syntactic criterion: A clause $A \leftarrow L_1, \ldots, L_m$ (resp. a goal $\leftarrow L_1, \ldots, L_m$) is called *allowed* if every variable that occurs in the clause (resp. the goal) occurs in at least one positive literal of the body $L_1, \ldots, L_m$ (see [12], e.g.). It is obvious that an allowed goal does not flounder immediately because, if it contains variables, it must contain a positive literal. In addition, a derivation step applied to an allowed goal and an allowed program yields again an allowed goal in case of standard unification. Hence, no derivation can flounder under this condition. In general, however, this criterion does not always apply. As a counterexample, consider the single program clause

$$p(0) \tag{8}$$

along with the (nonregular) equational theory E_0 (2). Both this program clause and the goal $\leftarrow p(h(x,y)), \neg q(x)$ are allowed. The leftmost subgoal can be solved

[4] At this point it is important to realize the difference between *finitely* and *generally* failed SLDENF-trees (see Subsection 2.4). Had we adopted Shepherdson's definition, there would be no need to additionally restrict the equational theories to finitary ones (note that the tree in Figure 2 is generally failed!). On the other hand, it is in general impossible to finitely compute generally failed trees; hence, a completeness result based on general failure would not imply that a refutation of a valid formula can be obtained in a finite number of computation steps. In case of finitary equational theories, both concepts – finite and general failure – coincide.

by applying the E_0-unifier $\{y \mapsto 0\}$ of the two terms $h(x,y)$ and 0. This, however, yields $\leftarrow\neg q(x)$ as the new goal, which is no longer allowed and flounders immediately.

The problem encountered in this example is that two terms might be E-equivalent although they include different variables. This might destroy the property that the application of an allowed clause preserves allowance of a goal. Hence, to guarantee nonfloundering, we have to find an additional condition to be met by the underlying equational theory. The following lemma shows that the concept of regularity (see Subsection 2.1) forms an adequate condition.

LEMMA 7. *Let E be a regular equational theory, $A \leftarrow B_1, \dots, B_l$ an allowed clause, $G = \leftarrow L_1, \dots, L_k, \dots, L_m$ an allowed goal ($m \geqslant 1$), and θ an E-unifier of L_k and A. Then $G' = \leftarrow(L_1, \dots, L_{k-1}, B_1, \dots, B_l, L_{k+1}, \dots, L_m)\theta$ is allowed.*

Proof. The applied clause being allowed implies that $\leftarrow (B_1, \dots, B_l)\theta$ is allowed. Furthermore, $\leftarrow (L_1, \dots, L_k, \dots, L_m)\theta$ is allowed as G itself is assumed to be allowed. In order to gain allowance of both together but without the literal $L_k\theta$, each $x \in Var(L_k\theta)$ should occur in a positive subgoal of G'. From $L_k\theta =_E A\theta$ and E being regular, we know that $x \in Var(L_k\theta)$ implies $x \in Var(A\theta)$. This and the fact that the applied clause is allowed guarantees that for each $x \in Var(L_k\theta)$ there is some positive $B_i\theta$ ($1 \leqslant i \leqslant l$) containing x. □

In the remainder of this section we take the preceding discussion into account by concentrating on hierarchical and allowed programs, that is, programs that contain only allowed clauses, along with an underlying equational theory that is both finitary and regular. Note that these two properties do not depend on each other; for example, the infinitary theory of associativity used in Figure 2 is regular, whereas unification wrt. the nonregular theory E_0 is finitary.

To generalize the classical completeness theorem, we need the following notion of decidability and a certain measure: A complete SLDENF-derivation tree is called *decidable* if it is finite and contains no branch that flounders. Given a hierarchical normal equational program (P, E), let κ be the maximal number of literals occurring in the body of a clause in P plus 1. Furthermore, let lev be an adequate level mapping, that is, one that satisfies the hierarchy property. We define the following *measure*:

1. $\mu(p(t_1, \dots, t_n)) = \kappa^{2 \cdot lev(p)}$ for each atom,
2. $\mu(\neg A) = \mu(A) + 1$ for each negative literal, and
3. $\mu(\leftarrow L_1, \dots, L_m) = \sum_{i=1}^{m} \mu(L_i)$ for each normal goal ($m \geqslant 0$).

It is easy to verify that for each clause $A \leftarrow B_1, \dots, B_l$ in a hierarchical program the relation $\mu(A) > \sum_{i=1}^{l} \mu(B_i)$ holds.

THEOREM 8. *Let (P, E) be a hierarchical and allowed normal equational program such that E is both finitary and regular. Furthermore, let G be an allowed goal and R a selection rule. Then each complete SLDENF-derivation tree for $(P, E) \cup \{G\}$ wrt. R is decidable.*

Proof. By induction on the measure $\mu(G)$ we show that each complete SLDENF-derivation tree for $(P, E) \cup \{G\}$ wrt. R is decidable and of rank $\mu(G)$. The base case is trivial because $\mu(G) = 0$ implies $G = \square$.

Let $G = \leftarrow L_1, \ldots, L_m$ $(m \geqslant 1)$, and assume that the claim holds for each allowed goal with measure smaller than $\mu(G)$. Lemma 7 ensures that no SLDENF-derivation for $(P, E) \cup \{G\}$ flounders. Let L_k be the selected literal $(1 \leqslant k \leqslant m)$.

1. If L_k is an atom, then
   (a) if there is no variant of a clause in P such that its head is E-unifiable with L_k, then the only SLDENF-derivation tree consists in the node G. This tree is decidable and of rank 0, hence also of rank $\mu(G)$;
   (b) otherwise each complete SLDENF-derivation tree consists of the root G and has a finite number of successor nodes
   $$G' = \leftarrow (L_1, \ldots, L_{k-1}, B_1, \ldots, B_l, L_{k+1}, \ldots, L_m)\theta,$$
   since P contains finitely many clauses and E is finitary. From
   $$\mu(L_k) = \mu(A) > \sum_{i=1}^{l} \mu(B_i)$$
   it follows that $\mu(G') < \mu(G)$ for each successor G'. Thus, the induction hypothesis implies that each complete SLDENF-derivation tree for each such $(P, E) \cup \{G'\}$ wrt. R is decidable and of rank less than $\mu(G)$. Hence, each SLDENF-derivation tree for $(P, E) \cup \{G\}$ wrt. R is also decidable and of rank $\mu(G)$.
2. Otherwise L_k is a negative ground literal $\neg A$ and $\mu(A) < \mu(G)$, since $\mu(A) = \mu(L_k) - 1$. Following the induction hypothesis, each complete SLDENF-derivation tree for $(P, E) \cup \{\leftarrow A\}$ wrt. R is decidable and of rank less than $\mu(G)$. Decidability along with soundness of SLDENF-resolution (Theorem 5) and consistency of $P^* \cup E^*$ (Lemma 6) implies that either each such tree is finitely failed or else each such tree contains a success branch.
   (a) In the former case, each complete SLDENF-derivation tree for $(P, E) \cup \{G\}$ wrt. R consists of the root G and its single successor $G' = \leftarrow L_1, \ldots, L_{k-1}, L_{k+1}, \ldots, L_m$. From $\mu(L_k) > 0$ we conclude that $\mu(G') < \mu(G)$; hence, each complete SLDENF-derivation tree for $(P, E) \cup \{G'\}$ is decidable and especially of rank $\mu(G)$ because of the induction hypothesis (note that G' is allowed as it is like G except for the ground literal L_k). Altogether, each complete SLDENF-tree for $(P, E) \cup \{G\}$ wrt. R is decidable and of rank $\mu(G)$.
   (b) In the latter case, each success branch in the SLDENF-derivation tree for $(P, E) \cup \{\leftarrow A\}$ constitutes an SLDENF-refutation of rank less than $\mu(G)$. Hence, the only SLDENF-derivation tree for $(P, E) \cup \{G\}$ wrt. R consists in the node G – this tree is decidable and of rank $\mu(G)$. $\square$

We are now in a position to prove the intended generalization of the classical completeness result [4, 1]. An answer substitution θ to a goal G is called *ground* iff $G\theta$ is variable-free.

THEOREM 9. *Let (P, E) be a hierarchical and allowed normal equational program such that E is both finitary and regular. Furthermore, let G be an allowed goal and R a selection rule. If θ is a correct ground answer to G, there exists an SLDENF-refutation for $(P, E) \cup \{G\}$ wrt. R with computed answer substitution $\tilde{\theta} =_E \theta$.*

Proof. The proof is by induction on the measure $\mu(G)$. In case $\mu(G) = 0$, ε (the empty substitution) is the only correct answer to $G = \Box$ wrt. (P, E), and it is also the computed answer.

Let $G = \leftarrow L_1, \ldots, L_k, \ldots, L_m$ $(m \geqslant 1)$, and assume that the claim holds for all allowed goals G' such that $\mu(G') < \mu(G)$. As θ is a correct ground answer to G, we know that

$$P^* \cup E^* \models (L_1 \wedge \cdots \wedge L_k \wedge \cdots \wedge L_m)\theta. \tag{9}$$

Let L_k be the literal selected by R $(1 \leqslant k \leqslant m)$.

1. If L_k is positive, then according to Theorem 8 we can find a decidable complete SLDENF-derivation tree $\mathcal{B}$ for $(P, E) \cup \{G\theta\}$ wrt. some selection rule that selects $L_k\theta$ in $G\theta$. From (9) and soundness of SLDENF-resolution it follows that this tree cannot be finitely failed because of the consistency of $P^* \cup E^*$ (Lemma 6). Hence, as $\mathcal{B}$ is decidable, it must include a success branch; that is, we can find an SLDENF-refutation for $(P, E) \cup \{G\theta\}$ wherein $L_k\theta$ is selected at the beginning. Let $A \leftarrow B_1, \ldots, B_l$ be the variant of a clause that is used in conjunction with the E-unifier σ in the first step of this refutation, which implies $A\sigma =_E L_k\theta\sigma = L_k\theta$ (recall that $L_k\theta$ is ground). Without loss of generality we assume that the domain of σ is restricted to $\mathcal{V}ar(A)$ and that σ does not introduce variables occurring in G. The resulting goal is

   $$\leftarrow (L_1\theta, \ldots, L_{k-1}\theta, B_1, \ldots, B_l, L_{k+1}\theta, \ldots, L_m\theta)\sigma.$$

   Since this goal occurs within the refutation for $(P, E) \cup \{G\theta\}$, soundness of SLDENF-resolution implies

   $$P^* \cup E^* \models \exists((L_1 \wedge \cdots \wedge L_{k-1} \wedge B_1 \wedge \cdots \wedge B_l \wedge L_{k+1} \wedge \cdots \wedge L_m)\theta\sigma) \tag{10}$$

   (note that $B_i\sigma = B_i\theta\sigma$ $(1 \leqslant i \leqslant l)$, since the goal, G, and the applied clause do not share variables). Now, let $\theta_1 = \theta|_{\mathcal{V}ar(L_k)}$ and $\theta_2 = \theta|_{\mathcal{V}ar(G)\setminus\mathcal{V}ar(L_k)}$ (this implies $\theta = \theta_1\theta_2$ because of θ_1 is a ground substitution). From $L_k\theta =_E A\sigma$ it follows $L_k\theta_1\sigma =_E A\theta_1\sigma$ because of $\mathcal{V}ar(L_k\theta_1) = \{\}$ and $\mathcal{V}ar(L_k) \cap$

$Var(A) = \{\}$. In other words, $\theta_1\sigma$ is an E-unifier of L_k and A. Hence, we can apply an SLDENF-step to $(P, E) \cup \{G\}$ wrt. R which yields

$$G' = \leftarrow(L_1, \ldots, L_{k-1}, B_1, \ldots, B_l, L_{k+1}, \ldots, L_m)\theta_1\sigma. \qquad (11)$$

From the above assumption concerning σ, the fact that $Var(G) \cap Var(A) = \{\}$, and θ_2 being a ground substitution, we conclude that $\theta_1\sigma\theta_2 = \theta_1\theta_2\sigma = \theta\sigma$. Furthermore, (10) guarantees the existence of some ground instance $\theta\sigma\rho$ of the literals occurring in (11) such that their conjunction is a logical consequence of the completion. Hence, since $\theta\sigma\rho = \theta_1\sigma\theta_2\rho$, we can find a substitution ρ such that $\theta_2\rho$ is a correct ground answer to G'. Since $\mu(G') < \mu(G)$ and G' is allowed (Lemma 7), we can apply the induction hypothesis to find a computed answer $\tilde{\rho}$ for $(P, E) \cup \{G'\}$ wrt. R such that $\tilde{\rho} =_E \theta_2\rho$. Combined with the first step from G to G' (11), this yields the computed answer $\theta_1\sigma\tilde{\rho}$ for $(P, E) \cup \{G\}$ wrt. R, which is E-equivalent to θ:

$$(\theta_1\sigma\tilde{\rho})|_{Var(G)} =_E (\theta_1\sigma\theta_2\rho)|_{Var(G)} = (\theta_1\theta_2\sigma\rho)|_{Var(G)} = \theta_1\theta_2 = \theta.$$

2. If L_k is a negative ground literal $\neg A$, then (9) implies $P^* \cup E^* \models \neg A$. According to Theorem 8 we can find a decidable complete SLDENF-derivation tree for $(P, E) \cup \{\leftarrow A\}$ wrt. R. This tree cannot contain a success branch, since otherwise soundness of SLDENF-resolution implies $P^* \cup E^* \models A$, which contradicts the consistency of $P^* \cup E^*$. Hence, this tree is finitely failed. Moreover, (9) implies

$$P^* \cup E^* \models (L_1 \wedge \cdots \wedge L_{k-1} \wedge L_{k+1} \wedge \cdots \wedge L_m)\theta;$$

that is, θ is a correct ground answer to $G' = \leftarrow L_1, \ldots, L_{k-1}, L_{k+1}, \ldots, L_m$ wrt. (P, E). Since $\mu(G') < \mu(G)$ and G' is allowed because of the allowance of G, the induction hypothesis implies the existence of an SLDENF-refutation for $(P, E) \cup \{G'\}$ wrt. R with computed answer $\tilde{\theta} =_E \theta$. Combined with the SLDENF-tree for $(P, E) \cup \{\leftarrow A\}$ this refutation constitutes an SLDENF-refutation for $(P, E) \cup \{G\}$ wrt. R with $\tilde{\theta}$ as its computed answer substitution. □

4. Conclusion

By investigating important aspects concerning the completeness of the SLDENF-resolution principle for normal equational logic programs wrt. an extended completion semantics, we have extended the work of Shepherdson, who proved soundness of this calculus [20]. We have illustrated that the classical completeness theorem known from nonequational logic programming with negation [4, 1] can be generalized only to some extent. As a concrete result, we have proved completeness for hierarchical and allowed programs, provided the underlying equational theories meet two conditions: they are finitary and regular. Moreover, these conditions turn out to be strict in a certain sense:

- Whenever E is an infinitary equational theory with unification complete E^*, it is possible to construct a hierarchical program (P, E) and a generally but not finitely failed SLDENF-tree in the spirit of (7) and Figure 2: Let s, t be two E-unifiable terms such that no finite $\mu U_E(s, t)$ exists, and let P denote the program clauses

$$\begin{aligned} &q(x, x) \leftarrow p(x) \\ &r \leftarrow q(s, t) \end{aligned}$$

  then $P^* \cup E^* \models \neg r$ but $(P, E) \cup \{\leftarrow \neg r\}$ has no SLDENF-refutation.

- Whenever E is an equational theory such that two terms s, t exist with $s =_E t$ but $Var(s) \neq Var(t)$, it is possible to construct an allowed program (P, E) along with an allowed but floundering goal in the spirit of (8): Without loss of generality assume there is some $x \in Var(s) \setminus Var(t)$. Let σ be a substitution that grounds all variables in s and t except x. Let P denote the allowed program clause

$$p(t\sigma)$$

  This clause can be applied to the allowed goal $\leftarrow p(s\sigma), \neg q(x)$ using the E-unifier $\{\varepsilon\}$, and the resulting goal flounders immediately.

The restriction to hierarchical programs in our completeness theorem is adopted from insight into nonequational logic programming. For instance, a simple counterexample shows that SLDENF-resolution is incomplete for the more general class of stratified programs [12, 2].

Finally, our result concerns only ground answer substitutions. Regarding the special case, we know that correct answers to allowed nonequational programs and allowed goals are necessarily ground, provided the completion of the program is consistent [19]. It is obvious that this result cannot be generalized to arbitrary equational theories (just recall the program (8) along with the theory E_0 (2), whose completion entails $\forall x. p(h(x, 0))$). However, we believe that it continues to hold in case of regular theories, which then would make the groundedness condition in Theorem 9 redundant, but a detailed proof remains to be done.

Acknowledgements

The author thanks Steffen Hölldobler for valuable remarks on a draft version of this paper. The author was partially supported by ESPRIT within basic research action MEDLAR-II under grant. no. 6471 and by the German Research Community (DFG) within project KONNEKTIONSBEWEISER under grant no. Bi 228/6-2.

References

1. Apt, K. R., Blair, H. A., and Walker, A.: Towards a theory of declarative knowledge, in J. Minker (ed.), *Foundations of Deductive Databases an Logic Programming*, Chapter 2, Morgan Kaufmann Publishers Inc., 1987, pp. 89–148.
2. Apt, K. R. and Bol, R.: Logic programming and negation: a survey, *J. Logic Programming* **19/20** (1994), 9–71.
3. Baader, F. and Siekmann, J. H.: Unification theory, in D. M. Gabbay, C. J. Hogger, and J. A. Robinson (eds), *Handbook of Logic in Artificial Intelligence and Logic Programming*, Oxford University Press, 1993.
4. Clark, K. L.: Negation as failure, in H. Gallaire and J. Minker (eds), *Logic and Data Bases*, Plenum, New York, 1978, pp. 293–322.
5. Gallier, J. H. and Raatz, S.: Extending SLD-resolution to equational horn clauses using E-unification, *J. Logic Programming* **6** (1989), 3–44.
6. Hölldobler, S.: *Foundations of Equational Logic Programming*, LNAI 353, Springer, 1989.
7. Hölldobler, S. and Thielscher, M.: Computing change and specificity with equational logic programs, *Ann. Mathematics and Artificial Intelligence* **14**(1) (1995), 99–133.
8. Jaffar, J., Lassez, J.-L., and Lloyd, J.: Completeness of the negation as failure rule, in A. Bundy (ed.), *Proc. Int. Joint Conf. on Artificial Intelligence (IJCAI)*, Karlsruhe, Germany, 1983, pp. 500–506.
9. Jaffar, J., Lassez, J.-L., and Maher, M. J.: A theory of complete logic programs with equality, *J. Logic Programming* **1**(3) (1984), 211–223.
10. Jaffar, J. and Maher, M. J.: Constraint logic programming: a survey, *J. Logic Programming* **19/20** (1994), 503–581.
11. Kunen, K.: Signed data dependencies in logic programs, *J. Logic Programming* **7** (1989), 231–246.
12. Lloyd, J. W.: *Foundations of Logic Programming*, Series Symbolic Computation, 2nd extended edition, Springer, 1987.
13. Plotkin, G.: Building in equational theories, *Machine Intelligence* **7** (1972), 73–90.
14. Robinson, J. A.: A review of automatic theorem proving, in *Annual Symposium in Applied Mathematics*, American Mathematical Society, 1967, pp. 1–18.
15. Sato, T.: Completed logic programs and their consistency, *J. Logic Programming* **9** (1990), 33–44.
16. Shepherdson, J. C.: Negation as failure: a comparison of Clark's completed data base and Reiter's closed world assumption, *J. Logic Programming* **1** (1984), 51–79.
17. Shepherdson, J. C.: Negation as failure II, *J. Logic Programming* **3** (1985), 185–202.
18. Shepherdson, J. C.: Negation in logic programming for general logic programs, in J. Minker (ed.), *Foundations of Deductive Databases and Logic Programming*, Chapter 1, Morgan Kaufmann Publishers Inc., 1987, pp. 19–88.
19. Shepherdson, J. C.: Correct answers to allowed programs and queries are ground, *J. Logic Programming* **11** (1991), 359–362.
20. Shepherdson, J. C.: SLDNF-resolution with equality, *J. Automated Reasoning* **8** (1992), 297–306.
21. Siekmann, J. H.: Unification of commutative terms, in *Proc. Int. Symp. on Symbolic and Algebraic Manipulation (EUROSAM)*, Marseille, France, June 1979, Springer LNCS 72, pp. 531–545.
22. Siekmann, J. H.: Unification theory, *J. Symbolic Computation* **7** (1989), 207–274. Special Issue on Unification.
23. Stroetmann, K.: A completeness result for SLDNF-resolution, *J. Logic Programming* **15** (1993), 337–355.