Contributions to the Theory of Logic Programming

KRZYSZTOF R. APT

Erasmus University Rotterdam, Rotterdam, The Netherlands

AND

M. H. VAN EMDEN

University of Waterloo, Waterloo, Ontario, Canada

Abstract. Horn clauses of first-order predicate logic can be regarded as a high-level programming language when SLD-resolution, a special-purpose resolution theorem prover, is used as interpreter. Consequently, the semantics of Horn clauses can be studied both by model-theoretic and fixpoint methods (in the sense of Scott). This possibility is exploited here by identifying the least (greatest) fixpoint with a least (greatest) model. Successful termination of SLD-resolution is characterized by least fixpoints. A semantic characterization of finite failure of SLD-resolution is given, which coincides with the greatest fixpoint only for a special case of clauses. It is shown that nondeterministic flowchart schemata of bounded nondeterminacy are modeled by this special case; the connection between finite failure and greatest fixpoint is then used to give a semantic characterization of termination, blocking, and nontermination of such flowchart schemata.

Categories and Subject Descriptors: F.3.1 **[Logics and Meanings of Programs]**: Specifying and Verifying and Reasoning about Programs—*logics of programs*; F.4.1 **[Mathematical Logic and Formal Languages]**: Mathematical Logic—*logic programming*; I.2.3 **[Artificial Intelligence]**: Deduction and Theorem Proving—*logic programming*

General Terms: Languages, Theory

Additional Key Words and Phrases: Fixpoint semantics, greatest fixpoints, continuity, completeness, nondeterministic flowchart schemata, blocking, termination

1. *Introduction*

This paper is a continuation of the investigation begun in [19], where various approaches to the semantics of predicate logic regarded as a programming language were discussed. Using the analogies provided by logic programming [6, 11, 12], the model-theoretic semantics of Horn sentences of first-order predicate logic was formulated in terms of a fixpoint semantics. In the present paper we exploit further the application of fixpoints of transformations to the semantics of Horn sentences by relating it to various properties of what we call SLD-resolution (*SL* resolution for *D*efinite clauses, first described in [11]). Among other results we prove by fixpoint techniques the soundness and strong completeness of SLD-resolution; the latter result is shown to be a consequence of the continuity of the transformation. Similar results

Authors' present addresses: K. R. Apt, LITP, Université Paris 7, 2 Place Jussieu, 75221 Paris, France; M. H. van Emden, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1.

have been obtained by Clark [5]. The first completeness proof is due to Hill [9]. This proof does not apply the fixpoint technique.

There exists an extensive literature on the use of least fixpoints in characterizing program behavior. The literature on the application of greatest fixpoints is much smaller; see [2] for some important results. The present paper shows that similar results can be obtained in the framework of Horn-clause logic with SLD-resolution as computational mechanism. Admittedly, the programs concerned are logic programs, but they have simple relationships with conventional models of programs or databases [17, 18]. These results are based on the fixpoint and semantical characterizations of finite failure of SLD-resolution; these characterizations are significant independently of the particular applications given here.

The interest of greatest fixpoints lies in the fact that they are not in all respects dual to least fixpoints. A transformation T, as typically associated with a Horn sentence or with a program, has a least fixpoint which is equal to the union of the finite powers of T applied to the least element of the universe; this property may be referred to as union-continuity. The T's used in program semantics typically are union-continuous, and ours are no exception. The dual property of intersection-continuity, which is the equality of the greatest fixpoint of T to the intersection of all finite powers of T applied to the greatest element of the universe, is typically not satisfied. We prove intersection-continuity for the T associated with a set of clauses which represents a flowchart schema of finite indeterminacy. Together with our Theorem 7.11, which shows that finite failure of SLD-resolution computes the complement of the above-mentioned intersection, this result can be applied to obtain a semantic characterization of nontermination and blocking of flowchart schemata of finite indeterminacy.

The paper is organized as follows. In Sections 2 and 3 we gather the basic results and definitions concerning fixpoints and logic in clausal form. The semantics of logic and the transformation T associated with Horn sentences are introduced in Section 4. In Section 5 SLD-refutations are introduced, and soundness and completeness of the method is proved. SLD-resolution is discussed in Section 6, where its strong completeness is proved. Finite failure of the SLD-resolution and its characterization using the greatest fixpoint of the transformation T is considered in Section 7. Section 8 is devoted to a semantical characterization of finite failure. Finally, in Section 9 we apply our results to a semantic characterization of some aspects of the behavior of nondeterministic flowchart schemata (see [17]).

2. *Basic Results on Fixpoints*

Let L be a complete lattice with set B, order relation $\subseteq$, greatest lower bound operation $\bigcap$, and least upper bound operation $\bigcup$. A function $T: B \to B$ is said to be *monotone* if $x_1 \subseteq x_2$ implies that $Tx_1 \subseteq Tx_2$, for any x_1 and x_2 in B.

Although, by the completeness of L, any subset S of B has a glb and an lub in B, S does not necessarily contain either of them. Subsets that do are of special interest. For example, $H = \{x : x \subseteq Tx\}$, where T is monotone, contains $h = \bigcup H$ because

$$
\begin{aligned}
h \supseteq x \text{ for any } x \in H &\Rightarrow (\text{monotonicity of } T),\\
Th \supseteq Tx \text{ for any } x \in H &\Rightarrow (\text{definition of } H),\\
Th \supseteq x \text{ for any } x \in H &\Rightarrow (\text{definition of } \bigcup),\\
Th \supseteq \bigcup H \Rightarrow Th \supseteq h &\Rightarrow h \in H.
\end{aligned}
$$

Likewise, for monotone T, $G = \{x : x \supseteq Tx\}$, and $g = \bigcap G$, we have $g \in G$. The *least fixpoint* of T (lfp(T), for short) is the least element x of B such that $Tx = x$.

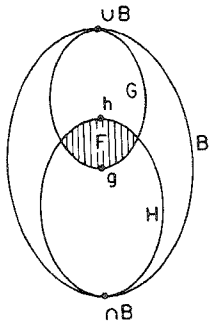

FIGURE 1

The *greatest fixpoint* of T (gfp(T), for short) is the greatest element x of B such that $Tx = x$. Both elements exist, as the following theorem shows.

THEOREM 2.1 (THE KNASTER–TARSKI FIXPOINT THEOREM). *A monotone function T has a greatest and a least fixpoint:*

$$h' = Th', \quad \text{where} \quad h' = \bigcup\{x : x = Tx\},$$
$$g' = Tg', \quad \text{where} \quad g' = \bigcap\{x : x = Tx\}.$$

PROOF. We shall show that $h = Th$, where $h = \bigcup\{x : x \subseteq Tx\}$. We prove that $h = h'$. We already showed that $h \subseteq Th$. It remains to show that $h \supseteq Th : h \subseteq Th \Rightarrow Th \subseteq T(Th) \Rightarrow Th \in H \Rightarrow Th \subseteq h$. $Th = h \Rightarrow h \subseteq h'$, by definition of h'. $h' \subseteq h$, because $\{x : x = Tx\} \subseteq \{x : x \subseteq Tx\}$. The existence of the least fixpoint may be shown in a similar way. □

Figure 1 may help to visualize the situation; $F = \{x : x = Tx\}$.

We will need unions and intersections of powers of T. Because the exponents in those powers may have to go beyond the natural numbers, we define the following *ordinal powers* of T:

$$\begin{aligned}
T\uparrow 0 &= \bigcap B, \\
T\uparrow n &= T(T\uparrow(n-1)) && \text{if } n \text{ is a successor ordinal,} \\
&= \bigcup\{T\uparrow k : k < n\} && \text{if } n \text{ is a limit ordinal;} \\
T\downarrow 0 &= \bigcup B, \\
T\downarrow n &= T(T\downarrow(n-1)) && \text{if } n \text{ is a successor ordinal,} \\
&= \bigcap\{T\downarrow k : k < n\} && \text{if } n \text{ is a limit ordinal.}
\end{aligned}$$

THEOREM 2.2. *For any ordinal n,*

$$T\uparrow n \subseteq lfp(T) \quad \text{and} \quad T\downarrow n \supseteq gfp(T).$$

There exist ordinals n_1 and n_2 such that

$$T\uparrow n_1 = lfp(T) \quad \text{and} \quad T\downarrow n_2 = gfp(T).$$

This theorem is well known in various areas of mathematics. Within theoretical computer science it has been popularized in [10], where its proof can be found.

In the sequel we shall often need the following corollary to this theorem.

COROLLARY 2.3

$$T\uparrow\omega \subseteq T\downarrow\omega.$$

PROOF. $T\uparrow\omega \subseteq \text{lfp}(T) \subseteq \text{gfp}(T) \subseteq T\downarrow\omega$. □

Sometimes a property of functions stronger than monotonicity is considered. A function $T:B \to B$ is said to be *continuous* if for every chain $x_1 \subseteq x_2 \subseteq \cdots$ of elements of B, $T(\bigcup\{x_i: i < \omega\}) = \bigcup\{T(x_i): i < \omega\}$. Note that continuity indeed implies monotonicity.

We have the following theorem.

THEOREM 2.4. *For a continuous function T,*

$$T\uparrow\omega = lfp(T). \qquad \square$$

3. *Syntax and Informal Semantics for Logic in Clausal Form*

A *sentence* is a possibly infinite set of clauses. A *clause* is a pair of sets of atomic formulas written as

$$A_1, \ldots, A_m \leftarrow B_1, \ldots, B_n, \qquad m \geq 0, \quad n \geq 0.$$

The set $\{A_1, \ldots, A_m\}$ is the *conclusion* of the clause; $\{B_1, \ldots, B_n\}$ is the *premise* of the clause. An *atomic formula* (or *atom*, for short) is $P(t_1, \ldots, t_k)$, where P is a k-place predicate symbol and $t_1, \ldots, t_k$ are terms. A *term* is a variable or $f(t_1, \ldots, t_j)$ where f is a j-place *functor* (also called *function symbol*), $t_1, \ldots, t_j$ are terms, and $j \geq 0$. A 0-place functor is called a *constant*. We write $a \equiv b$ to denote that a and b are the same sequence of symbols.

Substitution is an operation, say θ, which replaces throughout an expression e all occurrences of a variable by a term. The result is denoted by $e\theta$ and is called an *instance* of e; e is said to be *more general* than $e\theta$ (even when $e \equiv e\theta$). If there exists for given expressions $e_1, \ldots, e_n$ a substitution θ such that $e \equiv e_1\theta \equiv \cdots \equiv e_n\theta$, then θ is said to be a *unifier* of $e_1, \ldots, e_n$.

According to the informal semantics of logic in clausal form, a sentence is to be understood as the conjunction of its clauses. A clause

$$A_1, \ldots, A_m \leftarrow B_1, \ldots, B_n$$

is to be understood as

$$\text{for all} \quad x_1, \ldots, x_k, \qquad A_1 \text{ or } \cdots \text{ or } A_m \quad \text{if} \quad B_1 \text{ and } \cdots \text{ and } B_n,$$

where $x_1, \ldots, x_k$ are the variables in the clause and $m > 0$, $n \geq 0$. A *definite clause* is one where $m = 1$. A sentence containing definite clauses only is called a *definite sentence*. A *negative clause* is one where $m = 0$ and $n > 0$. It is to be understood as: For all $x_1, \ldots, x_k$, it is not the case that B_1 and $\cdots$ and B_n. The "Horn clauses" often used in the literature are clauses which are definite or negative. An *empty clause* is one in which $n = 0$ and $m = 0$. Such a clause is to be understood as a contradiction. It is written as $\square$.

This informal semantics is defined formally in the next section.

4. *Semantics of Logic in Clausal Form*

We define the Herbrand base U of a sentence S to be the set of variable-free atoms containing no predicate symbols or functors other than those occurring in S. Any subset of U is an *interpretation* (for S).

Definition 4.1. Let I be an interpretation.

(1) A sentence is *true in I* iff each of its clauses is true in I.
(2) A clause is *true in I* iff each of its variable-free instances is true in I.

(3) A variable-free clause

$$A_1, \ldots, A_m \leftarrow B_1, \ldots, B_n$$

is *true in* I iff at least one of $A_1, \ldots, A_m$ is true in I or at least one of $B_1, \ldots, B_n$ is not true in I.

(4) A variable-free atom F is true in I iff $F \in I$.

Note that the set of variable-free instances of a term, atom, or clause is understood to result from substitution by terms containing only functors from a given sentence, in this case the sentence for which I is an interpretation.

An interpretation I such that a sentence S is true in I is called a *model* of S. In more general treatments of logic this is called a "Herbrand model." As we consider only such models, the qualification will be omitted. We denote the set of models of S by $M(S)$. If S has no model, then S is said to be *inconsistent.* When, for sentences S_1 and S_2, $M(S_1) \subseteq M(S_2)$, we say that S_2 is a *semantic implication* of S_1, and we write $S_1 \models S_2$. An example of this relationship between sentences is when S_2 is a set of instances of clauses in S_1. Hence

PROPOSITION 4.2. *If a set of instances of clauses in a sentence S is inconsistent, then S is inconsistent.*

$S_1 \models S_2$ implies that $\bigcap M(S_1) \supseteq \bigcap M(S_2)$. Another interesting special case occurs when $S_2 = \{A\}$, where A is a variable-free atom. Now $\bigcap M(\{A\}) = \{A\}$, so that $\bigcap M(S_1) \supseteq \{A\}$. Apparently we have

PROPOSITION 4.3. $\bigcap M(S)$ *is the set of all variable-free atoms A such that* $S \models A$.

With a definite sentence P we associate a function T_P from interpretations to interpretations. Let I be an interpretation. We define T_P with

$A \in T_P(I)$ iff there exists in P a clause $B_0 \leftarrow B_1, \ldots, B_n$ $(n \geq 0)$ such that $A \equiv B_0\theta$ and $\{B_1\theta, \ldots, B_n\theta\} \subseteq I$ for some substitution θ.

We apply the basic results on fixpoints by making the set of the lattice equal to the powerset of the Herbrand base and by making the partial order of the lattice equal to inclusion among subsets of the Herbrand base. Note that T_P is monotone with respect to this order.

THEOREM 4.4. *For any definite sentence P and interpretation I,* $I \supseteq T(I)$ *iff I is a model of P where T is the transformation associated with P.*

PROOF. See [19]. □

COROLLARY 4.5. *For a definite sentence P we have*

$$\bigcap M(P) \in M(P)$$

(the "model-intersection property" for definite sentences).

PROOF. From Section 2 we recall that for monotone T,

$$\bigcap \{x : x \supseteq Tx\} \in \{x : x \supseteq Tx\}.$$

Theorem 4.4 translates this directly to

$$\bigcap M(P) \in M(P).$$ □

In [19] this corollary is proved directly, without recourse to T.

THEOREM 4.6. *Let T be the transformation associated with a definite sentence P. Then T is continuous. In particular,*

$$lfp(T) = T\uparrow\omega.$$

PROOF. See [19]. □

Apparently, for this type of domain and this T, the N in $T\uparrow N = \mathrm{lfp}(T)$ (Theorem 2.2) may always be taken equal to ω. It may, however, happen that

$$\mathrm{gfp}(T) \neq T\downarrow\omega.$$

The following example of such a T is due in part to K. Clark and in part to H. Andreka and I. Nemeti:

$$\begin{aligned} S = \{&(P(a) \leftarrow P(x),\ Q(x)),\ P(s(x)) \leftarrow P(x)\\ &,\ Q(b),\ Q(s(x)) \leftarrow Q(x)\\ &\}. \end{aligned}$$

Let U be the Herbrand base for S generated by the predicate symbols P and Q, the functor s, and the constants a and b. For all finite n we have

$$\begin{aligned} T^n(U) = U\setminus\{&Q(a), \ldots, Q(s^{n-1}(a))\\ &,\ P(b), \ldots, P(s^{n-1}(b))\\ &\}. \end{aligned}$$

Hence $T\downarrow\omega = \{P(s^n(a)) : n < \omega\} \cup \{Q(s^n(b)) : n < \omega\}$. Now, $P(a) \notin T(T\downarrow\omega)$; hence $T\downarrow\omega \neq \mathrm{gfp}(T)$ In fact, $T^n(T\downarrow\omega) = T\downarrow\omega\setminus\{P(s^i(a)) : i < n\}$, for finite n. We have $T\downarrow(\omega+\omega) = \{Q(s^n(b)) : n < \omega\} = \mathrm{gfp}(T) = \mathrm{lfp}(T)$.

The least ordinal n such that $T_P\downarrow n = T_P(T_P\downarrow n)$ has been called by Blair a *closure ordinal*; in [1] these ordinals are investigated for various kinds of definite sentence P.

In Section 9 we prove a general theorem showing under what conditions $T\downarrow\omega$ is the greatest fixpoint of T. This theorem implies, in particular, that if P is finite and there are no function symbols in P, then $T_P\downarrow\omega = \mathrm{gfp}(T_P)$.

5. *SLD Refutations and Their Semantics*

A refutation of a sentence is a syntactic object which is intended to demonstrate the sentence's unsatisfiability. A refutation is not to be confused with a refutation *procedure*, which is a symbol-manipulation procedure for finding a refutation.

Numerous refutation procedures have been based on J. A. Robinson's resolution principle, first by Robinson himself [15, 16] and subsequently by many others [3, 13]. For our purpose the SL-resolution procedure is most important, especially a variant [11, 12, 19] intended for use with sentences containing, apart from one negative clause, only definite clauses. Because of this restriction we refer to this resolution refutation procedure as SLD-resolution: *SL*-resolution for *D*efinite clauses. This section is concerned with SLD refutations. The corresponding refutation procedure is discussed in the next section.

Let P be a definite sentence and N a negative clause. An *SLD-derivation* of $P \cup \{N\}$ consists of a finite or infinite sequence $N_1, N_2, \ldots$ of negative clauses, a sequence $d_1, d_2, \ldots$ of variants of clauses in P (the *input clauses* of the derivation), and a sequence $\theta_1, \theta_2, \ldots$ of substitutions. A variant of a clause y is a clause that differs from y at most in the names of its variables. The variants are such that no input clause d_i of a derivation has a variable in common with the negative clause N_i. Each nonempty N_i contains one atom, which is the *selected atom* of N_i. The clause N_{i+1} is said to be *derived from* N_i and d_i with substitution θ_i.

The relationship of being derived is defined as follows. Let

$$N_i \equiv \leftarrow A_1, \ldots, A_k, \ldots, A_m, \qquad m \geq 1,$$

with A_k as selected atom. Let

$$d_i \equiv A \leftarrow B_1, \ldots, B_q, \qquad q \geq 0,$$

be any clause in P such that A and A_k are unifiable, that is, such that $A\theta \equiv A_k\theta$ for some substitution θ. Then N_{i+1} is

$$\leftarrow (A_1, \ldots, A_{k-1}, B_1, \ldots, B_q, A_{k+1}, \ldots, A_m)\theta,$$

and θ_{i+1} is θ. Each atom $A_j\theta$ of N_{i+1} is said to be a *derived atom* of N_{i+1} (derived from A_j in N_i). Each atom $B_j\theta$ in N_{i+1} is said to be an *introduced atom* of N_{i+1} (introduced by d_i).

Apparently, if a derivation contains the empty clause, it must be its last clause. Such a derivation is called an *SLD-refutation.* The *success set* of a definite sentence P is the set of all A in the Herbrand base of P such that $P \cup \{\leftarrow A\}$ has an SLD-refutation. SLD-refutations are said to be *sound* if the success set is contained in the least model of P (or, by Corollary 4.5, is contained in every model of P); the opposite inclusion (Lemma 5.5) is a form of completeness.

Let $[A]$ denote the set of all variable-free instances of an atom A. Let $\theta_1, \ldots, \theta_n$ be the sequence of substitutions of a refutation. Then their composition is called the *answer substitution.* This term has been chosen because in logic programming this substitution is usually interpreted as an answer to a database query or to a computational problem.

THEOREM 5.1. *We assume that an SLD-refutation exists of a sentence $P \cup \{G\}$, where P is a definite sentence and G is a negative clause containing no functors not occurring in P, and that θ is the answer substitution. We assert that for every atom A in $G\theta_1 \cdots \theta_n$, $[A] \subseteq T^n(\varnothing)$.*

PROOF. Let $G_0, \ldots, G_n$ be the successive negative clauses of the refutation. $G_0 = G$. $G_n = \square$. We show by induction on i that for every atom A in $G_{n-i}\theta_{n-i+1} \cdots \theta_n$, $[A] \subseteq T^i(\varnothing)$.

If $i = 1$, then G_{n-i} consists of a single atom matching a clause, say, with conclusion C, without a premise. $G_{n-1}\theta_n = C\theta_n$. $[C] \subseteq T(\varnothing)$; a fortiori, $[G_{n-1}\theta_n] \subseteq T(\varnothing)$. This takes care of the induction basis $i = 1$.

Suppose now, as the induction assumption, that $[A] \subseteq T^i(\varnothing)$ for any atom A in $G_{n-i}\theta_{n-i+1} \cdots \theta_n$. Let X be an atom of G_{n-i-1}. Suppose first that X is not the selected atom. Then $X\theta_{n-i}$ is an atom of G_{n-i}. The induction assumption ensures that $[(X\theta_{n-i})\theta_{n-i+1} \cdots \theta_n] \subseteq T^i(\varnothing)$. The monotonicity of T implies that $T^i(\varnothing) \subseteq T^{i+1}(\varnothing)$. Thus, X being an atom, but not the selected atom, of G_{n-i-1} implies that $[X\theta_{n-i} \cdots \theta_n] \subseteq T^{i+1}(\varnothing)$.

Suppose that X is the selected atom of G_{n-i-1}. Let $A \leftarrow B_1, \ldots, B_m$ be the $(n-i)$th input clause of the refutation. $X\theta_{n-i}$ is an instance of A.

Case $m = 0$. $[A] \subseteq T(\varnothing)$, by the definition of T. Also $[X\theta_{n-i} \cdots \theta_n] \subseteq [X\theta_{n-i}] \subseteq [A]$ and $T(\varnothing) \subseteq T^{i+1}(\varnothing)$. Hence $[X\theta_{n-i} \cdots \theta_n] \subseteq T^{i+1}(\varnothing)$.

Case $m > 0$. $B_1\theta_{n-i}, \ldots, B_m\theta_{n-i}$ are atoms of G_{n-i}. By the induction hypothesis, $[B_j\theta_{n-i} \cdots \theta_n] \subseteq T^i(\varnothing)$ for $j = 1, \ldots, m$. Hence, by the definition of T, $[X\theta_{n-i} \cdots \theta_n] \subseteq T^{i+1}(\varnothing)$. $\square$

COROLLARY 5.2 (SOUNDNESS OF SLD-REFUTATIONS). *Let P be a definite sentence and N a negative clause such that there exists an SLD refutation of $P \cup \{N\}$. Then $P \cup \{N\}$ is inconsistent.*

PROOF. By Theorem 5.1 there exists a substitution θ such that for all atoms A in N, $A\theta \in \mathrm{lfp}(T) = \bigcap M(P)$. This implies that $N\theta$ is not true in $\bigcap M(P)$, so it is not true in any model of P; a fortiori, the same holds for N. Therefore $P \cup \{N\}$ is inconsistent. □

COROLLARY 5.3. *The success set of a definite sentence is contained in its least model.*

Corollaries 5.2 and 5.3 can be proved from the soundness of resolution in general, which has a simpler proof than Theorem 5.1. The value of this theorem lies elsewhere, namely, in the way it justifies the constructive use of SLD-refutations. What we mean by this is illustrated by the following example. Let

$$P = \{\mathrm{app}(\mathrm{nil}, y, y)$$
$$, \mathrm{app}(u \cdot x, y, u \cdot z) \leftarrow \mathrm{app}(x, y, z)$$
$$\}.$$

The functor "$\cdot$" is used as infix operator: for example, $u \cdot x$ stands for $\cdot(u, x)$. An additional notational convention determines that, for example, $\alpha \cdot \beta \cdot \gamma$ stands for $\alpha \cdot (\beta \cdot \gamma)$ and not for $(\alpha \cdot \beta) \cdot \gamma$. $u \cdot x$ stands for a list with u as first element; x stands for the rest of the list. The constant "nil" stands for the empty list. The clauses of P state theorems about the "append" relation among three lists, where the third list is the result of appending the second list to the end of the first.

Let $A = \mathrm{app}(x_1, 3 \cdot y_1, 2 \cdot 3 \cdot 4 \cdot z_1)$. Now $P \cup \{\leftarrow A\}$ has an SLD-refutation with substitutions, say, $\theta_1, \ldots, \theta_n$. By itself, the soundness of resolution only guarantees the existence of unspecified x_1, y_1, and z_1 such that $2 \cdot 3 \cdot 4 \cdot z_1$ is the result of appending y_1 to x_1. But Theorem 5.1 allows us to use resolution logic as a computational formalism: $A\theta_1 \cdots \theta_n$ is $\mathrm{app}(2 \cdot \mathrm{nil}, 3 \cdot 4 \cdot w, 2 \cdot 3 \cdot 4 \cdot w)$, thereby stating that the x_1, y_1, and z_1 that must exist, by the soundness of resolution, are $2 \cdot \mathrm{nil}$, $4 \cdot w$, and w, respectively, where w can be any variable-free term.

To prove the completeness of SLD-refutations, we establish some lemmas first.

LEMMA 5.4. *Let P be a definite sentence, $N \equiv \leftarrow A_1, \ldots, A_k$ a negative clause, and θ a substitution. If there exists a refutation of $P \cup \{N\theta\}$ with $A_i\theta$ as the first selected atom, then there exists a refutation of $P \cup \{N\}$ with A_i as the first selected atom.*

PROOF. We can assume that θ does not act on any of the variables in P. Suppose now that a refutation of $P \cup \{N\theta\}$ exists, where $A_i\theta$ (in $N\theta$) is the first selected atom. Suppose the first input clause of the refutation is $B_0 \leftarrow B_1, \ldots, B_m$. It follows that $A_i\theta\zeta \equiv B_0\zeta$ for some substitution ζ.

By the assumption there exists a refutation of

$$P \cup \{\leftarrow (A_1\theta, \ldots, A_{i-1}\theta, B_1, \ldots, B_m, A_{i+1}\theta, \ldots, A_k\theta)\zeta\}.$$

But $B_j \equiv B_j\theta$ for $j = 0, \ldots, m$, since θ does not act on any of the variables of P. So $A_i\theta\zeta \equiv B_0\theta\zeta$ and by the above there exists a refutation of $P \cup \{N\}$ with A_i as the first selected atom, $\theta\zeta$ as the first substitution, and the same input clause. □

LEMMA 5.5. *The least model of a definite sentence is contained in its success set.*

PROOF. The proof makes use of the fact that the transformation T is continuous. Assume that A is in the least model of a definite sentence P. By Corollary 4.5, the

least model of P is $\cap M(P)$. By Theorem 2.1, $\cap M(P) = \text{lfp}(T)$. By Theorem 4.6, T is continuous and consequently $A \in \text{lfp}(T) \Rightarrow A \in T^k(\varnothing)$ for some finite k. We now prove by induction on k that $A \in T^k(\varnothing)$ implies that an SLD-refutation exists of $P \cup \{\leftarrow A\}$.

If $k = 1$, then $A \in T^k(\varnothing)$ implies that A is a variable-free instance of the conclusion of a clause in P with an empty premise. Hence there is an SLD-refutation of $P \cup \{\leftarrow A\}$ (of length 1).

If $A \in T^{k+1}(\varnothing)$, then by the definition of T there exists a variable-free instance of a clause $B_0 \leftarrow B_1, \ldots, B_m$ in P such that $A \equiv B_0\theta$ and $\{B_1\theta, \ldots, B_m\theta\} \subseteq T^k(\varnothing)$, for some θ. By the induction hypothesis there exists a refutation of $P \cup \{\leftarrow B_i\theta\}$ for $i = 1, \ldots, m$. Because of the absence of variables in $B_1\theta, \ldots, B_m\theta$, there exists a refutation of $P \cup \{\leftarrow B_1\theta, \ldots, B_m\theta\}$. Hence, there exists a refutation $\leftarrow A, \leftarrow(B_1, \ldots, B_m)\theta, \ldots, \square$ of $P \cup \{\leftarrow A\}$. □

THEOREM 5.6 (COMPLETENESS OF SLD REFUTATIONS). *Let P be a definite sentence and N a negative clause such that $P \cup \{N\}$ is inconsistent. For each atom A_k of N there exists an SLD refutation of $P \cup \{N\}$ with A_k as the first selected atom.*

PROOF. Suppose $N \equiv \leftarrow A_1, \ldots, A_n$. If $P \cup \{N\}$ is inconsistent, then N is not true in $\cap M(P)$; then there exists a variable-free instance $N\theta$ of N which is not true in $\cap M(P)$; then $\{A_1\theta, \ldots, A_n\theta\} \subseteq \cap M(P)$. By Lemma 5.5 there exists an SLD-refutation of $P \cup \{\leftarrow A_i\theta\}$ for $i = 1, \ldots, n$. As all $A_i\theta$ are variable-free, there also exists an SLD refutation of $P \cup \{N\theta\}$, whatever the first selected atom. By Lemma 5.4 there exists an SLD-refutation of $P \cup \{N\}$, whatever the first selected atom. □

COROLLARY 5.7. *The least model of a definite sentence P is the success set of P.*

6. SLD *Refutation Procedures*

Let us now consider symbol manipulation procedures that find an SLD-refutation whenever one exists. Such a procedure would be in some sense an "automatic theorem prover." We are more interested in the use of such procedures for automatic computation; the interpreter for the programming language PROLOG [5, 6] can be regarded as an SLD-refutation procedure.

According to the definition of an SLD-derivation the following choices have to be made in each step of constructing a refutation:

(a) choice of selected atom;
(b) choice of input clause, if two or more clauses have a conclusion unifying with the selected atom;
(c) choice of substitution.

In searching for a refutation, derivations are constructed with the goal of encountering an empty clause. The totality of derivations to be constructed by an SLD-refutation procedure we call the *search space.* Any necessity to consider alternatives to the choices (a)–(c) contributes to the size of the search space. In fact, in the search space we consider, the SLD-tree, only alternatives to choice (b) exist. We define the SLD tree and prove that alternatives (a) and (c) need not be considered.

In the first place, it is easy to see (from Lemma 5.4) that if a refutation of $P \cup \{N\}$ exists, then there also exists one for which every substitution is the most general unifier of the selected atom and the conclusion of the input clause. In fact, in most treatments of resolution, for this reason and because (as far as we know) most general unifiers are no harder to compute than other unifiers, only most general unifiers are

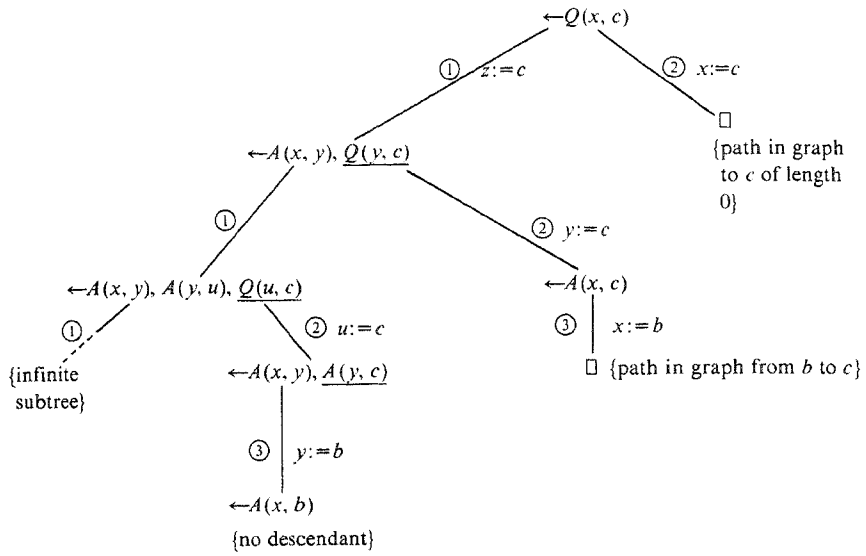

FIG. 2. An SLD-tree for $P \cup \{\leftarrow Q(x, c)\}$.

considered. We showed that according to our less stringent definition, refutations also refute, and we introduce the condition of unifications being most general only to reduce the search space for the refutation procedure. All the results of Section 5 remain valid when this modified definition of refutation is adopted.

We call a search space for the SLD refutation procedure an *SLD-tree* and define it as follows. Let P be a definite sentence and N a negative or empty clause. An SLD-tree for $P \cup \{N\}$ has N as root. All its nodes are negative or empty clauses. A nonempty node has one atom which is the selected atom. A node

$$\leftarrow A_1, \ldots, A_k, \ldots, A_m, \qquad 1 \leq k \leq m, \quad m \geq 1,$$

with selected atom A_k, has a descendant for every clause,

$$A \leftarrow B_1, \ldots, B_q, \qquad q \geq 0,$$

such that A and A_k are unifiable, say, with most general unifier θ. The descendant is

$$\leftarrow (A_1, \ldots, A_{k-1}, B_1, \ldots, B_q, A_{k+1}, \ldots, A_m)\theta.$$

Note that every path in an SLD-tree is an SLD-derivation and that every path to an empty clause is a refutation. Also, for every refutation of $P \cup \{N\}$ there exists an SLD-tree for $P \cup \{N\}$ of which a path is the most general version of this refutation. In general, a given $P \cup \{N\}$ has different SLD-trees depending on which atoms are the selected atoms. Often there are very many SLD-trees, of vastly differing size.

Example 6.1

$$P = \{(\overset{①}{Q}\ (x, z) \leftarrow A(x, y), Q(y, z)), \overset{②}{Q}(x, x), \overset{③}{A}(b, c)\}.$$

In P, x, y, and z are variables, and b and c are constants. A possible meaning is as follows. Objects are nodes of a graph: $A(x, y)$ if there is an arc from x to y; $Q(x, y)$ if there is a path from x to y. Clauses ① and ② define the path relation. Clause ③ gives an arc of the graph. The clause $\leftarrow Q(x, c)$ negates that a path to c exists. One SLD-tree for $P \cup \{\leftarrow Q(x, c)\}$ is shown in Figure 2. The selected atoms are underlined. Another SLD-tree for $P \cup \{\leftarrow Q(x, c)\}$ is shown in Figure 3.

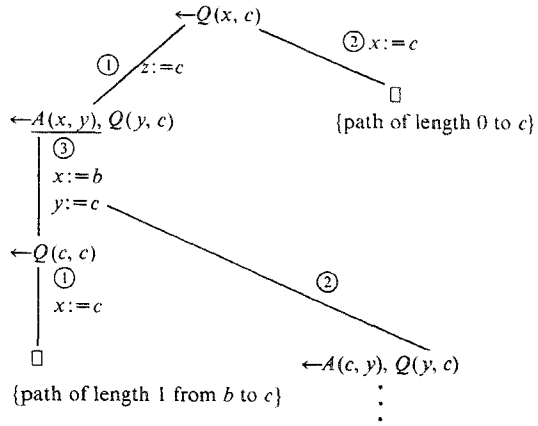

FIG. 3. Another SLD-tree for $P \cup \{\leftarrow Q(x, c)\}$.

We now prove a special form of completeness of SLD-resolution. This result (due to Hill [9]) is a strengthening of the previously proved completeness of SLD refutations (Theorem 5.6). As usual, we need some definitions and lemmas first.

Definition 6.2. A negative clause N is *k-refutable*, $k \geq 1$, if in every SLD-tree with N as root there exists an empty clause with a path length from the root of at most k.

The following lemma corresponds to Lemma 5.4.

LEMMA 6.3. *Let P be a definite clause and $N \equiv \leftarrow A_1, \ldots, A_n$ a negative clause. For any substitution θ, if $N\theta$ is k-refutable, then N is k-refutable.*

PROOF. The proof proceeds by induction on k. The case when $k = 1$ is obvious. To prove the induction step, consider an arbitrary SLD-tree with N as root. Suppose that A_i is the selected atom of the root. Since $N\theta$ is k-refutable, there exists a clause $B_0 \leftarrow B_1, \ldots, B_m$ in P such that $A_i\theta\zeta \equiv B_0\zeta$ for some most general unifier ζ and $\leftarrow(A_1\theta, \ldots, A_{i-1}\theta, B_1, \ldots, B_m, A_{i+1}\theta, \ldots, A_n\theta)\zeta$ is $(k-1)$-refutable.

Similarly as in the proof of Lemma 5.4, we can assume that θ does not act on the variables in P. Thus $B_j \equiv B_j\theta$ for $j = 0, \ldots, m$. For some most general unifier η, $A_i\eta \equiv B_0\eta$ and $\theta\zeta = \eta\eta_1$ for some η_1. By the above, $\leftarrow(A_1, \ldots, A_{i-1}, B_1, \ldots, B_m, A_{i+1}, \ldots, A_n)\eta\eta_1$ is $(k-1)$-refutable; so by the induction hypothesis, $\leftarrow(A_1, \ldots A_{i-1}, B_1, \ldots, B_m, A_{i+1}, \ldots, A_n)\eta$ is $(k-1)$-refutable. But this clause is a direct descendant of the root N of the SLD-tree under consideration. This proves the induction step and concludes the proof. □

LEMMA 6.4. *Assume that $A_1, \ldots, A_n$ are atomic formulas with no variables in common. If each $\leftarrow A_i$ is k_i-refutable, $i = 1, \ldots, n$, then $\leftarrow A_1, \ldots, A_n$ is $k_1 + \cdots + k_n$-refutable.*

PROOF. Straightforward by induction on $k_1 + \cdots + k_n$. □

LEMMA 6.5. *If A is in the least model of P, then for some k, $\leftarrow A$ is k-refutable.*

PROOF. Analogous to the proof of Lemma 5.5 using Lemma 6.4. □

THEOREM 6.6 (STRONG COMPLETENESS OF SLD-RESOLUTION). *If $P \cup \{N\}$ is inconsistent, then every SLD-tree with N as root contains an empty clause.*

PROOF. Analogously to the proof of Lemma 5.5, with the use of Lemmas 6.5 and 6.3 we may show that inconsistency of $P \cup \{N\}$ implies that for some k, N is k-

refutable. By the definition of k-refutability it follows that every SLD-tree with N as root contains an empty clause. □

The strong completeness theorem shows that alternatives in choice (a) (namely, of the selected atom) need not be considered by an SLD-refutation procedure; any one SLD-tree is a complete search space for such a procedure. Whether the procedure will actually find a refutation in an SLD-tree containing the empty clause depends on the tree-search algorithm. A breadth-first algorithm is guaranteed to find an empty clause if one exists, provided that the degree of the SLD-tree is bounded, which may not be the case with an infinite set of clauses. A depth-first algorithm, which is preferable for efficiency of implementation, can fail to find an existing refutation if the SLD-tree is infinite.

The choice of the selected atom can make an enormous difference in the size of the SLD-tree. For example, with a choice that makes the SLD-tree finite, a depth-first algorithm is guaranteed to succeed, whereas with another choice, leading to an infinite SLD-tree, the same algorithm may fail to find a refutation.

An even stronger completeness result for SLD-resolution is proved by Clark [5]. He proves that for every "correct answer substitution" θ there is in every SLD-tree a refutation that has as answer substitution one that has θ as a special case.

7. *The Fixpoint Semantics of Finite Failure*

Consider a definite sentence P with Herbrand base U. The *success set* of P is the subset of U consisting of all variable-free atoms A such that the SLD-trees for P with $\leftarrow A$ as root contain an empty clause. One way of expressing the soundness and the weak completeness of SLD-refutations is to say that the success set equals the least model of P, and therefore equals also the least fixpoint of the associated T, and therefore equals $T\uparrow\omega$ also.

A *finitely failed SLD-tree* is one which is finite and contains no empty clause. The *finite-failure set* of a definite sentence P is the subset in U of all variable-free atoms A such that there exists a finitely failed SLD-tree with $\leftarrow A$ as root. In this section we show that the finite-failure set is equal to the complement in U of $T\downarrow\omega$. Because $T\downarrow\omega \supseteq \text{gfp}(T)$, with equality not necessarily being true, we can only conclude in general that the finite-failure set is included in the complement of $\text{gfp}(T)$. Because of this result we are especially interested in classes of definite sentences for which $T\downarrow\omega = \text{gfp}(T)$ also holds. After this section we give an intuitive and semantic interpretation of $\text{gfp}(T)$.

THEOREM 7.1. *The finite-failure set of a definite sentence P is included in the complement in U of $T\downarrow\omega$.*

PROOF. Assume that for a variable-free atom A, $\leftarrow A$ is the root of a finitely failed SLD-tree of depth $\leq k$. We prove by induction over finite values of k that $A \notin T^k(U)$.

If $k = 1$, then A is not an instance of the conclusion of any clause in P, so $A \notin T(U)$.

Assume now that $k \geq 1$. Suppose also that $A \in T^k(U)$; we show that this leads to a contradiction. There exists a clause $B_0 \leftarrow B_1, \ldots, B_n$ in P such that $A \equiv B_0\theta$ and $\{B_1\theta, \ldots, B_n\theta\} \in T^{k-1}(U)$ for some variable-free substitution θ. For some most general unifier λ, $A\lambda \equiv B_0\lambda$ and $\theta = \lambda\eta$ for some substitution η. Hence $\leftarrow(B_1, \ldots, B_n)\lambda$ is a direct descendant of the root $\leftarrow A$ in the SLD-tree, which is therefore the root of a finitely failed SLD-tree of depth $\leq k - 1$. By Lemma 7.2, $\leftarrow(B_1, \ldots, B_n)\theta$ is also the root of a finitely failed SLD-tree of depth $\leq k - 1$. Now, by Lemma 7.3, for

some $i = 1, \ldots, n$, $\leftarrow B_i\theta$ is the root of a finitely failed SLD-tree of depth $\leq k-1$. By the induction hypothesis, $B_i\theta \notin T^{k-1}(U)$, which contradicts the supposition that $A \notin T^k(U)$. □

We now state the two lemmas used.

LEMMA 7.2. *Let λ be a substitution and N a negative clause. Assume that N is the root of a finitely failed SLD-tree of depth $\leq k$. Then $N\lambda$ is the root of a finitely failed SLD-tree of depth $\leq k$ also.*

LEMMA 7.3. *Let $A_1, \ldots, A_n$ be variable-free atomic formulas such that $\leftarrow A_1, \ldots, A_n$ is the root of a finitely failed SLD-tree tree of depth $\leq k$. Then for some $i = 1, \ldots, n$, $\leftarrow A_i$ is the root of a finitely failed SLD-tree of depth $\leq k$ also.*

The proofs of both lemmas proceed by induction on k and are straightforward.

If we view the construction of a finitely failed SLD-tree as a method of computing complements in U of $T\downarrow\omega$, then Theorem 7.1 states the *correctness* of the method. We prepare the *completeness* proof by introducing some definitions and lemmas.

Definition 7.4. A definite sentence is said to be of *finite degree* if for no negative clause N there exists an SLD-tree with N as root and containing a node of infinite degree.

A negative clause N is called *infinite* (with respect to a definite sentence P) if every SLD-tree for P with N as root is infinite.

For example, if no predicate symbol occurs in infinitely many conclusions of clauses, then P is of finite degree.

LEMMA 7.5. *For no atom A in an infinite negative clause N can $\leftarrow A$ be the root of a finitely failed SLD-tree.*

PROOF. If a finitely failed SLD-tree F with A as root did exist, then a finitely failed SLD-tree tree could be constructed with N as root by initially selecting A and then selecting the same atoms as in F. □

LEMMA 7.6. *Let N be a negative clause which is infinite with respect to a definite sentence of finite degree. In every SLD-tree with N as root, there is a direct descendant of N which is infinite.*

This is a form of König's Lemma.

Definition 7.7. Let M and N be negative clauses. N contains the atom A, and θ is a substitution. We write $N \Rightarrow^{\theta;A} M$ to denote the fact that there exists an SLD-derivation with $G_0, \ldots, G_k$ as sequence of clauses such that

(i) $G_0 \equiv N$ and $G_k \equiv M$.
(ii) $\theta = \eta_1, \ldots, \eta_k$ is the composition of the successive substitutions of the derivation.
(iii) A is the selected atom of N.
(iv) The selected atoms of the clauses $G_1, \ldots, G_{k-1}$ are introduced atoms (in the sense of the definition of Section 5). This implies that M is of the form $\leftarrow(A_1, \ldots, A_{i-1}, N', A_{i+1}, \ldots, A_n)\theta$, for some list N' of atoms, if N is $\leftarrow A_1, \ldots, A_{i-1}, A, A_{i+1}, \ldots, A_n$.

LEMMA 7.8. *Suppose that $\leftarrow A_1, \ldots, A_n$ is infinite with respect to a definite sentence of finite degree and that $\leftarrow A_i$ is not infinite. Then for some substitution θ, we have $\leftarrow A_1, \ldots, A_n \Rightarrow^{\theta;A_i} \leftarrow(A_1, \ldots, A_{i-1}, A_{i+1}, \ldots, A_n)\theta$, $[A_i\theta] \subseteq T\downarrow\omega$, and $\leftarrow(A_1, \ldots, A_{i-1}, A_{i+1}, \ldots, A_n)\theta$ is infinite.*

PROOF. By Lemma 7.5, $\leftarrow A_i$ cannot be the root of a finitely failed SLD-tree. As $\leftarrow A_i$ is not infinite, it must be the root of a finite SLD-tree F containing the empty clause. By Theorem 5.1, for some θ, $[A_i\theta] \subseteq \mathrm{lfp}(T)$; also, $\mathrm{lfp}(T) \subseteq T\downarrow\omega$. By the definition of $\Rightarrow$, the first part of the lemma is proved. We now show that if $\leftarrow(A_1, \ldots, A_{i-1}, A_{i+1}, \ldots, A_n)\theta$ were not infinite, then $\leftarrow A_1, \ldots, A_n$ would not be infinite either.

Consider a tree F', isomorphic to F, obtained by performing the same resolutions on the same selected atoms but with $\leftarrow A_1, \ldots, A_n$ as root rather than $\leftarrow A_i$. F' is itself not necessarily an SLD-tree, but only the initial part of one. We complete F' to an SLD-tree by constructing SLD-trees with the terminal nodes of F' as roots. In a terminal node of F', corresponding to a nonempty terminal node of F, we select the same atom as in F. As a result, the node in F' has no descendant. In a terminal node $\leftarrow(A_1, \ldots, A_{i-1}, A_{i+1}, \ldots, A_n)\theta$ of F' corresponding to an empty (terminal) node of F, we select an arbitrary atom. There is only a finite number of this kind of nodes. If none of them were infinite, then a finite SLD-tree, with $\leftarrow A_1, \ldots, A_n$ as root, could be constructed. □

LEMMA 7.9. *Assume a definite sentence of finite degree. For each $k \geq 1$, if $\leftarrow A_i$ is infinite, then for every infinite clause $\leftarrow N$ containing A_i there exists a substitution θ and an infinite clause $\leftarrow M$ such that*

$$\leftarrow N \Rightarrow^{\theta; A_i} \leftarrow M \qquad \text{and} \qquad [A_i\theta] \subseteq T^k(U).$$

PROOF. Let $\leftarrow A_i$ and $N \equiv \leftarrow A_1, \ldots, A_i, \ldots, A_n$ both be infinite clauses. By Lemma 7.6 there exists an SLD-tree with N as root having as direct descendant the infinite clause,

$$\leftarrow(A_1, \ldots, A_{i-1}, B_1, \ldots, B_m, A_{i+1}, \ldots, A_n)\theta. \tag{1}$$

We prove the lemma by induction on k. Clearly $[A_i\theta] \subseteq T(U)$, which provides the induction basis. For the induction step assume that the lemma holds for $k - 1$.

Case 1. $\leftarrow B_1\theta$ is infinite. By the induction hypothesis there exists an infinite clause N_1 and a substitution θ_1, such that $[B_1\theta\theta_1] \subseteq T^{k-1}(U)$,

$$\leftarrow(A_1, \ldots, A_{i-1}, B_1, \ldots, B_m, A_{i+1}, \ldots, A_n)\theta \Rightarrow^{\theta_1; B_1\theta} N_1, \tag{2}$$

where, by the definition of $\Rightarrow$, N_1 is of the form

$$\leftarrow A_1\theta\theta_1, \ldots, A_{i-1}\theta\theta_1, M_1, B_2\theta\theta_1, \ldots, B_m\theta\theta_1, A_{i+1}\theta\theta_1, \ldots, A_n\theta\theta_1. \tag{3}$$

Case 2. $\leftarrow B_1\theta$ is not infinite. By Lemma 7.8 there exists a substitution θ_1 such that $[B_1\theta\theta_1] \subseteq T\downarrow\omega \subseteq T^{k-1}(U)$,

$$\leftarrow(A_1, \ldots, A_{i-1}, B_1, \ldots, B_m, A_{i+1}, \ldots, A_n)\theta \Rightarrow^{\theta_1; B_1\theta} N_1,$$

where N_1 is

$$\leftarrow(A_1, \ldots, A_{i-1}, B_2, \ldots, B_m, A_{i+1}, \ldots, A_n)\theta\theta_1.$$

Thus in both cases there exists an infinite clause N_1 and a substitution θ such that $[B_1\theta\theta_1] \subseteq T^{k-1}(U)$, (2) holds, and N_1 is of the form (3). We proceed now in the same way with $B_j\theta\theta_1 \cdots \theta_{j-1}$, for $j = 2, \ldots, m$, successively. Each time we obtain a substitution θ_j and an infinite clause N_j such that

$$N_{j-1} \Rightarrow^{\theta_j; B_j\theta\theta_1 \cdots \theta_{j-1}} N_j \qquad \text{and} \qquad [B_j\theta\theta_1 \cdots \theta_{j-1}] \subseteq T^{k-1}(U).$$

By the definition of $\Rightarrow$ we have

$$N \Rightarrow^{\theta\theta_1 \cdots \theta_m; A_i} N_m.$$

Note that $[B_j\theta\theta_1 \cdots \theta_{j-1}] \supseteq [B_j\theta\theta_1 \cdots \theta_m]$ for $j = 1, \ldots, m$, so we can conclude that $[B_j\theta\theta_1 \cdots \theta_m] \subseteq T^{k-1}(U)$ for $j = i, \ldots, m$. Thus, by the definition of T, $[A_i\theta\theta_1 \cdots \theta_m] \subseteq T^k(U)$, which completes the induction step. □

COROLLARY 7.10. *If A is a variable-free atom such that $\leftarrow A$ is infinite with respect to a definite sentence of finite degree, then $A \in T{\downarrow}\omega$.*

By Corollary 5.7 we have that the success set of a definite sentence P is $T{\uparrow}\omega$. We are now in a position to state its dual.

THEOREM 7.11. *Let P be a definite sentence of finite degree. The finite-failure set of P is the complement in U of $T \downarrow \omega$.*

PROOF. Theorem 7.1 states that the finite-failure set is not too large. To show that it is not too small, let A be in the complement in U of $T \downarrow \omega$. By Corollary 7.10, $\leftarrow A$ is not infinite. If $\leftarrow A$ were the root of an SLD-tree containing the empty clause, then by Corollary 5.7, $A \in \text{lfp}(T) \subseteq T \downarrow \omega$. Hence A is in the finite-failure set of P. □

8. *Negation Inferred from Finite Failure*

We have discussed an extremely specialized kind of inference system: the applicability of SLD-resolution is restricted to sentences in clausal form, which, moreover, have to consist of definite clauses and exactly one negative clause. Clausal form is, in principle, no restriction in expressiveness. But the restriction to definite clauses does limit what one can say.

For example,

$$E = \{\text{Element(Fire)}, \text{Element(Air)}, \text{Element(Water)}, \text{Element(Earth)}, \text{Stuff(Mud)}\}$$

says what some of the elements are. But it does not express the fact that these are the *only* elements.

If we want to establish that Air is an Element, then we can use SLD-resolution to refute $E \cup \{\leftarrow\text{Element(Air)}\}$. But how can we use SLD resolution to show that Mud is not an Element? We cannot expect to be able to do so by constructing a refutation with input clauses from E alone, because $\leftarrow$Element(Mud) is not a semantic implication of E; Element(Mud) is not false in all models of E. In fact, in general, for any definite sentence P we can say that $P \models \leftarrow A$ holds for no A in the Herbrand base U of P; as U itself is a model of P, none of its elements is false in all models of P. With respect to a definite clause, some things are necessarily true, but nothing is necessarily false.

In the traditional syntax of predicate logic, E can be expressed as

$$\text{Stuff(Mud)} \wedge \forall x.\text{Element}(x) \leftarrow x = \text{Air} \vee x = \text{Fire} \vee x = \text{Water} \vee x = \text{Earth} \cdots \qquad (4)$$

If we want to add the information that the things said to be Elements are the only such, we can simply change the implication to an equivalence. In clausal form this information can, of course, also be expressed. But the resulting clausal sentence is not definite, hence SLD-resolution does not apply.

Yet, is it a coincidence that a finitely failed SLD-tree exists for E with $\leftarrow$Element(Mud) as root? It is not: we shall show that this implies that $\neg$Element(Mud) is a semantic implication of the if-and-only-if version of (4). This use of finite failure is due to Clark [4], who justified it by showing that the finitely failed SLD-tree is

isomorphic to a first-order deduction using the if-and-only-if version of the clauses together with axiom schemata for equality. In this section we give a justification on the basis of Theorem 7.11.

We associate with each definite clausal sentence first-order formulas in the traditional syntax of predicate logic. One is called the IF-definition of the clause. It is equivalent to the clausal sentence. The other is called the IFF-definition. It differs from the IF-definition only in that all implications are replaced by the equivalence connective. We then show that whenever a finitely failed SLD tree exists for $P \cup \{\leftarrow A\}$, with A a variable-free atomic formula, the negation of A is semantically implied by the IFF-definition associated with P.

The IF-definition associated with a *finite* definite sentence P is obtained by applying each of the following rules, in the order given. We assume that P has no occurrence of the two-place infix predicate symbol "=."

Rule 1. Change each clause

$$R(s_1, \ldots, s_n) \leftarrow A_1, \ldots, A_m, \qquad n \geq 1, \quad m \geq 0,$$

of P to the universally quantified formula

$$\forall x_1, \ldots, x_n.\ R(x_1, \ldots, x_n) \leftarrow \exists y_1, \ldots, y_k.\ x_1 = s_1 \wedge \cdots \wedge x_n = s_n \wedge A_1 \wedge \cdots \wedge A_m,$$

where $x_1, \ldots, x_n$ are different from the clause's variables and $y_1, \ldots, y_k$ are the variables occurring in $s_1, \ldots, s_n, A_1, \ldots, A_m$.

Note that the clause is true in I (according to Definition 4.1) iff the resulting formula is true in I (according to the usual definition of truth in Definition 8.1).

For the sake of simplicity we prefer not to add a rule for the case $n = 0$. The requirement that predicates have to have at least one argument is no loss of expressiveness and hardly an inconvenience.

Rule 2. Change each set $\{\forall x_1, \ldots, x_m.\ (P \leftarrow Q_1), \ldots, \forall x_1, \ldots, x_m.\ (P \leftarrow Q_n)\}$ of implications, obtained by the above rule and having the same predicate symbol in the conclusion, to $\forall x_1, \ldots, x_m.\ P \leftarrow (Q_1 \vee \cdots \vee Q_n)$. As a result of this rule, the original clausal sentence has been transformed to a set of universally quantified implications, no two of which have the same predicate symbol in the conclusion, having a disjunction of existentially quantified conjunctions as premise.

We now apply the following.

Rule 3. Change this set to the conjunction of its elements. This conjunction is the IF-definition associated with the original definite clausal sentence.

The last rule is the following.

Rule 4. For any (say, n-place) predicate symbol Q which occurs in a premise but in no conclusion of a clause, add the implication $\forall x_1, \ldots, x_n.\ Q(x_1, \ldots, x_n) \leftarrow Q(x_1, \ldots, x_n)$.

Note that dropping the original clause that contains such a Q does not affect the least model. Since we are interested in other models as well, we have to consider the case when such clauses are present. The added clause affects only the transformation T_P whose fixpoints we shall consider. It has another effect as well: previously finite SLD trees may become infinite.

Example

$$(\forall x\,[\text{Element}(x) \leftarrow x = \text{Air} \vee x = \text{Fire} \vee x = \text{Water} \vee x = \text{Earth}])$$
$$\wedge\,(\forall x\,[\text{Stuff}(x) \leftarrow x = \text{Mud}])$$

is the IF-definition associated with E. □

Example

$$\forall u, v, w.[\text{App}(u, v, w) \leftarrow (\exists y.u = \text{nil} \wedge v = y \wedge w = y)$$
$$\vee\,(\exists u_1, x, y, z.(u = u_1 \cdot x \wedge v = y \wedge w = u_1 \cdot z \wedge \text{App}(x, y, z)))]$$

is the IF-definition associated with

$$\{\text{App}(\text{nil}, y, y)$$
$$, \text{App}(u \cdot x, y, u \cdot z) \leftarrow \text{App}(x, y, z)$$
$$\}.$$

□

As was mentioned before, the IFF-definition associated with a definite clause is the IF-definition with each implication changed to the equivalence connective.

We now define when a certain class of nonclausal first-order formulas is true in an interpretation I. As before, interpretations are subsets of a certain Herbrand base.

Definition 8.1

(1) A universally quantified implication is true in I iff for each variable-free instantiation of the implication which makes the premise true in I, the conclusion is also true in I.
(2) An existentially quantified conjunction with no free variables is true in I iff at least one variable-free instance of the conjunction is true in I.
(3) A variable-free conjunction is true in I iff each conjunct is true in I.
(4) A variable-free disjunction is true in I iff at least one disjunct is true in I.
(5) A variable-free atomic formula A is true in I iff $A \in I$ or, independently of I, A is $t_1 = t_2$ with t_1 and t_2 the same variable-free term.

LEMMA 8.2. *Let R be an n-place predicate symbol in a conclusion of an implication of an IF-definition associated with a finite definite clausal sentence P. $R(t_1, \ldots, t_n) \in T_P(I)$ iff the substitution $(t_1/x_1, \ldots, t_n/x_n) = \lambda$ makes the implication's premise true in I, where $x_1, \ldots, x_n$ are the free variables of the universally quantified implication.*

PROOF. $R(t_1, \ldots, t_n) \in T_P(I)$ iff there exists in P a clause $R(s_1, \ldots, s_n) \leftarrow B_1, \ldots, B_m$ such that $R(t_1, \ldots, t_n) \equiv R(s_1, \ldots, s_n)\theta$ and $\{B_1\theta, \ldots, B_m\theta\} \subseteq I$, for some θ, iff the substitution λ makes the premise of the implication true in I. □

LEMMA 8.3. *For all interpretations I, the IFF-definition associated with a finite definite sentence P is true in I iff $I = T_P(I)$.*

PROOF

If. We should first verify that P and its IF-definition have the same set of models. This is immediate in the case where every predicate symbol occurs in a conclusion. In the other case it is obvious that an IF-definition containing an implication of the form $\forall x_1, \ldots, x_n.\ Q(x_1, \ldots, x_n) \leftarrow Q(x_1, \ldots, x_n)$ which has no counterpart in P (see Rule 4) also has the the same set of models as P. Hence Theorem 4.4 can be used to conclude that for all interpretations I, $I \supseteq T_P(I)$ iff the IF-definition is true in I.

Assume now that $I \subseteq T_P(I)$ and $R(t_1, \ldots, t_n)$ is true in I, with R the predicate symbol in a left-hand side and $t_1, \ldots, t_n$ variable-free. By the assumption $I \subseteq T_P(I)$,

$R(t_1, \ldots, t_n)$ is true in $T_P(I)$. By Lemma 8.2, the corresponding instance of the right-hand side is true in I. We conclude that $I = T(I)$ implies that the IFF-definition is true in I.

Only if. We assume the IFF-definition is true in I. We have to show $I \subseteq T_P(I)$. $R(t_1, \ldots, t_n) \in I \Rightarrow$ because of Definition 8.1 there exists an equivalence having $R(x_1, \ldots, x_n)$ in its conclusion and the substitution $(t_1/x_1, \ldots, t_n/x_n)$ makes its premise true in $I \Rightarrow$ (Lemma 8.2) $R(t_1, \ldots, t_n) \in T_P(I)$. □

THEOREM 8.4 [4, 5]. *Let P be a finite definite clausal sentence. If A is in the finite failure set of P, then $\neg A$ is semantically implied by the IFF-definition associated with P.*

PROOF. Suppose A is the root of a finitely failed SLD-tree. By Theorem 7.11, $A \notin T_P \downarrow \omega$. Also, we have $T_P \downarrow \omega \supseteq \text{gfp}(T_P)$. Hence $A \notin \text{gfp}(T_P)$; hence, by Theorem 2.1, $A \notin I$ for any I such that $I = T(I)$; hence, by Lemma 8.3, A is false in all models of the IFF-definition. □

In some cases we can prove the converse theorem.

THEOREM 8.5. *Let P be a finite definite sentence such that $T_P \downarrow \omega = gfp(T_P)$, and let A be a variable-free atom. If $\neg A$ is semantically implied by the IFF-definition associated with P, then A is in the finite-failure set of P.*

PROOF. The argument of the proof of Theorem 8.4 can be reversed provided that $T_P \downarrow \omega = \text{gfp}(T_P)$. □

9. *Applications*

The fixpoint semantics of finite failure has at least two interesting applications. The first is to the semantics of the "closed-world assumption" for databases, various aspects of which are discussed in [4, 14, 18]. A database can be regarded as a sentence stating that certain relations hold. A conventional relational database then becomes a definite clause of a very special form: no conditional clauses, no variables, no functors. Other types of databases are being investigated for which some of these restrictions have been lifted. As a result, the definite sentence is a useful model (not in the technical sense) of databases in general.

Definite sentences only explicitly say what is true. There are two distinct ways of stating that certain atomic formulas are false. One is the "closed-world assumption" discussed by Reiter [14], which assumes that every variable-free atomic formula not provable from a definite sentence S is false. The other way, let us call it the IFF assumption (see also [12, Ch. 11]), is to regard S as having as meaning the associated IFF definition. Now, what is false under the closed-world assumption is the complement of the least fixpoint of T_S; what is false under the IFF assumption is the complement of the greatest fixpoint. It is thus clear that the two assumptions are equivalent only for rather special definite sentences (which include conventional relational databases). Moreover, the use of finite failure to conclude negations approximates more closely the IFF assumption; the approximation becomes an equality if $\text{gfp}(T) = T \downarrow \omega$.

The second application is to the semantic characterization of the behavior of nondeterministic programs. For a discussion of this we need a brief description of flowgraph programs, their computations, and their representation in logic. For examples and comparisons with other models of computation the reader is referred to [17].

A flowgraph is a possibly infinite directed graph in which the arcs are labeled by commands. There is one node called the start node S; it has no incoming arc. There is one node called the halt node H; it has no outgoing arc. Each command is a binary relation over states of a machine. The *transition relation* holds between two (node, state)-pairs (N_{i-1}, σ_{i-1}) and (N_i, σ_i) iff there is an arc from N_{i-1} to N_i and $(\sigma_{i-1}, \sigma_i) \in C_i$, the command labeling that arc. A *computation* is a possibly infinite sequence of (node, state) pairs such that every element (N_j, σ_j) has a successor (N_{j+1}, σ_{j+1}) in the sequence if (N_j, σ_j) is in the transition relation with some (node, state)-pair and (N_{j+1}, σ_{j+1}) must be one such pair. Also, the node of the first pair in a computation must be the start node S. It follows that whenever the halt node H occurs in a computation, it must be finite, and that H must occur in the last pair. Such a computation is called *successful.* A finite computation which is not successful is called *blocked.* Flowgraphs also admit infinite computations.

For a given (node, state)-pair (N, σ) there may be several pairs (N', σ') such that (N, σ) and (N', σ') are in the successor relation. It is this feature that makes the epithet "nondeterministic" applicable to flowgraphs. A flowgraph is said to be of finite *nondeterminacy* if the number of such successors is finite. The assumption of finite nondeterminacy has been extensively studied in the literature. Perhaps the best known reference is [7]. Our subsequent considerations are closely related to [8], where various execution methods for the case of nondeterministic programs are studied and the distinction is made among infinite, successful, and blocked computations.

With a flowgraph and a machine we associate a definite clausal sentence P. For each distinct node or command there is a distinct two-place predicate symbol. For each arc from V to W labeled with C there is a clause in P,

$$V(x, z) \leftarrow C(x, y), W(y, z).$$

In addition to these clauses there is the clause

$$H(x, x).$$

We also add to P all clauses $C(a, b)$ such that "C" is the name of a command and $(a, b) \in C$.

With a computation we associate a sequence of negative clauses as follows. For $i = 1, 2, \ldots$, if (N_i, s_i), (N_{i+1}, s_{i+1}) are the ith and $(i + 1)$st pairs of the computation, then $\leftarrow N_i(s_i, z)$, $(\leftarrow C(s_i, y), N_{i+1}(y, z))$, and $\leftarrow N_{i+1}(s_{i+1}, z)$ are the $(2i - 1)$st, $2i$th, and $(2i + 1)$st negative clauses.

LEMMA 9.1. *The sequence of clauses associated with a computation is a derivation in which the leftmost atom is always selected. The computation is successful iff the associated derivation can be made into a refutation by appending the empty clause to it.*

This correspondence between computations and derivations is the basis for our characterizations of certain behaviors of flowgraphs. Note that computations form a tree, just as derivations form an SLD-tree. An *interpreter* for flowgraphs is a procedure to be used with the purpose of constructing a successful computation. In [8] such procedures are studied in a general framework of computation trees and are called *execution methods.* An interpreter is analogous to an SLD-refutation procedure for the clauses associated with the flowgraph. Conventionally, only interpreters are considered that perform a depth-first search of the tree of computations. These correspond to the DT execution method of [8].

In the case of possibility of nondeterminacy, such an interpreter will, after having constructed a blocked computation, backtrack to the most recent point where a

choice in successor remained untried. It will then continue, using that previously untried choice. Of course, other interpreters are possible, as shown, for example, in [8].

For the behavior of such an interpreter when starting in a state a, we distinguish the following mutually exclusive contingencies.

(A) For some choice of successors, a successful computation is found, with final state b.
(B) For no choice of successors, the interpreter terminates successfully; for some choice of successors, the interpreter does not terminate.
(C) For any choice of successors, the interpreter terminates and no successful computation is found.

The distinction among these contingencies has been made in [8, 10] and other papers in the context of nondeterministic computation. Let P be the clausal sentence associated with the flowgraph. In [17] one may find an equivalent of the following.

THEOREM 9.2. *We have contingency (A) iff* $P \vDash S(a, b)$.

PROOF

Only if. Successful computation exists $\Rightarrow$ by Lemma 9.1, the corresponding derivation $\leftarrow S(a, x), \ldots, \leftarrow H(b, x)$ exists $\Rightarrow$ refutation $\leftarrow S(a, x), \ldots, \leftarrow H(b, x), \square$ exists, in which b is substituted for $x \Rightarrow S(a, b) \in \mathrm{lfp}(T_P)$, by Theorem 5.1 $\Rightarrow$ $P \vDash S(a, b)$.

If. $P \vDash S(a, b) \Rightarrow P \cup \{\leftarrow S(a, b)\}$ inconsistent $\Rightarrow$ (Lemma 5.5) the refutation $\leftarrow S(a, b), \ldots, \square$ exists $\Rightarrow$ because $H(x, x)$ is the only clause without a premise, refutation $\leftarrow S(a, b), \ldots, \leftarrow H(t, b), \square$ is a refutation $\Rightarrow$ (Lemma 9.1) $(S, a), \ldots, (H, b)$ is a successful computation. $\square$

To obtain a characterization of contingencies (B) and (C) we use the results of Section 8. Therefore we restrict our attention to flowgraphs for which the associated definite clausal sentence P is finite. This sentence is finite if both the flowgraph and the state set are finite.

First we prove the following result, which is of independent interest.

THEOREM 9.3. *If P represents a flowgraph of finite nondeterminacy, then* $T_P{\downarrow}\omega = gfp(T_P)$.

PROOF. For any P we have $T_P{\downarrow}\omega \supseteq \mathrm{gfp}(T_P)$. Hence it suffices to show that $T_P{\downarrow}\omega \subseteq T(T_P{\downarrow}\omega)$.

$Q(a, c) \in T_P{\downarrow}\omega \Rightarrow$ for all n, $Q(a, c) \in T^n(U)$
$\Rightarrow$ there exists a clause $Q(x, z) \leftarrow C(x, y), R(y, z)$ in P such that for infinitely many n, $C(a, \sigma_n)$ and $R(\sigma_n, c)$ both in $T^{n-1}(U)$; this is because, by finite nondeterminacy, only finitely many clauses resolve with $\leftarrow Q(a, c)$
$\Rightarrow$ there exists a σ such that $C(a, \sigma)$ and $R(\sigma, c)$ both in $T^{n-1}(U)$ for infinitely many n; this is because, by finite nondeterminacy, only finitely many σ_n exist such that $C(a, \sigma_n)$
$\Rightarrow$ $C(a, \sigma)$ and $R(\sigma, c)$ both in $T_P{\downarrow}\omega$
$\Rightarrow$ $Q(a, c) \in T(T_P{\downarrow}\omega)$, by the definition of T. $\square$

We can now provide a characterization of contingency (C).

THEOREM 9.4. *Assume the state set is finite. For a finite flowgraph we have contingency (C) iff* $P' \models \neg S(a, \sigma)$ *for all states* σ, *where* P' *is the IFF-definition associated with* P.

PROOF

Only if. Contingency (C) $\Rightarrow$ (Lemma 9.1) $\leftarrow S(a, x)$ is the root of a finitely failed SLD-tree $\Rightarrow$ (Lemma 7.2) for all states σ, $\leftarrow S(a, \sigma)$ is the root of a finitely failed SLD-tree $\Rightarrow$ (Theorem 8.4) for all states σ, $P' \models \neg S(a, \sigma)$.

If. For all states σ, $P' \models \neg S(a, \sigma) \Rightarrow$ (Theorems 8.5 and 9.3) for all states σ, there exists a finitely failed SLD-tree with $\leftarrow S(a, \sigma)$ as root $\Rightarrow$ (by the form of P) for all states σ there exists a finitely failed SLD tree with $\leftarrow S(a, \sigma)$ as root, where the leftmost atom is always the selected atom $\Rightarrow$ (Lemma 9.1) all computations starting in state a are blocked; that is, we have contingency (C). □

COROLLARY 9.5. *Assume the state set is finite. For a finite flowgraph we have contingency (B) iff*

$$P \models S(a, \sigma) \qquad \textit{for no state } \sigma$$

and

$$P' \models S(a, \sigma) \qquad \textit{for at least one state } \sigma,$$

where P is the definite sentence representing the flowgraph and P′ is the IFF-definition associated with P.

Alternative characterizations of the contingencies (A)–(C) can be given by referring to the least and greatest fixpoints of T since, as in fact used in the above proofs,

$$\begin{array}{lll} P \models S(a, \sigma) & \text{iff} & S(a, \sigma) \in \text{lfp}(T_P), \\ P' \models S(a, \sigma) & \text{iff} & S(a, \sigma) \in \text{gfp}(T_P). \end{array}$$

We conclude this section by formulating a general theorem which concerns sentences P for which $T_P \downarrow \omega = \text{gfp}(T_P)$.

THEOREM 9.6. *If each SLD-tree for* $P \cup \{\leftarrow A\}$, *where A is any variable-free atom, is of finite degree and there are finitely many variable-free terms in the Herbrand universe, then* $T_P \downarrow \omega = gfp(T_P)$.

PROOF. The proof is analogous to that of Theorem 9.3. □

ACKNOWLEDGMENTS. The initial impetus of this work resulted from discussions with Robert Kowalski. Further discussions with Hajnal Andreka, Keith Clark, and Istvan Nemeti have provided valuable insights. A detailed referee's report by David Harel helped to correct errors and improve the presentation. The National Science and Engineering Research Council has contributed supporting facilities.

REFERENCES

1. BLAIR, H.A. The recursion-theoretic complexity and fixpoint semantics of definite sentences. Ph.D. Dissertation, Dep. of Computer and Information Science, Syracuse Univ., Syracuse, N.Y., 1980.
2. BLIKLE, A. Proving programs by sets of computations. In *Mathematical Foundations of Computer Science III*, Lecture Notes in Computer Science 28, A. Blikle, Ed., Springer-Verlag, Berlin, Heidelberg, New York, 1975, pp. 333–358.
3. CHANG, C.L., AND LEE, R.C.T. *Symbolic Logic and Mechanical Theorem-Proving.* Academic Press, New York, 1973.

4. CLARK, K.L. Negation as failure. In *Logic and Data Bases*, H. Gallaire and J. Minker, Eds., Plenum Press, New York, 1978, pp. 293–324.
5. CLARK, K.L. Predicate logic as a computational formalism. Res. Rep., Dep. of Computing, Imperial College, London, 1979.
6. COLMERAUER, A. Metamorphosis grammars. In *Natural Language Communication with Computers*, L. Bolc, Ed., Lecture Notes in Computer Science 63, Springer-Verlag, Berlin, Heidelberg, New York, pp. 133–189.
7. DIJKSTRA, E.W. *A Discipline of Programming*. Prentice Hall, Englewood Cliffs, N.J., 1976.
8. HAREL, D. On the total correctness of nondeterministic programs. *Theor. Comput. Sci. 13* (1981), 175–192.
9. HILL, R. LUSH-resolution and its completeness. Memo 78, Dep. of Computational Logic, Univ. of Edinburgh, Edinburgh, Scotland, 1974.
10. HITCHCOCK, P., AND PARK, D. Induction rules and termination proofs. In *Automata, Languages, and Programming*, M. Nivat, Ed., North-Holland, Amsterdam, 1973.
11. KOWALSKI, R.A. Predicate logic as a programming language. In *Information Processing 74*, J. Rosenfeld, Ed., North-Holland, Amsterdam, 1974, pp. 556–574.
12. KOWALSKI, R.A. *Logic for Problem-Solving*. North-Holland, Amsterdam, 1979.
13. LOVELAND, D.W. *Automated Theorem Proving*. North-Holland, Amsterdam, 1978.
14. REITER, R. On closed-world data bases. In *Logic and Data Bases*, H. Gallaire and J. Minker, Eds., Plenum Press, New York, 1978, pp. 55–76.
15. ROBINSON, J.A. A machine-oriented logic based on the resolution principle. *J. ACM 12*, 1 (Jan. 1965), 23–41.
16. ROBINSON, J.A. *Logic: Function and Form*. Edinburgh University Press, Edinburgh, Scotland, 1979.
17. VAN EMDEN, M.H. Verification conditions as programs. In *Automata, Languages, and Programming*, S. Michaelson and R. Milner, Eds., Edinburgh University Press, Edinburgh, Scotland, 1976, pp. 99–119.
18. VAN EMDEN, M.H. Computation and deductive information retrieval. In *Formal Description of Programming Concepts*, E. Neuhold, Ed., North-Holland, Amsterdam, 1978, pp. 421–440.
19. VAN EMDEN, M.H., AND KOWALSKI, R.A. The semantics of predicate logic as a programming language. *J. ACM 23*, 4 (Oct. 1976), 733–742.

RECEIVED APRIL 1980; REVISED APRIL 1981; ACCEPTED MAY 1981