Theoretical Computer Science 29 (1984) 167–184
North-Holland

CLOSURES AND FAIRNESS IN THE SEMANTICS OF PROGRAMMING LOGIC*

J.-L. LASSEZ AND M. J. MAHER
Department of Computer Science, University of Melbourne, Parkville, Victoria, 3052, Australia

Communicated by Z. Manna
Received November 1982
Revised May 1983

Abstract. We use the notions of closures and fair chaotic iterations to give a semantics to logic programs. The relationships between the semantics of individual rules and the semantics of the whole program are established and an application to parallel processing is mentioned. A chaotic fixed point theorem is given, which carries the non-determinism inherent to resolution. Finally, we introduce a general definition of finite failure and the concept of fair SLD resolution, and show that this procedure is sound and strongly complete with respect to both success and finite failure, thus extending a result of Apt and Van Emden (1982).

Key words. Logic programming, semantics, chaotic iterations, SLD resolution, finite failure, PROLOG.

1. Introduction

Van Emden and Kowalski [8] gave very elegant simple fixed point and model-theoretic semantics for logic programs in definite clausal form. Apt and Van Emden [1] further exploited fixed point techniques to establish the soundness and completeness of SLD resolution with respect to success and to characterize SLD finite failure. These two papers which give the formal semantics of what could be called 'pure' PROLOG have been widely cited in the literature.

P. Cousot and R. Cousot in a long series of papers (see [6] for a bibliography) have used the notions of closure and chaotic iterations to establish, in particular, formal models of global data flow analysis.

Taking a slightly different approach from Van Emden and Kowalski's we consider a logic program to consist only of inference rules, the axioms being considered as input. The function $[P]$ representing the semantics of the program is then the closure (idempotent, increasing) of the function $[R]$ representing the semantics of the rules. This closure property allows us to use the techniques of Cousot and Cousot. Further classical fixed point theorems can then be adapted to provide simple algebraic characterizations in the semantics of logic programming.

* Research supported by the Australian Computer Research Board.

After a preliminary section in which we give the necessary definitions, notations and classical results, a first theorem is given which is the version, in our setting, of Van Emden and Kowalski's fixed point characterization of least model semantics. As a corollary we obtain the relationship between the model-theoretic and fixed point semantics of the whole program and the model-theoretic and fixed point semantics of its rules. These and the following result give a denotational semantics to logic programs in that "the meaning of any composite phrase is expressed in terms of the meanings of its immediate constituents" (cf. Tennent [17] for instance).

To establish the link between more operational semantics, we adapt techniques due to Tarski [16] and P. Cousot and R. Cousot [5] used to study common fixed points of sets of functions. The result we obtain can be considered as a particular case of chaotic iteration. We briefly mention the general case and adapt the notion to suit our presentation. We show that the fixed point semantics can be obtained by any chaotic iteration. We also semantically characterize programs amenable to division into parts which can be executed independently by parallel processors. Chaotic iterations incorporate a fair treatment of the rules, but to deal with finite failure we need a different fairness—a fairness in the selection of atoms in SLD resolution. This is treated in the final section.

Apt and Van Emden [1] provided an important characterization of SLD finite failure using fixed point techniques. We give a definition of finite failure which is independent of any particular implementation. We then show that the result of Apt and Van Emden can be interpreted as a form of soundness and completeness of SLD resolution with respect to finite failure: A is finitely failed iff it is SLD finitely failed (i.e., there exists a finitely failed SLD tree with A at the root). However, SLD resolution in general will not find this finitely failed SLD tree when it exists. We adapt SLD resolution to search for failure by introducing fair SLD resolution which is still sound and complete with respect to success and, as we prove, correctly implements finite failure: we will find a finitely failed SLD tree with A at the root iff A is finitely failed.

2. Preliminaries: Programming logic in clausal form

In this section we first give the necessary definitions that lead to Van Emden and Kowalski's [8] model for pure PROLOG. We then give the informal semantics of a program. Finally, we give the least fixed point semantics, the model-theoretic semantics and the SLD resolution semantics from Apt and Van Emden [1]. A generalized treatment is to be found in [18] and in a forthcoming volume [10]. We further refer the reader to Kowalski [9] and Van Emden [7] for logic programming and Loveland [11], Manna [12] and Robinson [13] for resolution. There are many references on PROLOG, the latest being Colmerauer [3].

Definitions. The sets of function symbols, predicate symbols and variables are disjoint sets. A constant is a 0-argument function symbol. A term is a variable or is $f(t_1, \ldots, t_n)$ where f is an n-argument function symbol and $t_1, \ldots, t_n$ are terms. An atomic formula (or atom) is $P(t_1, \ldots, t_n)$ where P is an n-argument predicate symbol and where $t_1, \ldots, t_n$ are terms. A definite clause is written as

$$A \leftarrow B_1, \ldots, B_n, \quad n \geq 0,$$

where $A, B_1, \ldots, B_n$ are atomic formulae. A program is a finite set of definite clauses. A substitution is an operation θ which, throughout a set of atomic formulae $\{A_1, \ldots, A_n\}$, replaces all occurrences of given variables by given terms, the same term being used for all occurrences of a given variable. The result denoted $\{A_1\theta, \ldots, A_n\theta\}$ is called an instance of $\{A_1, \ldots, A_n\}$. This extends naturally to clauses.

Semantics. A program is understood as the conjunction of its definite clauses, and a clause

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

is to be understood as follows: For all $x_1, \ldots, x_k$ variables in the atomic formulae $A, B_1, \ldots, B_n$ the conjunction of the B_j's logically implies A. Thus in the case when $n = 0$ we have: For all values of its variables A is true. Similarly $\leftarrow B_1, \ldots, B_n$ means that for all values of its variables the conjunction of the B_j's is not true, and is referred to as a negative clause. A negative clause with $n = 0$ is called an empty clause, and is understood as a contradiction. The Herbrand base HB of a program P is the set of all variable free atoms containing no predicate or function symbols other than those occurring in P. An interpretation is a subset of HB. We now define the notion of truth with respect to a given interpretation I:

- $A \in HB$ is true in I iff $A \in I$.
- A variable free clause $A \leftarrow B_1, \ldots, B_n$ is true in I iff A is true in I or at least one of the B_i's is not ..ue in I.
- A clause $A \leftarrow B_1, \ldots, B_n$ is true in I iff each of its variable free instances is true in I.
- Finally a program is true in I iff all its clauses are true in I. A model of a program P is an interpretation I such that P is true in I.

Clearly the set of subsets of HB forms naturally a complete lattice with set inclusion as partial order, union as lub and intersection as glb operations, the empty set as the least element and HB as the greatest. Let $M(P)$ be the set of models for a program P.

Proposition 2.1. *Any intersection of models for a program P is again a model for P. In particular $\bigcap M(P)$ is a model, the least one.*

Now a function T from interpretations to interpretations is associated to a program P in the following way:

$$A \in T(I) \text{ iff there exists in } P \text{ a clause } B_0 \leftarrow B_1, \ldots, B_n$$
$$\text{and a substitution } \theta \text{ such that } A = B_0\theta \text{ and } \{B_1\theta, \ldots, B_n\theta\} \subseteq I.$$

We have the following proposition.

Proposition 2.2. *An interpretation I of a program P is a model for P if and only if $T(I) \subseteq I$.*

The fact that T is continuous (i.e., $T(\bigcup_{i=0}^{\infty} X_i) = \bigcup_{i=0}^{\infty} T(X_i)$ for every increasing sequence $X_0 \subseteq X_1 \subseteq \cdots \subseteq X_n \subseteq X_{n+1} \subseteq \cdots$) is simply verified and we may now apply the least fixed point theorem which provides two characterizations of the least fixed point of a continuous function f over a complete lattice—an existential one as the least element x such that $f(x) \leqslant x$, and a constructive one as lub$\{f^n(\perp)$: $n \geqslant 0\}$ where $\perp$ is the least element of the lattice.

Theorem 2.3. *The function T associated to a program P has a least fixed point* lfp(T) *which can be characterized the following ways:*

$$\text{lfp}(T) = \bigcap M(P) = \bigcup_{n=0}^{\infty} T^n(\emptyset).$$

Therefore, we see that the semantics defined as least model of P or least fixed point of T are the same. To establish operationally that $A \in \bigcap M(P)$ one may use SLD resolution:

An SLD derivation of $P \cup \{\leftarrow A\}$ consists of a sequence of $N_0, \ldots, N_m$ of negative clauses (that is clauses without consequences: $\leftarrow B_1, \ldots, B_n$) with $N_0 = \leftarrow A$, a sequence $d_1, \ldots, d_n$ of variants of clauses of P (a variant d_i is obtained from a clause of P by renaming variables so that it has no variables in common with the negative clause N_i), and finally a sequence of substitutions $\theta_1, \ldots, \theta_n$. In each negative clause N_i a distinguished atom is considered, called the selected atom. The clause N_{i+1} is said to be derived from N_i and θ_{i+1}. This relationship is defined below.

Let $N_i = (\leftarrow A_1, \ldots, A_k, \ldots, A_p)$, $p > 0$, with A_k as the selected atom. Let $d_{i+1} = (A \leftarrow B_1, \ldots, B_q)$, $q \geqslant 0$, be the variant of any clause in P such that $A\theta = A_k\theta$ for some substitution θ. Then

$$N_{i+1} = (\leftarrow A_1\theta, \ldots, A_{k-1}\theta, B_1\theta, \ldots, B_q\theta, A_{k+1}\theta, \ldots, A_p\theta) \quad \text{and} \quad \theta_{i+1} = \theta.$$

If a derivation contains the empty clause, it must be its last clause. Such a derivation is called an SLD refutation. When there is a substitution θ so that $A\theta = A_k\theta$, A and A_k are said to be unifiable. A substitution ξ such that $A\xi = A_k\xi$ and if $A\theta = A_k\theta$, then $\theta = \xi\eta$ for some substitution η is called a most general unifier (mgu) of A and A_k.

The success set of a program P is the interpretation S such that $A \in S$ iff $P \bigcup \{\leftarrow A\}$ has an SLD refutation. As one can show that the least model $\bigcap M(P)$ is the set of atoms logically implied by P, the following theorem can be interpreted as expressing the soundness and completeness of SLD refutations, as well as expressing the equivalence of model-theoretic and operational semantics. The proof of this result by Apt and Van Emden [1] is a good illustration of the use of fixed point techniques.

Theorem 2.4. *The success set of a program P is equal to its least model.*

3. Closure semantics

The classical notion of closure has been used extensively by P. Cousot and R. Cousot [5] and Scott [15]. In order that this notion appears naturally, we will modify the previous definition of a program, but first we solicit intuitive support of these modifications with an example:

$\text{path}(A, B) \leftarrow$

$\text{path}(B, C) \leftarrow$

$\text{path}(D, E) \leftarrow$

$\text{path}(x, y) \leftarrow \text{path}(y, x)$

$\text{path}(x, z) \leftarrow \text{path}(x, y), \text{path}(y, z)$

The above simple program determines all the paths in the graph defined by the first three clauses. It is clear that the nature of these three clauses differs from that of the remaining two—the latter would appear in any program of this sort whereas the former specify one particular graph. Hence, it seems natural to separate the two types of clauses in the following definitions.

Let $A \leftarrow B_1, \ldots, B_n$ be a definite clause C. If $n = 0$, C is called an axiom; if $n > 0$, C is called a rule. We now consider a program P to be the conjunction of a finite set of rules R. We assume the existence of a suitable Herbrand base. We will associate to a program P two functions from interpretations to interpretations: $[R]$ which is defined as T was previously (that is, $\forall I \subseteq HB$, $A \in [R](I)$ iff there is a rule $B_0 \leftarrow B_1, \ldots, B_n$ and a substitution θ such that $B_0\theta = A$ and $\{B_1\theta, \ldots, B_n\theta\} \subseteq I$) and $[P]$ which assigns to any subset $Ax \subseteq HB$ the set of atoms that can be generated by repeated applications of the rules.

We now look for a recursive definition of $[P]$. Let $Ax \subseteq HB$. We want $[P](Ax)$ to be the limit of the following sequence:

$$V_0 = Ax, \qquad V_{i+1} = [R](V_i) \cup V_i.$$

Letting Id represent the identity function on interpretations, we find that $[P]$ must satisfy the recursive definition:

$$[P] = [R] \circ [P] + [P],$$

that is,

$$[P]=([R]+\mathrm{Id})\circ[P]$$

(where $f+g$ is defined as $(f+g)(I)=f(I)\cup g(I)$ and $\circ$ denotes function composition). Now $[P]$ is not the least fixed point of this equation, but we can find a more standard least fixed point semantics for $[P]$ by using an equivalent definition for V_{i+1}.

By definition,

$$V_{i+1}=[R](V_i)\cup V_i=[R](V_i)\cup[R](V_{i-1})\cup V_{i-1}$$

As $[R]$ is continuous and $\{V_i\}$ is a chain we have

$$[R](V_i)\supseteq[R](V_{i-1})\supseteq\cdots\supseteq[R](Ax)$$

so

$$V_{i+1}=[R](V_i)\cup V_{i-1}=\cdots=[R](V_i)\cup Ax.$$

Therefore, the sequence becomes

$$V_0=Ax,\qquad V_{i+1}=[R](V_i)\cup Ax,$$

from which we derive another recursive equation that $[P]$ must satisfy

$$[P](Ax)=[R]\circ[P](Ax)\cup Ax,$$

that is,

$$[P]=[R]\circ[P]+\mathrm{Id}.$$

One easily verifies that the operator $\tau: f\to[R]\circ f+\mathrm{Id}$ is continuous and we have the following equivalent formal semantics for $[P]$. We are using the notation f^ω for the function defined as $f^\omega(A)=\bigcup_{n=0}^{\infty}f^n(A)$.

Proposition 3.1. *The two following definitions of $[P]$ are equivalent:*
(1) $[P]=([R]+\mathrm{Id})^\omega$ *(so $[P]$ is a solution of* $[P]=([R]+\mathrm{Id})\circ[\mathrm{P}]$).
(2) $[P]$ *is the least fixed point of the operator* $\tau(f)=[R]\circ f+\mathrm{Id}$.

A closure [5, 15] is a monotonic function which is idempotent ($f\circ f=f$) and increasing ($f(x)\geq x\ \forall x$). We have the following proposition.

Proposition 3.2. *$[P]$ is a continuous closure.*

Proof. $[P]$ is clearly increasing. Note that $([R]+\mathrm{Id})$ is continuous. To show that $[P]$ is continuous, let $\{I_i\}$, $i=0,1,\ldots,$ be an increasing sequence with $\bigcup_{i=0}^{\infty}I_i=$

$I \subseteq HB$. Then

$$[P](I) = \bigcup_{n=0}^{\infty} ([R]+\mathrm{Id})^n(I) = \bigcup_{n=0}^{\infty} \bigcup_{i=0}^{\infty} ([R]+\mathrm{Id})^n(I_i)$$

$$= \bigcup_{i=0}^{\infty} \bigcup_{n=0}^{\infty} ([R]+\mathrm{Id})^n(I_i)$$

$$= \bigcup_{i=0}^{\infty} [P](I_i).$$

For idempotence,

$$[P]\circ[P](I) = [P]\circ\left(\bigcup_{n=0}^{\infty} ([R]+\mathrm{Id})^n(I)\right)$$

$$= \bigcup_{m=0}^{\infty} ([R]+\mathrm{Id})^m\left(\bigcup_{n=0}^{\infty} ([R]+\mathrm{Id})^n(I)\right)$$

$$= \bigcup_{m=0}^{\infty} \bigcup_{n=0}^{\infty} ([R]+\mathrm{Id})^{m+n}(I))$$

$$= \bigcup_{k=0}^{\infty} ([R]+\mathrm{Id})^k(I) = [P](I). \qquad \square$$

In fact one can also prove that $\Psi:[R]\to[P]$ is also a continuous closure and that $[P]$ is the closest closure to $[R]$ in the sense that every closure above $[R]$ is above $[P]$.

4. Model and fixed point semantics

Similarly to Van Emden and Kowalski we define an interpretation I to be a model of P iff P is true in I. Note that the difference comes from the fact that now a program is restricted to the conjunction of rules. We easily verify the following proposition.

Proposition 4.1. *The following statements are equivalent*:

(1) *I is a model of P.*

(2) $I \supseteq [R](I)$.

(3) *I is a fixed point of* $[P]$.

Now the equality between model, fixed point, and closure semantics is given by the following theorem.

Theorem 4.2. *$[P](Ax)$ is the least fixed point of $[P]$ which contains Ax.*

$[P](Ax)$ is the least model of P which contains Ax.

Proof. Let I be a fixed point of $[P]$ containing Ax. Then $[R](I) \subseteq I$ and so $([R]+\mathrm{Id})(I) = I$. By monotonicity, if $Ax \subseteq I$, then

$$([R]+\mathrm{Id})^n(Ax) \subseteq ([R]+\mathrm{Id})^n(I) = I \quad \forall n.$$

Hence $[P](Ax) \subseteq I$, and since $[P]$ is idempotent and increasing, the first part holds. The second part follows from the equivalence of (1) and (3) in the above proposition. □

Let T be the function associated to $R \cup Ax$. The equivalence between the closure semantics and the semantics of Van Emden and Kowalski is shown by the following proposition.

Proposition 4.3. $[P](Ax) = T^{\omega}(\emptyset)$.

Proof. It is clear from the definitions of T and $[R]$ that $T = [R] + [Ax]$, where $[Ax]$ is the constant function $[Ax](I) = Ax$, $\forall I \subseteq HB$. Let V_i be defined as in Section 3. Then

$$V_{i+1} = [R](V_i) \cup Ax = [R](V_i) \cup [Ax](V_i) = T(V_i).$$

Also $V_0 = Ax = T(\emptyset)$. Hence, by induction, $V_i = T^{i+1}(\emptyset)\ \forall i$ and so $[P](Ax) = T^{\omega}(\emptyset)$. □

Let $R = \{R_1, \ldots, R_n\}$ be a set of rules, P the program associated with R and $P_1, \ldots, P_n$ the programs associated respectively with the individual rules $R_1, \ldots, R_n$. We now give the denotational version of the least fixed point and model-theoretic semantics.

Theorem 4.4. *$[P](Ax)$ is the least common fixed point of the functions $[P_1], \ldots, [P_n]$, which contains Ax.*

$[P](Ax)$ is the least common model of the programs $P_1, \ldots, P_n$, which contains Ax.

Proof. Clearly $[R] = \sum_{i=1}^{n} [R_i]$ so $[R](I) \subseteq I$ iff, for $i = 1, \ldots, n$, $[R_i](I) \subseteq I$. Hence, using Proposition 4.1 for P and P_i, $i = 1, \ldots, n$, $[P](I) = I$ iff $[R](I) \subseteq I$ iff $[R_i](I) \subseteq I$, $i = 1, \ldots, n$, iff $[P_i](I) = I$, $i = 1, \ldots, n$, iff I is a model of P_i, $i = 1, \ldots n$. Hence, the fixed points of $[P]$ are exactly the common fixed points of the $[P_i]$'s which are the common models of the P_i's. Since $[P](Ax)$ is the least fixed point of $[P]$ which contains Ax, the results follow. □

A classic result of lattice theory (Tarski [16], P. Cousot and R. Cousot [5]) tells us that a set of commuting monotonic functions has a complete lattice of common fixed points. The constructive characterization of the least common fixed point is obtained by taking the least fixed point of the composition of all functions in the set. We can adapt this technique to our present situation where the notion of increasing replaces the commutativity hypothesis. The following theorem gives us the semantics of the conjunction of rules. We use $\prod_{k=1}^{n} f_k$ for the composition $f_1 \circ \cdots \circ f_n$.

Theorem 4.5

$$[P]=[R_1 \wedge \cdots \wedge R_n]=\left(\prod_{k=1}^{n}[P_k]\right)^{\omega}=\left(\prod_{k=1}^{n}([R_k]+\mathrm{Id})^{\omega}\right)^{\omega}.$$

Proof. $\sum_{k=1}^{n}([R_k]+\mathrm{Id})\leqslant\sum_{k=1}^{n}[P_k]$ since $([R_k]+\mathrm{Id})\leqslant[P_k]$, $k=1,\ldots,n$, and $\sum_{k=1}^{n}[P_k]\leqslant\prod_{k=1}^{n}[P_k]$ since $[P_k]$ is increasing, $k=1,\ldots,n$. Hence

$$\begin{aligned}[P]=([R]+\mathrm{Id})^{\omega}&=\left(\sum_{k=1}^{n}([R_k]+\mathrm{Id})\right)^{\omega}\\ &\leqslant\left(\sum_{k=1}^{n}[P_k]\right)^{\omega}\leqslant\left(\prod_{k=1}^{n}[P_k]\right)^{\omega}\\ &\leqslant\left(\prod_{k=1}^{n}[P]\right)^{\omega}=[P]^{\omega}=[P].\end{aligned}$$

Hence $[P]=(\prod_{k=1}^{n}[P_k])^{\omega}$ and since $[P_k]=([R_k]+\mathrm{Id})^{\omega}$, the result follows. □

This theorem is, for Programming Logic, analogous to $[S_1;S_2]=[S_2]\circ[S_1]$ in the denotational semantics of a sequential programming language. These theorems remain valid if $\{R_1,\ldots,R_n\}$ represents any (not necessarily disjoint) division of the set of rules instead of the one above. An interesting case is when the program P can be divided into two (not necessarily disjoint) subprograms P_1 and P_2 such that $[P]=[P_1]+[P_2]$. This can always be enforced by taking the trivial solution $P_1=P$. A program may have only such trivial decompositions. Another extreme case is when R_1 and R_2 use different predicates. In this case, SLD resolution will automatically use the relevant subset of rules, ignoring the other. In the other cases SLD resolution can be used in parallel for P_1 and P_2, the two search spaces involved being in general far smaller than the whole search space for P, the construction of which necessitates combinations of rules from both programs. In any case we have the following characterization of decomposition.

Theorem 4.6

$$[P]=[P_1]+[P_2] \text{ \textit{iff} } [P_1]\circ[P_2]=[P_2]\circ[P_1]=[P_1]+[P_2].$$

Proof. We know from the previous theorem that $[P]=([P_1]\circ[P_2])^{\omega}$. Using monotonicity, if $[P]=[P_1]+[P_2]$, then

$$[P]=[P_1]+[P_2]\leqslant[P_1]\circ[P_2]\leqslant([P_1]\circ[P_2])^{\omega}=[P].$$

Hence, by symmetry,

$$[P_1]\circ[P_2]=[P_2]\circ[P_1]=[P_1]+[P_2].$$

Using idempotence, if

$$[P_1]\circ[P_2]=[P_2]\circ[P_1]=[P_1]+[P_2],$$

then

$$([P_1]\circ[P_2])^n=[P_1]^n\circ[P_2]^n=[P_1]\circ[P_2]=[P_1]+[P_2]\quad\forall n.$$

Hence,

$$[P]=([P_1]\circ[P_2])^\omega=[P_1]+[P_2].\quad\Box$$

In fact if program P is divided into n subprograms, then $[P]=\sum_{i=1}^n[P_i]$ iff $\prod_{i=1}^n[P_{\rho(i)}]=\sum_{i=1}^n[P_i]$ for all permutations ρ on $\{1,\ldots,n\}$. At the level of the rules one obtains the following.

Theorem 4.7. *If*

$$\begin{aligned}([R_1]+\mathrm{Id})\circ([R_2]+\mathrm{Id})&=([R_2]+\mathrm{Id})\circ([R_1]+\mathrm{Id})\\&=([R_1]+\mathrm{Id})+([R_2]+\mathrm{Id}),\end{aligned}$$

then $[P]=[P_1]+[P_2]$.

Proof. Let $T_i=([R_i]+\mathrm{Id}), i=1,2$. Then, for $i=1,2$, T_i commutes with T_1+T_2 since

$$T_i\circ(T_1+T_2)=T_i\circ T_1\circ T_2=T_1\circ T_2\circ T_i=(T_1+T_2)\circ T_i.$$

We prove by induction that $(T_1+T_2)^k=T_1^k+T_2^k$. This is clearly true for $k=0$ and $k=1$ and if it is true for all $k\leq n$, then

$$\begin{aligned}(T_1+T_2)^{n+1}&=T_1\circ(T_1+T_2)^n+T_2\circ(T_1+T_2)^n\\&=(T_1+T_2)^n\circ T_1+(T_1+T_2)^n\circ T_2&&\text{by commutativity}\\&=(T_1^n+T_2^n)\circ T_1+(T_1^n+T_2^n)\circ T_2&&\text{by induction hypothesis}\\&=T_1^{n+1}+T_2^n\circ T_1+T_1^n\circ T_2+T_2^{n+1}\\&=T_1^{n+1}+(T_2^{n-1}+T_1^{n-1})\circ T_1\circ T_2+T_2^{n+1}\\&=T_1^{n+1}+(T_1+T_2)^n+T_2^{n+1}&&\text{by induction hypothesis}\\&=T_1^{n+1}+T_1^n+T_2^n+T_2^{n+1}&&\text{by induction hypothesis}\\&=T_1^{n+1}+T_2^{n+1}.\end{aligned}$$

Hence $\forall I \subseteq HB$,

$$\begin{aligned}[P](I) &= \bigcup_{n=0}^{\infty} (T_1 + T_2)^n(I) = \bigcup_{n=0}^{\infty} (T_1^n + T_2^n)(I)\\ &= \bigcup_{n=0}^{\infty} (T_1^n(I) \cup T_2^n(I)) = \bigcup_{n=0}^{\infty} T_1^n(I) \cup \bigcup_{n=0}^{\infty} T_2^n(I)\\ &= [P_1](I) \cup [P_2](I) = ([P_1]+[P_2])(I). \qquad \square\end{aligned}$$

This sufficient condition is not a necessary condition since we can take $R_2 = R_1$, in which case $[P]=[P_1]+[P_2]$ but, in general,

$$([R_1]+\mathrm{Id}) \circ ([R_2]+\mathrm{Id}) \neq ([R_1]+\mathrm{Id}) + ([R_2]+\mathrm{Id}).$$

However, a generalization similar to that of the previous theorem holds.

Example 4.8. Consider $R=\{N(x) \leftarrow N(S(x)),\ N(S(x)) \leftarrow N(x)\}$ where the Herbrand base HB is $\{N(S^n(0))\}$, $n=0,1,\ldots$. Let Ax be any subset of the Herbrand base. The Van Emden–Kowalski semantics $\bigcup_{m=0}^{\infty} T^m(\emptyset)$ is built step by step by adding to a set the immediate predecessors and the immediate successors of its elements. In comparison, the semantics

$$[P](Ax) = ([P_1] \circ [P_2])^{\omega}(Ax) = (([R_1]+\mathrm{Id})^{\omega} \circ ([R_2]+\mathrm{Id})^{\omega})^{\omega}(Ax)$$

is obtained by changing the synchronization between the applications of the rules in the following way: We take the closure of the set under the successor operation, then the closure of the resulting set under the predecessor operation. In this particularly simple case we do not need to repeat the process since we have reached the fixed point HB after ω applications of R_2 and a finite number of applications of R_1.

One also verifies easily that $[P_1] \circ [P_2] = [P_2] \circ [P_1] = [P_1]+[P_2]$. Therefore, one can break the connection between the two rules and perform SLD resolution on each of them separately. That is, for a given $A \in HB$ we look simultaneously in parallel for successors only and for predecessors only. Without this split, SLD resolution would search alternatively for predecessors and successors creating a search space 'exponentially' larger than the two preceding ones.

We see from this example that changes in the order of application of the rules do not necessarily affect the least fixed point. We are, in fact, in a particular case of chaotic iterations.

5. Chaotic iterations

P. Cousot and R Cousot [4] adapted and generalized the concept of chaotic iterations from numerical analysis to lattice theory to generate fixed points of functions with multiple arguments. In this context the interpolation of their result would be, informally, that $[P](Ax)$ may be reached by transfinite repetitive applications of the rules in *any* order, provided a fairness condition is met: For any rule R_i, for any ordinal α there exists an ordinal $\beta \geqslant \alpha$ such that the rule R_i will be applied at 'time' β, assuming that each successor ordinal corresponds to the application of a rule. Other fairness conditions can be considered and we adopt one here which will be more in tune with SLD resolution for which there is no need to go beyond ω.

Definitions. Let $R = \{R_1, \ldots, R_n\}$ be a set of rules and P the associated program. A fair sequence FS is a sequence of functions $\{f_k\}$, $k = 0, 1, \ldots$, such that

$$(\forall k \quad f_k = [R_i] + \mathrm{Id} \text{ for some } i,\ 1 \leqslant i \leqslant n,$$

and

$$(\forall i,\ 1 \leqslant i \leqslant n)(\forall l)(\exists m > i) \quad f_m = [R_i] + \mathrm{Id}.$$

A chaotic iteration for P defined by a fair sequence FS, starting at $Ax \subseteq HB$ is the sequence of subsets of HB given by the recurrence relation

$$X_0 = Ax, \qquad X_{i+1} = f_i(X_i).$$

Clearly $X_{i+1} \supseteq X_i$ and we define the limit of the chaotic iteration to be $\bigcup_{i=0}^{\infty} X_i$.

We can now state the chaotic fixed point theorem which introduces non-determinism similar to that inherent in resolution into the sequence converging to the least fixed point above Ax.

Theorem 5.1. *Every chaotic iteration for a program P starting at $Ax \subseteq HB$ converges to $[P](Ax)$.*

Proof. Let $\{X_k\}$, $k = 0, 1, \ldots$, be an arbitrary chaotic iteration defined by $\{f_k\}$, $k = 0, 1, \ldots$. Recall from the proof of Theorem 4.4 that $[P](I) = I$ iff $[R_i](I) \subseteq I$, $i = 1, \ldots, n$, iff $([R_i] + \mathrm{Id})(I) = I$, $i = 1, \ldots, n$, so that, for any fixed point I of $[P]$, if $X_k \subseteq I$, then $X_{k+1} = f_k(X_k) \subseteq f_k(I) = I\ \forall k$. Since $Ax \subseteq [P](Ax)$, $X_k \subseteq [P](Ax)\ \forall k$ and so $\bigcup_{k=0}^{\infty} X_k \subseteq [P](Ax)$. The sequence $\{f_k\}$, $k = 0, 1, \ldots$, is fair so

$$(\forall i,\ 1 \leqslant i \leqslant n)(\forall k)(\exists m > k) \quad X_{m+1} = ([R_i] + \mathrm{Id})(X_m).$$

Since $\{X_k\}$, $k = 0, 1, \ldots$, is increasing and $([R_i] + \mathrm{Id})$ is monotonic,

$$(\forall i,\ 1 \leqslant i \leqslant n)(\forall k)(\exists m > k) \quad ([R_i] + \mathrm{Id})(X_k) \subseteq X_{m+1}.$$

Hence

$$([R_i] + \mathrm{Id})\left(\bigcup_{k=0}^{\infty} X_k\right) = \bigcup_{k=0}^{\infty} ([R_i] + \mathrm{Id})(X_k) \subseteq \bigcup_{k=0}^{\infty} X_k \subseteq ([R_i] + \mathrm{Id})\left(\bigcup_{k=0}^{\infty} X_k\right).$$

Hence $\bigcup_{k=0}^{\infty} X_k$ is a common fixed point of the $([R_i] + \mathrm{Id})$'s, so $[P](Ax) \subseteq \bigcup_{k=0}^{\infty} X_k$. Hence $[P](Ax) = \bigcup_{k=0}^{\infty} X_k$. $\square$

In the informal semantics we want to generate all true facts from the axioms by repeated applications of the rules. No particular order or synchronization of the rules is specified. Chaotic iterations model this more closely than the functions T and $[R]$, in which the rules are applied simultaneously. In this respect the above theorem provides a further justification for the use of $[R]$ and T.

6. Finite failure and SLD finite failure

In the language of automatic theorem proving, $[P](Ax)$ represents the set of theorems proved for the system (R, Ax). An element A which does not belong to $[P](Ax)$ is a non-theorem. In attempting to prove A as a theorem, it may be that a theorem prover will exhaust all possibilities of proof in a finite time. In this case we can deduce that A is a nontheorem. The finite failure set that we will introduce corresponds to this set of facts which we can prove are non-theorems. From an implementation point of view, it is preferable to have a theorem prover which will halt when given such a fact to prove, rather than proceed with an infinite computation. As these non-theorems can be used to infer falsehood, the concept of finite failure is also important from a semantic point of view. We now revert to Van Emden and Kowalski's definition of program (i.e., a program consists of rules *and* axioms).

As a procedure to find SLD refutations, Clark [2] and Apt and Van Emden [1] construct SLD trees. When this construction fails in a finite time to find a refutation a finitely failed SLD tree is obtained. This concept is used in PROLOG to infer negation. Clark provided this use with a formal justification.

Therefore, in [1] and [2] the concept of finite failure has been defined only in the context of SLD resolution. In this section we define finite failure in general and show that it is clearly equivalent to Apt and Van Emden's 'fixed point' characterization of SLD finite failure. This leads us to reinterpret Apt and Van Emden's result as proving the soundness and weak completeness of SLD resolution with respect to finite failure. We then extend this result by introducing a simple condition of fairness. The resulting fair SLD resolution is sound and strongly complete with respect to finite failure and so will always find a finitely failed SLD tree if one exists. We restrict our attention to SLD resolutions in which every substitution is an mgu, as is done in [1]. These resolutions are sufficient to obtain a refutation if one exists. The definition of finite failure in terms of SLD resolution [1] can be stated as follows (our terminology differs slightly from that of [1]).

Definitions. An SLD tree for a negative clause N for a program P is a tree with N at the root and with a negative clause $\leftarrow A_1, \ldots, A_m$ and selected atom A_k (if $m > 0$) at every node. Each node has a descendant $(\leftarrow A_1, \ldots, A_{k-1}, B_1, \ldots, B_n, A_{k+1}, \ldots, A_m)\theta$ for every clause (rule or axiom) $B_0 \leftarrow B_1, \ldots, B_n$ of P such that A_k and B_0

are unifiable, where θ is an mgu. An SLD tree is finitely failed if it is finite and does not contain the empty clause. A negative clause N is SLD finitely failed (sldff) if there is a finitely failed SLD tree which has N as its root. The SLD finite failure set is the set of atoms $A \in HB$ such that $\leftarrow A$ is sldff.

Apt and Van Emden obtain, after a series of involved lemmas, the following characterization of SLD finite failure as one of their main theorems.

Theorem 6.1. *Let P be a definite sentence of finite degree. The* SLD *finite failure set of P is the complement in HB of $T\downarrow\omega$.*

Here $T\downarrow\omega = \bigcap_{n=0}^{\infty} T^n(HB)$. The concept of definite sentence is a generalization of program which allows an infinite set of definite clauses. A definite sentence is 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. Equivalently and independently of SLD resolution, a definite sentence is of finite degree if every predicate symbol appears in the conclusion of only finitely many clauses.

We now give a general definition of finite failure. For the remainder of this paper we implicitly refer to a program P with rules R and set of axioms which defines a subset Ax of HB. Although we consider programs with only a finite number of clauses, the proofs in this section also hold for definite sentences of finite degree. It is clear that $A \in HB$ is finitely failed if A does not unify with any conclusion of a rule nor any axiom. Now if the only sets of variable free atoms which imply A contain a finitely failed atom, then A must also be finitely failed. Formally, we can state the following.

Definition. For a given program P, an element A of HB is finitely failed ($A \in \mathrm{FF}$) iff $\exists d$: A is failed by depth d($A \in \mathrm{FF}_d$).

Failure by depth d is defined recursively by: $A \in \mathrm{FF}_0$ iff $A \notin [R](HB)$ and $A \notin Ax$.

For $d > 0$, $A \in \mathrm{FF}_d$ iff for every rule $(B_0 \leftarrow B_1, \ldots, B_q)$ in R and variable free substitution θ, if $A = B_0\theta$, then $(\exists k)(1 \leq k \leq q)$ and $B_k\theta \in \mathrm{FF}_m$ for some $m < d$.

The finite failure set FF is closely related to a set constructed by Sato [14]. If $A \in \mathrm{FF}_m$, then $A \in \mathrm{FF}_d\ \forall d \geq m$ (from the definition of FF_d). With this it is easy to verify that, for $d > 0$, $A \in \mathrm{FF}_d$ iff, for every rule $(B_0 \leftarrow B_1, \ldots, B_q)$ in R and variable free substitution θ with $A = B_0\theta$, $(\exists k)(1 \leq k \leq q)$ and $B_k\theta \in \mathrm{FF}_{d-1}$. This property will be used in the proof of the following proposition. ($\bar{S}$ denotes the complement of S in HB.)

Proposition 6.2

$$\mathrm{FF} = \bigcup_{n=0}^{\infty} \overline{T^n(HB)} = \overline{T\downarrow\omega}.$$

Proof. We prove by induction that $\mathrm{FF}_d = \overline{T^{d+1}(HB)}$. This is immediate for $d=0$. Now $A \notin T^{d+1}(HB)$ iff (by definition) one cannot find a rule $B_0 \leftarrow B_1, \ldots, B_q$ in R and a variable free substitution θ such that $A = B_0\theta$ and $(\forall k)(1 \leq k \leq q)$, $B_k\theta \in T^d(HB)$ iff for every rule $B_0 \leftarrow B_1, \ldots, B_q$ in R and variable free substitution θ, $A \neq B_0\theta$ or $(\exists k)(1 \leq k \leq q)$, $B_k\theta \notin T^d(HB)$ iff (by induction hypothesis) for every rule $B_0 \leftarrow B_1, \ldots, B_q$ in R and variable free substitution θ, $A \neq B_0\theta$ or $(\exists k)$ $(1 \leq k \leq q)$ and $B_k\theta \in \mathrm{FF}_{d-1}$ iff (by definition) $A \in \mathrm{FF}_d$. The result easily follows. □.

Therefore $\overline{T\downarrow\omega}$ is an alternative definition of finite failure, independent of any implementation. In this context, the Apt–Van Emden result (Theorem 6.1) can be considered as a form of soundness and completeness for an SLD implementation of finite failure. If A is SLD finitely failed, it is indeed finitely failed (soundness). If A is finitely failed, Theorem 6.1 and the definition of SLD finite failure guarantee only the existence of a finitely failed SLD tree among others that may be infinite (see Example 6.3 below). In that respect, it is a weak form of completeness; the use of SLD resolution does not guarantee that a finitely failed tree will be found when one exists. By introducing a fairness condition, we adapt SLD resolution to search for failure as well as success. For this fair SLD resolution, we obtain a strong form of completeness: If A is finitely failed, then there exists a finitely failed SLD tree and all fair SLD trees are finitely failed. Therefore, the use of fair SLD resolution guarantees that a finitely failed SLD tree will be found when one exists.

Definitions. For a given SLD derivation $\langle N_i, d_i, \theta_i\rangle$, an atom B in N_k is an instantiated copy of an atom A in N_m, $m \leq k$, if there is a sequence of $k-m+1$ atoms $\{B_j\}$ with $B_0 = A$, $B_{k-m} = B$ and B_j is in N_{m+j} and $B_j = B_{j-1}\theta_{m+j}$, $j = 1, \ldots, k-m$.

An initial atom in an SLD derivation has level 0. An atom has level n if it is an instantiated copy of an atom which was introduced by expanding an atom of level $n-1$.

An SLD derivation is fair if, for every atom B in the derivation, either some instantiated copy is selected within a finite number of steps or some instantiated copy is in a failed clause (i.e., a clause where the selected atom does not unify with the conclusion of any clause in P)

A computation rule chooses the selected atom from a negative clause. A computation rule is fair if every SLD derivation produced from it is fair.

Example 6.3. Assume that C does not unify with any conclusion of a rule nor any axiom of a program which contains the following rules

$$A(x) \leftarrow B(x), C$$

$$A(x) \leftarrow B(x)$$

$$B(x) \leftarrow A(x)$$

$$B(x+1) \leftarrow A(x)$$

The derivations $\leftarrow \underline{A}(1); \leftarrow \underline{B}(1), C; \leftarrow A(1), \underline{C}$ and $\leftarrow \underline{A}(1); \leftarrow \underline{B}(1); \cdots; \leftarrow \underline{A}(1);$

$\leftarrow \underline{B}(\underline{1});\cdots$ and $\leftarrow \underline{A}(\underline{1});\leftarrow \underline{B}(\underline{1}), C;\leftarrow \underline{A}(\underline{0}), C;\leftarrow B(0), C, \underline{C}$ are all fair (selected atoms are underlined). However, the derivation $\leftarrow \underline{A}(\underline{1});\leftarrow \underline{B}(\underline{1}), C;\leftarrow \underline{A}(\underline{1}), C;\cdots;$ $\leftarrow \underline{B}(\underline{1}), C, \ldots, C; \leftarrow \underline{A}(\underline{1}), C, \ldots, C;\cdots$ is not fair since C is never used as the selected atom. Hence, for this program the computation rule which expands the first atom in a negative clause and places the introduced atoms at the front of the negative clause (as is done in many implementations of PROLOG) is not a fair computation rule. Note that each C in a given negative clause has a different level.

The elimination of 'unfair' SLD resolution is justified, as an unfair computation rule systematically ignores potentially relevant information. In the unfair derivation of the above example the fact that C cannot be proved is ignored. Now the following theorem shows that finite failure guarantees that *all*, but the unfair, SLD trees are finitely failed. Furthermore, its corollary shows that the existence of a single finitely failed SLD tree also guarantees that all, but the unfair, SLD trees are finitely failed.

It is possible for an SLD derivation to reuse a variable which has previously been instantiated, for efficiency reasons. For the sake of simplicity in the presentation of the following proof, we do not allow this.

Theorem 6.4. *Fair* SLD *resolution is sound and complete with respect to finite failure.*

Proof. *Completeness*: Let $A \in HB$ and suppose there is a fair computation rule which starts with $\leftarrow A$ and does not give a finitely failed SLD tree and does not succeed (i.e., the SLD tree is infinite). Then there is an infinite fair SLD derivation $\{\langle N_i, d_i, \theta_i\rangle\}$, $i = 0, 1, \ldots$ (by König's Lemma). By the remark before this theorem, the θ_i's each act on different variables. We need to show that $A \notin \mathrm{FF}$. From Proposition 6.2 it is equivalent to show that $A \in T^d(HB)$ for every d. In order to apply T we need variable free atoms and so any variable remaining in $\{N_i\theta_1 \ldots \theta_p\}$, $i = 0, 1, \ldots, p$, is replaced by an arbitrary variable free term. Thus let $\theta = \theta_1 \ldots \theta_p\sigma$, where σ is some variable free substitution on $\{N_i\theta_1 \ldots \theta_p\}$, $i = 0, 1, \ldots, p$. For all atoms G of level d in the derivation, G has been expanded within p steps and our choice of θ implies that $G\theta$ unifies with the conclusion of some clause and so $G\theta \in T(HB)$. Let E be an atom in the derivation of level $d-1$. Then it has been expanded into $F_1, \ldots, F_k$, $k \geq 0$, atoms of level d. Since $F_1, \ldots, F_k$ are atoms of level d, $\{F_1\theta, \ldots, F_k\theta\} \subseteq T(HB)$ and so $E\theta \in T^2(HB)$. By performing this process d times we obtain $A \in T^{d+1}(HB)$, i.e., $A \notin \mathrm{FF}_d$. But d was arbitrary so $A \notin \mathrm{FF}_d\ \forall d \geq 0$, i.e., $A \notin \mathrm{FF}$.

Soundness: Theorem 7.1 of [1] and Proposition 6.2 show that SLD resolution is sound with respect to finite failure, so certainly fair SLD resolution is sound. □

A consequence of this proof is the following corollary which shows that fair SLD trees are equivalent with respect to finite failure in a manner similar to SLD trees being equivalent with respect to success.

Corollary 6.5. *If there is a finitely failed* SLD *tree, then every fair* SLD *tree is finitely failed.*

Proof. From the proof of soundness in the previous theorem, if $\leftarrow A$ is sldff, then $A \in \text{FF}$ and so, by completeness, $\leftarrow A$ is finitely failed for any fair computation rule. □

It is immediate from the soundness and strong completeness of general SLD resolution with respect to success [1] that fair SLD resolution also has these properties. Hence, in fair SLD resolution we have a procedure which is sound and strongly complete with respect to both success and finite failure.

Acknowledgment

We wish to thank K. Apt for providing much insight and his comments on a draft of this paper and J.F. Perrot, J. Lloyd and J. Schultz for many valuable discussions. The first author wants to thank M. Nivat for making possible a four-month stay at the LITP, during which this work was initiated. We aiso wish to thank the referees for their careful reviewing.

References

[1] K.R. Apt and M.H. van Emden, Contributions to the theory of logic programming, *J. ACM* **29** (1982) 841–862.

[2] K.L. Clark, Negation as failure, in: H. Gallaire and J. Minker, eds., *Logic and Data Bases* (Plenum Press, New York, 1978) pp. 293–324.

[3] A. Colmerauer, PROLOG II manuel de référence et modèle, théorique, Rapp. Rech. GIA ERA CNRS 363, Université Aix-Marseille II, 1982.

[4] P. Cousot and R. Cousot, Automatic synthesis of optimal invariant assertions: Mathematical foundations, *ACM Symp. on Artificial Intelligence and Programming Languages*; *SIGPLAN Notices* **12** (8) (1977) 1–12.

[5] P. Cousot and R. Cousot, Constructive versions of Tarski's fixed point theorems, *Pacific J. Math.* **82** (1) (1979) 43–57.

[6] P. Cousot, Semantic foundations of program analysis, in. S.S. Muchnick and N.D. Jones, eds., *Program Flow Analysis: Theory and Application* (Prentice-Hall, London, 1981) pp. 303–342.

[7] M.H. Van Emden, Programming with resolution logic, in: E.W. Elcock and D. Michie, eds., *Machine Intelligence Vol. 8* (Ellis Horwood, Chichester, 1977) pp. 266–299.

[8] M.H. Van Emden and R.A. Kowalski, The semantics of predicate logic as a programming language, *J. ACM* **23** (1976) 733–742.

[9] R.A. Kowalski, *Logic for Problem Solving* (Elsevier (North-Holland), New York, 1979).

[10] J.-L. Lassez, M.J. Maher and D.A. Wolfram, *The Semantics of Logic Programs*, in preparation.

[11] D.W. Loveland, *Automated Theorem Proving: A Logical Basis* (Elsevier (North-Holland), New York, 1978).

[12] Z. Manna, *Mathematical Theory of Computation* (McGraw-Hill, New York, 1974).

[13] J.A. Robinson, A machine-oriented logic based on the resolution principle, *J. ACM* **23** (1965) 23–41.

[14] T. Sato, Negation and semantics of prolog programs, *Proc. 1st Internat. Logic Programming Conf.*, Marseille, France (1982) pp. 169–174.
[15] D. Scott, Lectures on a mathematical theory of computation, Tech. Monograph PRG-19, Programming Research Group, Oxford University, 1981.
[16] A. Tarski, A lattice-theoretical fixpoint theorem and its applications, *Pacific J. Math.* **5** (1955) 285–309.
[17] R.D. Tennent, *Principles of Programming Languages* (Prentice-Hall, London, 1981).
[18] D.A. Wolfram, M.J. Maher and J.-L. Lassez, A unified treatment of resolution strategies for logic programs, Tech. Rept. 83/12, Dept. of Computer Science, Univ. of Melbourne, 1983.