CONTRIBUTIONS TO THE THEORY OF LOGIC PROGRAMMING

by

Krzysztof R. Apt[1]
M.H. van Emden[2]

RESEARCH REPORT CS-80-12

University of Waterloo
Department of Computer Science
Waterloo, Ontario, Canada

February 1980

1 Vakgroep Informatica, Faculteit Economische Wetenschappen, Erasmus Universiteit, Rotterdam, Holland.

2 University of Waterloo, Department of Computer Science, Waterloo, Ontario N2L 3G1, Canada.

1. Introduction

This paper is a continuation of the investigation begun in [SPL], where various approaches to the semantics of predicate logic regarded as a programming language were discussed. Using the analogies provided by logic programming [LPS,GDM,PLPL], 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 (first described in [PLPL]). Among other results we prove by fixpoint techniques the soundness and completeness of SLD-resolution; the latter result is shown to be a consequence of the continuity of the transformation.

In programming semantics extensive use has been made of least fixpoints. The dual notion of greatest fixpoint has been used only indirectly for the characterization of program behaviour: for example, in [BGF] an induction rule is proposed which is dual to the one proposed by Scott; the latter is based on least fixpoints. The present paper shows that in the framework of Horn clause logic with SLD-resolution as computational mechanism, greatest fixpoints yield useful results in characterizing program behaviour. Admittedly, these programs are logic programs, but they have simple relationships with conventional models of programs or data bases [VCP,CDI]. 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 bounded indeterminacy. Together with our theorem (7.14) 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 schemas of bounded indeterminacy.

The paper is organized as follows. In section 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 is 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 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 behaviour of nondeterministic flowchart schemas (see [VCP]).

2. Basic Results on Fixpoints

Let L be a complete lattice with set B, order relation $\subseteq$, greatest lower bound operation $\cap$, and least upper bound operation $\cup$. 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 a 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 $h = \cup H$ because

$$h \supseteq x \text{ for any } x \in H \Rightarrow \text{(monotonicity of T)}$$
$$Th \supseteq Tx \text{ for any } x \in H \Rightarrow \text{(definition of H)}$$
$$Th \supseteq x \text{ for any } x \in H \Rightarrow \text{(definition of } \cup)$$
$$Th \supseteq \cup H \Rightarrow Th \supseteq h \Rightarrow h \in H.$$

Likewise, for monotone T, $G = \{x : x \supseteq Tx\}$, and $g = \cap G$, we have $g \in G$.

Theorem ... (2.1)

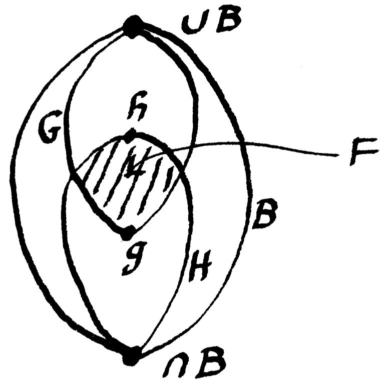
(The Knaster/Tarski fixpoint theorem)

A monotone function T has a greatest and a least fixpoint:

$$h' = Th' \text{ where } h' = \cup\{x : x = Tx\}$$
$$g' = Tg' \text{ where } g' = \cap\{x : x = Tx\}$$

Proof: We shall show that $h = Th$, where $h = \cup\{x : x \subseteq Tx\}$. 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 ∎

The diagram 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:

$T\uparrow 0 = \cap B$

$T\uparrow n = T(T\uparrow(n-1))$ if n is a successor ordinal

$\quad = \cup\{T\uparrow k : k<n\}$ if n is a limit ordinal

$T\downarrow 0 = \cup B$

$T\downarrow n = T(T\downarrow(n-1))$ if n is a successor ordinal

$\quad = \cap\{T\downarrow k : k<n\}$ if n is a limit ordinal

Theorem ... (2.2)

For any ordinal n

$T\uparrow n \subseteq lfp(T)$ and $T\downarrow n \supseteq gfp(T)$

There exists ordinals n_1 and n_2 such that

$T\uparrow n_1 = lfp(T)$ and $T\downarrow n_2 = gfp(T)$ ∎

This theorem is well known in various areas of mathematics. Within theoretical computer science it has been popularized in [IRTP].

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

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

The set $\{A_1,\ldots,A_m\}$ is the _conclusion_ of the clause; $\{B_1,\ldots,B_n\}$ is the _premiss_ 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 where $t_1,\ldots,t_k$ are terms. A _term_ is a variable or $f(t_1,\ldots,t_j)$ where f is a j-place _functor_*) and where $t_1,\ldots,t_j$ 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 $e \equiv e_1\theta \equiv \ldots \equiv e_n\theta$, 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

> for all $x_1,\ldots,x_k$, A_1 or ... or A_m if B_1 and ... and B_n
> where $x_1,\ldots,x_k$ are the variables in the clause and where
> $m>0$, $n\geq 0$.

*) or _function symbol_

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 ... and B_n. The "Horn clauses" often used in the literature are clauses which are definite or negative.

An empty clause is one where n=0 and m=0. Such a clause is to be understood as a contradiction. It is written as $\square$.

This informal semantics will be 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). 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.

Definition: ... (4.1)

Let I be an interpretation.

- A sentence is true in I iff each of its clauses is true in I.
- A clause is true in I iff each of its variable-free instances is true in I.
- 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.
- A variable-free atom F is true in I iff $F \in I$∎

An interpretation I such that a sentence S is true in I is called a model*) of S. We denote the set of models of S by M(S). S is unsatisfiable (also inconsistent) means that S has no model. 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 the

Proposition ... (4.1)

If a set of instances of clauses in a sentence S is unsatisfiable, then S is unsatisfiable∎

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

Proposition ... (4.2)

$\cap M(S)$ is the set of all variable-free atoms such that $S \models A$ ∎

Theorem ... (4.3)

$I \supseteq T(I)$ iff I is a model of P where P is a definite sentence, T is the associated transformation, and I is an interpretation.

Proof: see [SPL] ∎

Corollary ... (4.5)

For a definite sentence P we have

*) A 'Herbrand model' in more general treatments of logic. We only consider such models.

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

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

<u>Proof</u>: From section 2 we recall that for monotone T

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

Theorem (4.3) translates this directly to

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

In [SPL] this corollary is proved directly, without recourse to T ∎

<u>Theorem</u> ... (4.6)

Let T be the transformation associated with a definite sentence P. Then we have

$$\mathrm{lfp}(T) = T{\uparrow}\omega$$

<u>Proof</u>: see [SPL] ∎

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, by the functor s, and by the constants a and b. For all finite n we have

$$T^n(U) = U\backslash\{Q(a),\dots,Q(s^{n-1}(a)),P(b),\dots,P(s^{n-1}(b))\}$$

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 $T{\downarrow}\omega \neq gfp(T)$. In fact, $T^n(T{\downarrow}\omega) = T{\downarrow}\omega\backslash\{P(s^i(a)): i<n\}$, for finite n. We have $T{\downarrow}(\omega+\omega) = \{Q(s^n(b)): n<\omega\} = gfp(T) = lfp(T)$.

However, definite sentences representing flowgraphs of bounded non-determinacy lead to a T such that $T{\downarrow}\omega$ is the greatest fixpoint (see section 9). A slightly different condition is given in the following theorem, which has a proof very similar to the one of lemma 9.5.

Theorem ... (4.7)

If each SLD-tree with root $\leftarrow A$, where A is variable-free, is of bounded degree and there are finitely many variable-free terms in the Herbrand universe, then $T{\downarrow}\omega = gfp(T)$ ▮

So, for example, if P is finite and there are no function symbols in P then $T_P{\downarrow}\omega = gfp(T_P)$. (SLD trees are introduced in section 6.)

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 [MOL,LFF] and subsequently by many others [see MTP,ATP]. For our purpose the SL-resolution procedure [LRS] is most important and especially a variant [SPL,PLPL,LPS] intended for use with sentences containing, apart from one negative clause, only definite clauses. Because of this restriction we will refer to this resolution refutation procedure as SLD-resolution: SL-resolution for Definite 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 $N_0, N_1, N_2, \ldots$ of negative clauses, a sequence $d_1, d_2, \ldots$ of clauses in P (the input clauses of the derivation), and a sequence $\theta_1, \theta_2, \ldots$ of substitutions. 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 $N_i \equiv \leftarrow A_1, \ldots, A_k, \ldots, A_m, \quad m \geq 1$

with A_k as selected atom.

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

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 (5.4)) is a form of completeness.

Let [A] denote the set of all variable-free instances of an atom A.

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, and that $\theta_1,\ldots,\theta_n$ is the sequence of substitutions of the refutation. We assert that for every atom A in $G\theta_1\ldots\theta_n$, $[A] \subseteq T^n(\phi)$.

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

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

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

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

Case m=0: $[A] \subseteq T(\phi)$, by definition of T. Also: $[X\theta_{n-i} \dots \theta_n] \subseteq [X\theta_{n-i}] \subseteq [A]$; also $T(\phi) \subseteq T^{i+1}(\phi)$. Hence $[X\theta_{n-i} \dots \theta_n] \subseteq T^{i+1}(\phi)$.

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

Corollary ... (5.2)

(Soundness of SLD-refutations)

Let P be a definite sentence and let N be a negative clause such that there exists an SLD refutation of $P \cup \{N\}$. Then $P \cup \{N\}$ is unsatisfiable.

Proof: By theorem (5.1) there exists a substitution θ such that for all atoms A in N, $A\theta \in \text{lfp}(T) = \bigcap M(P)$. 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 unsatisfiable■

Corollary ... (5.25)

The success set of a definite sentence is contained in its least model■

Corollaries (5.2) and (5.25) 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 = {app(nil,y,y)

,app(u•x,y,u•z) $\leftarrow$ app(x,y,z)

}

The functor "•" is used as infix operator: for example u•x stands for •(u,x). This term 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 between three lists, where the third list is the result of appending the second list to the end of the first.

Let A = app(x_1,3•y_1,2•3•4•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 ... and z_1 such that 2•3•4•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: $A\theta_1 \ldots \theta_n$ is app(2•nil,3•4•w,2•3•4•w), thereby stating that the x_1,y_1, and z_1 that must exist by the soundness of resolution, are 2•nil, 4•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 ... (5.3)

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 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 $B_i \equiv B_i\theta$ for $i=0,\ldots,n$ 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 refuation of $P \cup \{N\}$ with A_i as the first selected atom, $\theta\zeta$ as the first substitution and the same input clause ∎

Lemma ... (5.4)

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

Proof: 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), $\cap M(P) = \mathrm{lfp}(T)$. By Theorem (4.6) $A \in \mathrm{lfp}(T) \Rightarrow A \in T^k(\phi)$ for some finite k. We now prove by induction k that $A \in T^k(\phi)$ implies that an SLD-refutation exists of $P \cup \{\leftarrow A\}$.

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

If $A \in T^{k+1}(\phi)$, then by 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 $\{B_1\theta,\ldots,B_m\theta\} \subseteq T^k(\phi)$, 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 $P \cup \{\leftarrow A\}$ ∎

<u>Theorem</u> ... (5.5)

(Completeness of SLD refutations)

Let P be a definite sentence and let N be a negative clause such that $P \cup \{N\}$ is unsatisfiable. For each atom A_k of N there exists an SLD refutation of $P \cup \{N\}$ with A_k as first selected atom.

<u>Proof</u>: Suppose $N \equiv \leftarrow A_1, \ldots, A_n$. If $P \cup \{N\}$ is unsatisfiable, 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 (5.4) 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 (5.3) there exists an SLD-refutation of $P \cup \{N\}$, whatever the first selected atom ∎

<u>Corollary</u> ...(5.6)

The least model of a definite sentence P is the success 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 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), (b), or (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 we prove that alternatives (a) and (c) need not be considered.

In the first place, it is easy to see that (from Lemma 5.3) if a refutation of $P \cup \{N\}$ exists, then there also exists one where 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 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 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 <u>SLD-tree</u>, and define it as follows. Let P be a definite sentence and let N be a negative or empty clause. An SLD-tree for P ∪ {N} has N as root. All its nodes are negative or empty clauses. A non-empty node has one atom which is the selected atom. A node

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

with selected atom A_k has a descendant for every clause

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

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

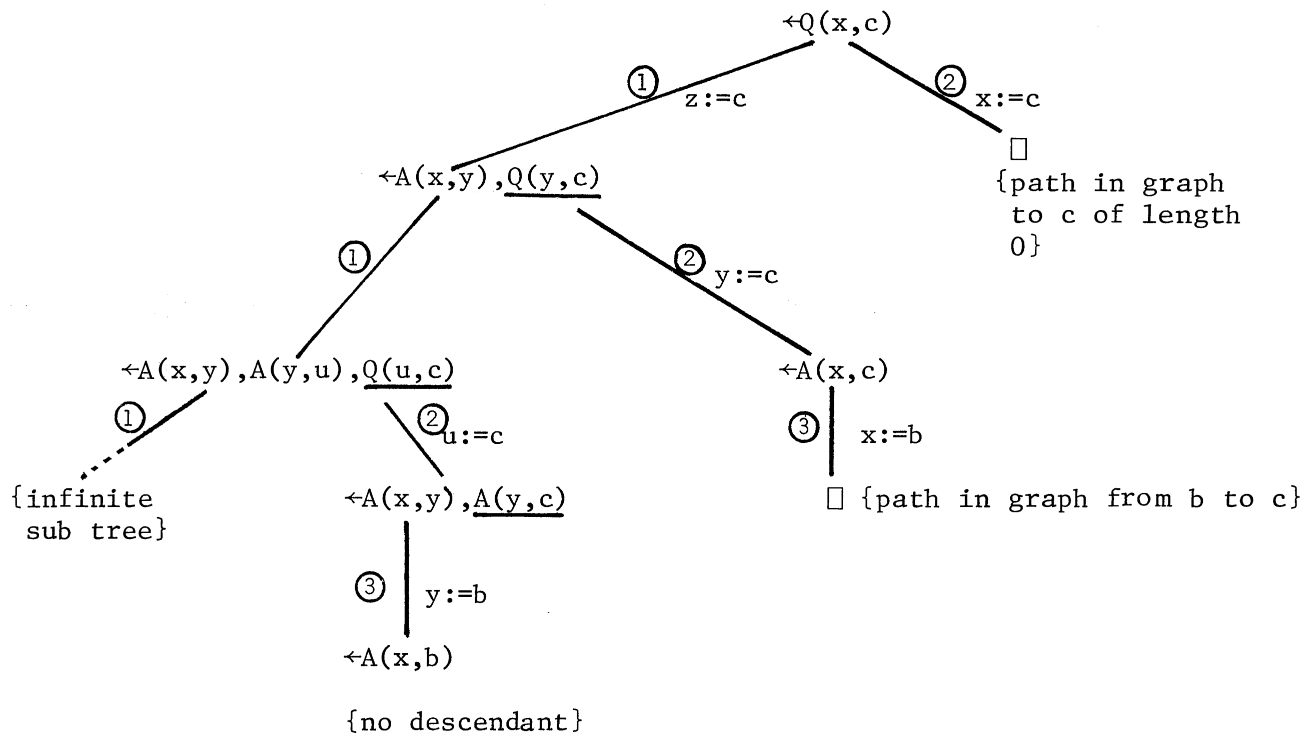
$$\leftarrow (A_1 , \ldots , A_{k-1} , B_1 , \ldots , B_q , A_{k+1} , \ldots , A_m)\theta .$$

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 ∪ {N}, there exists an SLD-tree for P ∪ {N} of which a path is the most general version of this refutation. In general, a given P ∪ {N} has different SLD-trees depending on which atoms are the selected atoms. Often there are very many SLD-trees, of vastly differing size.

<u>Example</u> (6.1)

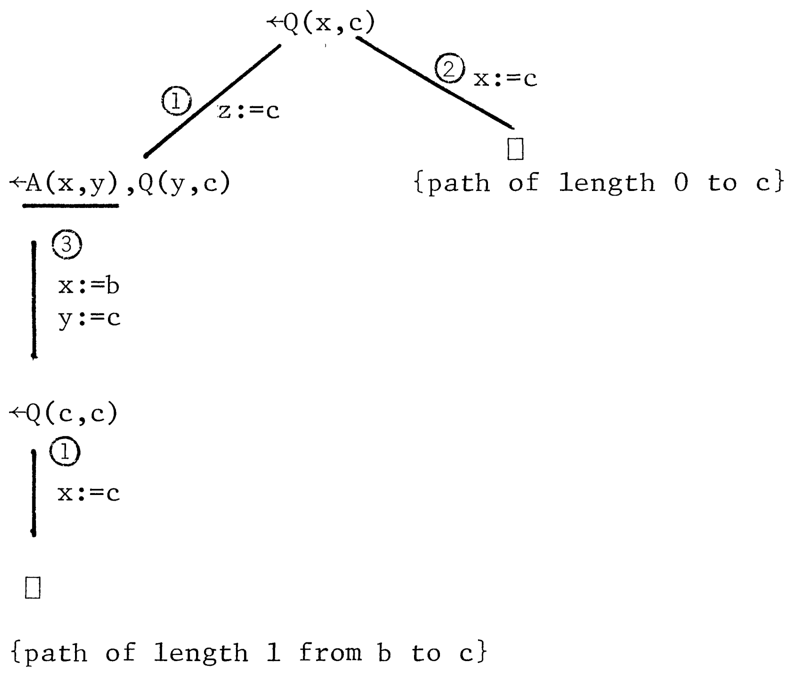
$$P = \{\overset{\textcircled{1}}{(}Q(x,z) \leftarrow A(x,y), Q(y,z)), \; \overset{\textcircled{2}}{Q}(x,x), \; \overset{\textcircled{3}}{A}(b,c)\}$$

In P x,y, and z are variables; 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 ←Q(x,c) negates that a path to c exists. One SLD-tree for P ∪ {←Q(x,c)} is as follows:



The selected atoms are underlined.

Another SLD-tree for P ∪ {←Q(x,c)} is finite:



<u>Lemma</u> ... (6.2)

If a refutation exists of P ∪ {$\leftarrow A_1,\ldots,A_k$} with A_i as first selected atom, then a refutation exists also with A_j as first selected atom, for i,j=1,...,k.

<u>Proof</u>: If a refutation exists, then by theorem (5.2), P ∪ {$\leftarrow A_1,\ldots,A_n$} is unsatisfiable. By theorem (5.5) a refutation of P ∪ {$\leftarrow A_1,\ldots,A_k$} exists, with A_j as selected atom ∎

Theorem ... (6.3)

("Strong completeness of SLD-resolution")

If $P \cup \{N\}$ is unsatisfiable, then any SLD-tree for $P \cup \{N\}$ contains the empty clause, where P is a definite sentence and N a negative clause.

Proof: If $P \cup \{N\}$ is unsatisfiable, then a refutation exists, according to weak completeness, say with A_i as first selected atom. We are now going to prove by induction on the length of the refutation that the existence of a refutation of $P \cup \{N\}$ implies that any SLD-tree for $P \cup \{N\}$ contains the empty clause.

This is obviously true for refutations of length 1. For the induction step, assume it is true for refutations of length n-1. Let $P \cup \{N\}$ have a refutation R of length n, with first selected atom A_i*). By lemma (6.2) there exists a refutation R' with A_j as first selected atom. The second clause C of R' is the root of one of the subtrees S' of S. There exists a refutation of $P \cup \{C\}$ of length n-1. By the induction hypothesis, S' contains $\square$, hence also S ∎

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.

*) Let S be an arbitrary SLD-tree with first selected atom A_j.

The choice of the selected atom can make an enormous difference in the size of the SLD-tree, as example (6.1) showed. 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.

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 fix-point 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 $T{\downarrow}\omega \supseteq$ 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 gfp(T). Because of this result we are especially interested in classes of definite sentences for which $T{\downarrow}\omega$ = gfp(T) also holds. After this section we give an intuitive and semantic interpretation of 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 $k>1$. 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:

<u>Lemma</u> ... (7.2)

Let λ by 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 ∎

<u>Lemma</u> ... (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 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 <u>correctness</u> of the method. We prepare the <u>completeness</u> proof by introducing some definitions and lemmas.

Definition ... (7.4)

A negative clause N is called infinite (with respect to a definite sentence P) iff every SLD-tree for P with N as root is infinite and has bounded degree ■

Lemma ... (7.5)

For no atom A in an infinite negative clause N can ←A be the root of a finitely failed SLD-tree.

Proof: If a finitely failed SLD-tree F with A as root would exist, then a finitely failed SLD-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 an infinite negative clause. In every SLD-tree with N as root, there is a direct descendent of N which is infinite ■

An obvious proof by contradiction exists.

Definition ... (7.7)

Let M and N be negative clauses. N contains the atom A, and θ is a substitution. We write N $\overset{\theta;A}{\Longrightarrow}$ 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 $\leftarrow A_1,\ldots,A_{i-1},A,A_{i+1},\ldots,A_n$ ∎

<u>Lemma</u> ... (7.8)

Suppose that $\leftarrow A_1,\ldots,A_n$ is infinite and that $\leftarrow A_i$ is not infinite. Then for some substitution θ, $\leftarrow A_1,\ldots,A_n \overset{\theta;A_i}{\Longrightarrow} \leftarrow(A_1,\ldots,A_{i-1},A_{i+1},\ldots,A_n)\theta$, $[A_i\theta] \subseteq T{\downarrow}\omega$, and $\leftarrow(A_1,\ldots,A_{i-1},A_{i+1},\ldots,A_n)\theta$ is infinite.

<u>Proof</u>: 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 "$\Longrightarrow$" the first part of the lemma is proved. We now show that, if $\leftarrow(A_1,\ldots,A_{i-1},A_{i+1},\ldots,A_n)\theta$ would not be 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 would be infinite, then a finite SLD-tree, with $\leftarrow A_1,\ldots,A_n$ as root, could be constructed∎

Lemma ... (7.9)

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

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

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

$$\leftarrow(A_1,\ldots,A_{i-1},B_1,\ldots,B_m,A_{i+1},\ldots,A_n)\theta \qquad \ldots (7.10)$$

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 I: $\leftarrow B_1\theta$ is infinite.

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

$$(7.10) \xRightarrow{\theta_1;B_1\theta} N_1, \qquad \ldots (7.11)$$

where, by the definition of "$\Longrightarrow$", N_1 is of the form

$$\leftarrow A_1\theta\theta_1,\ldots,A_{i-1}\theta\theta_1,M_1,B_2\theta\theta_1,\ldots,B_m\theta\theta_1,A_{i+1}\theta\theta_1,\ldots,A_n\theta\theta_1. \qquad \ldots (7.12)$$

Case II: $\leftarrow B_1\theta$ is not infinite.

By lemma (7.8) there exists a substitution θ_1 such that $[B_1\theta\theta_1] \subseteq T{\downarrow}\omega \subseteq T^{k-1}(U)$,

(7.10) $\xRightarrow{\theta_1;B_1\theta} N_1$, 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 $[B_1\theta\theta_1] \subseteq T^{k-1}(U)$, (7.11) holds, and N_1 is of the form (7.12). We proceed now in the same way with $B_j\theta\theta_1\ldots\theta_{j-1}$, for $j=2,\ldots,m$, successively. Each time we obtain a substitution θ_j and an infinite clause N_j such that

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

By the definition of "$\Longrightarrow$" we have

$$N \xRightarrow{\theta\theta_1\ldots\theta_m\ ;\ A_i} N_m.$$

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

Corollary ... (7.13)

If A is a variable-free atom such that $\leftarrow A$ is infinite, then $A \in T{\downarrow}\omega$ ∎

By Corollary (5.6) 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 ... (7.14)

The finite-failure set of a definite sentence 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 $T{\downarrow}\omega$. $\leftarrow A$ is not infinite. If $\leftarrow A$ were the root of an SLD-tree containing the empty clause, then, by corollary (5.6), $A \in \mathrm{lfp}(T) \subseteq T{\downarrow}\omega$. 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,

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

says what some of the elements are. But it does not express the fact that these are the <u>only</u> elements.

If we want to establish that Air is an Element, then we can use SLD-resolution to refute $E \cup \{\leftarrow Element(Air)\}$. 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

$$Stuff(Mud) \wedge$$
$$\forall x.Element(x) \leftarrow x = Air \vee x = Fire \vee x = Water \vee x = Earth \quad \ldots (8.1)$$

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 (8.1). This use of finite failure is due to Clark [NAF], 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 schemas for equality. In this section we give a justification on the basis of theorem (7.14).

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 ... (8.5)

Change each clause

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

of P to the universally quantified formula

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

where $x_1,\ldots,x_n$ are different from the clause's variables and where $y_1,\ldots,y_k$ 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 at (8.5)).

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 ... (8.3)

Change each set $\{\forall x_{11},\ldots,x_{1m}.P\leftarrow Q_1\ ,\ldots,\forall x_{n1},\ldots,x_{nm_n}(P\leftarrow Q]\}$ of implications, obtained by the above rule and having the same predicate symbol in the conclusion, to $\forall x_1,\ldots,x_m.P\leftarrow(Q_1 \vee\ldots\vee Q_n)$*) 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, and a disjunction of existentially quantified conjunctions as premiss ∎

We now apply the following

Rule ... (8.4)

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

*) where x_i (i=1, ..., m) are all variables mentioned in the above formulas

The last rule is the following one.

<u>Rule</u> ... (8.45)

For any (say, n-place) predicate symbol Q which occurs in a premiss 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 ∎

<u>Example</u>

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

is the IF-definition associated with E ∎

<u>Example</u>

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

is the IF-definition associated with

$\{\mathrm{App}(\mathrm{nil},y,y)$

$,\mathrm{App}(u{\cdot}x,y,u{\cdot}z) \leftarrow \mathrm{App}(x,y,z)$

$\}$ ∎

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 non-clausal first-order fomulas is true in an interpretation I. As before, interpretations are subsets of a certain Herbrand base.

Definition ... (8.5)

- A universally quantified implication is true in I iff for each variable-free instantiation of the implication which makes the premiss true in I, the conclusion is true in I also
- 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
- A variable-free conjunction is true in I iff each conjunct is true in I
- A variable-free disjunction is true in I iff at least one disjunct is true in I
- 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 ... (8.6)

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. $R(t_1,\dots,t_n) \in T_P(I)$ iff the substitution $(t_1/x_1,\dots,t_n/x_n) = \lambda$ makes the implication's premiss true in I, where $x_1,\dots,x_n$ are the free variables of the universally quantified implication.

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

Lemma ... (8.7)

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 $\forall x_1,\dots,x_n.$ $Q(x_1,\dots,x_n) \leftarrow Q(x_1,\dots,x_n)$ which has no counterpart in P (see rule (8.45)), also has the same set of models as P. Hence theorem (4.3) 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 that $R(t_1,\dots,t_n)$ is true in I, with R the predicate symbol in a left-hand side and $t_1,\dots,t_n$ are variable-free. By the assumption $I \subseteq T_P(I)$, $R(t_1,\dots,t_n)$ is true in $T_P(I)$. By lemma (8.6), 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 $I \subseteq T_P(I)$.

$R(t_1,\dots,t_n) \in I \Rightarrow$

because of rule 5 there exists an equivalence having $R(x_1,\dots,x_n)$ in its conclusion and the substitution $(t_1/x_1,\dots,t_n/x_n)$ makes its premiss true in I $\Rightarrow$ (lemma 8.6) $R(t_1,\dots,t_n) \in T_P(I)$.

<u>Theorem</u> ... (8.8)

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.

<u>Proof</u>: Suppose A is the root of a finitely failed SLD-tree. By theorem (7.14), $A \notin T_P{\downarrow}\omega$. Also, we have $T_P{\downarrow}\omega \supseteq gfp(T_P)$. Hence $A \notin gfp(T_P)$; hence by Theorem (2.1) $A \notin I$ for any I such that $I = T(I)$; hence, by lemma 8.7, A is false in all models of the IFF-definition

<u>Theorem</u> ... (8.9)

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.

<u>Proof</u>: The argument of the proof of theorem (8.8) can be reversed provided that $T_P{\downarrow}\omega = gfp(T_P)$ ∎

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", various aspects of which are discussed in [CWDB,CDI,NAF]. The second application is to the semantic characterization of the behaviour of nondeterministic programs. This application has not, as far as we know, been published.

At this point 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 [VCP].

A flowgraph is a possibly infinite directed graph where the arcs are labelled 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},x_{i-1}) and $N_i,x_i)$ iff there is an arc from N_{i-1} to N_i and $(x_{i-1},x_i) \in C_i$, the command labelling that arc. A computation is a possibly infinite sequence of (node,state) pairs such that every element (N_j,x_j) has a successor (N_{j+1},x_{j+1}) in the sequence if (N_j,x_j) is in the transition relation with some (node,state)-pair and (N_{j+1},x_{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 a finite computation and 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,x) there may be several such pairs (N',x') such that (N,x) and (N',x') are in the successor relation. It is this feature that makes the epithet "nondeterministic" applicable to flowgraphs. A flowgraph is said to be of <u>bounded nondeterminacy</u> if the number of such successors is bounded.

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 labelled 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,..., if (N_i,s_i), (N_{i+1},s_{i+1}) are the i-th 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.

<u>Lemma</u> ... (9.1)

The sequence of clauses associated with a computation is a derivation in which always the leftmost atom is 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 behaviours of flowgraphs. Note that computations form a tree, just as derivations form an SLD-tree.

An <u>interpreter</u> for a flowgraph is a procedure to be used with the purpose of constructing a successful computation. 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.

In the case of a 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.

For the behaviour of such an interpreter when starting 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

(C) for any choice of successors, the interpreter terminates and no successful computation is found.

Let P be the clausal sentence associated with the flowgraph. In [VCP] one may find an equivalent of the following

<u>Theorem</u> ... (9.2)

We have contingency A iff $P \models S(a,b)$.

<u>Proof</u>: (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 $\leftarrow S(a,x),\ldots,H(b,x)$, $\Box$ exists, in which b is substituted for x $\Rightarrow S(a,b) \in \mathrm{lfp}(T_P)$, by theorem 5.1 $\Rightarrow P \models S(a,b)$.

(if)

$P \models S(a,b) \Rightarrow P \cup \{\leftarrow S(a,b)\}$ unsatisfiable $\Rightarrow$ (lemma 5.4)) refutation $\leftarrow S(a,b),\ldots,\square$ exists $\Rightarrow$ because $H(x,x)$ is the only clause without a premiss, refutation $\leftarrow S(a,b),\ldots, H(t,b), \square$ exists with t some term; by the strong completeness (theorem (6.3)) such a refutation exists where the left-most atom is the selected atom $\Rightarrow$ by the form of P, t must equal b in the above refutation $\Rightarrow$ (lemma (5.3)) $\leftarrow S(a,x),\ldots,\leftarrow H(b,x), \square$ is a refutation $\Rightarrow$ (lemma (9.1)) $(S,a),\ldots,(H,b)$ is a successful computation ■

<u>Theorem</u> (9.3)

Assuming that the nondeterminacy of a finite flowgraph is bounded, we have contingency C iff $P' \models \neg S(a,\sigma)$ for all states σ, where P' is the IFF-definition associated with P.

<u>Proof</u>: (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 (8.8)) for all states σ, $P' \models \neg S(a,\sigma)$.

(if)

For all states σ, $P' \models \neg S(a,\sigma) \Rightarrow$ (Lemma (8.7) and Theorem (2.1))for all states σ, $S(a,\sigma) \notin gfp(T_P)$

$\Rightarrow$ (Lemma (9.5)) for all states σ, $S(a,\sigma) \notin T_P\downarrow\omega$

$\Rightarrow$ (Corollary (7.13)) 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 ■

<u>Corollary</u> ... (9.4)

Let P be the definite sentence representing a finite flowgraph of bounded nondeterminacy. We have contingency B iff

$S(a,\sigma) \in gfp(T_P)$ for at least one state σ and

$S(a,\sigma) \in lfp(T_P)$ for no state σ ∎

<u>Lemma</u> ... (9.5)

If P represents a finite flowgraph of bounded nondeterminacy, then $T_P{\downarrow}w = gfp(T_P)$.

<u>Proof</u>: As we already have $T_P{\downarrow}\omega \supseteq gfp(T_P)$, 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 only finitely many clauses resolve with $\leftarrow Q(a,c)$

$\Rightarrow$ there exists a σ and an N such that $C(a,\sigma)$ and $R(\sigma,c)$ both in $T^{n-1}(U)$ for infinitely many n; this because, be bounded 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 ∎

10. Acknowledgements

The initial impetus of this work was given by discussions with Robert Kowalski. Further discussions with Hajnal Andreka, Keith Clark, and Istvan Nemeti have provided valuable insights. The National Science and Engineering Research Council has contributed supporting facilities.

11. Literature References

[ATP] D.W. Loveland: Automated Theorem Proving; North Holland, Amsterdam, etc., 1978.

[BGF] W.P. de Roever: On backtracking and greatest fixpoints; in: Formal Description of Programming Concepts, E. Neuhold (ed.), North Holland, Amsterdam, etc., 1978.

[CDI] M.H. van Emden: Computation and deductive information retrieval; in: Formal Description of Programming Concepts, E. Neuhold (ed.), North Holland, Amsterdam, etc., 1978.

[CWDB] R. Reiter: On closed-world data bases; in: [LDB].

[GDM] A. Colmerauer: Grammaires de mētamorphose; "Natural Language Communication with Computers", L. Bolc (ed.), Springer Lecture Notes in Computer Science, 1978.

[IRTP] P. Hitchcock and D. Park: Induction rules and termination proofs; in: Automata, Languages, and Programming, M. Nivat (ed.), North Holland, Amsterdam, etc., 1973.

[LDB] H. Gallaire and J. Minker (eds.): Logic and Data Bases; Plenum, New York, 1978.

[LFF] J.A. Robinson: Logic: Function and Form; Edinburgh University Press, 1979.

[LPS] R.A. Kowalski: Logic for Problem-Solving; North-Holland-Elsevier, New York, 1979.

[LUSH] Robert Hill: LUSH-resolution and its completeness; DCL Memo 78, Department of Computational Logic, University of Edinburgh, 1974.

[MOL] J.A. Robinson: A machine-oriented logic based on the resolution principle; J.ACM 12 (1965), 23-44.

[MTP] C.L. Chang and R.C.T. Lee: Symbolic Logic and Mechanical Theorem-Proving; New York, Academic Press, 1973.

[NAF] K.L. Clark: Negation as Failure, see [LDB].

[PLPL] R.A. Kowalski: Predicate logic as a programming language; Proc. IFIP 1974, 556-574.

[SPL] M.H. van Emden and R.A. Kowalski: The semantics of predicate logic as a programming language; J.ACM 23 (1976), 733-742.

[VCP] M.H. van Emden: Verification conditions as programs; in: Automata, Languages, and Programming, S. Michaelson and R. Milner (eds.), Edinburgh University Press, 1976.

1. Introduction

This paper is a continuation of the investigation begun in [SPL], where various approaches to the semantics of predicate logic regarded as a programming language were discussed. Using the analogies provided by logic programming [LPS,GDM,PLPL], 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 (first described in [PLPL]). Among other results we prove by fixpoint techniques the soundness and completeness of SLD-resolution; the latter result is shown to be a consequence of the continuity of the transformation.

In programming semantics extensive use has been made of least fixpoints. The dual notion of greatest fixpoint has been used only indirectly **for the characterization of program behaviour: for example, in [BGF] an** induction rule is proposed which is dual to the one proposed by Scott; the latter is based on least fixpoints. The present paper shows that in the framework of Horn clause logic with SLD-resolution as computational mechanism, greatest fixpoints yield useful results in characterizing program behaviour. Admittedly, these programs are logic programs, but they have simple relationships with conventional models of programs or data bases [VCP,CDI]. 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 bounded indeterminacy. Together with our theorem (7.14) 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 schemas of bounded indeterminacy.

The paper is organized as follows. In section 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 is 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 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 behaviour of nondeterministic flowchart schemas (see [VCP]).

The last rule is the following one.

Rule ... (8.45)

For any (say, n-place) predicate symbol Q which occurs in a premiss 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 ∎

Example

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

is the IF-definition associated with E ∎

Example

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

is the IF-definition associated with

{App(nil,y,y)

,$\mathrm{App}(u\cdot x,y,u\cdot z) \leftarrow \mathrm{App}(x,y,z)$

} ∎

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 non-clausal first-order fomulas is true in an interpretation I. As before, interpretations are subsets of a certain Herbrand base.

Definition ... (8.5)

- A universally quantified implication is true in I iff for each variable-free instantiation of the implication which makes the premiss true in I, the conclusion is true in I also
- 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
- A variable-free conjunction is true in I iff each conjunct is true in I
- A variable-free disjunction is true in I iff at least one disjunct is true in I
- 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 ... (8.6)

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 premiss 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 premiss of the implication true in I ∎

Lemma ... (8.7)

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 (8.45)), also has the same set of models as P. Hence theorem (4.3) 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 that $R(t_1,\ldots,t_n)$ is true in I, with R the predicate symbol in a left-hand side and $t_1,\ldots,t_n$ are variable-free. By the assumption $I \subseteq T_P(I)$, $R(t_1,\ldots,t_n)$ is true in $T_P(I)$. By lemma (8.6), 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 rule 5 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 premiss true in $I \Rightarrow$ (lemma 8.6) $R(t_1,\ldots,t_n) \in T_P(I)$.

Theorem ... (8.8)

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 (7.14), $A \notin T_P{\downarrow}\omega$. Also, we have $T_P{\downarrow}\omega \supseteq gfp(T_P)$. Hence $A \notin gfp(T_P)$; hence by Theorem (2.1) $A \notin I$ for any I such that $I = T(I)$; hence, by lemma 8.7, A is false in all models of the IFF-definition

Theorem ... (8.9)

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 (8.8) can be reversed provided that $T_P{\downarrow}\omega = gfp(T_P)$ ∎

(if)

$P \models S(a,b) \Rightarrow P \cup \{\leftarrow S(a,b)\}$ unsatisfiable $\Rightarrow$ (lemma 5.4)) refutation $\leftarrow S(a,b),\ldots,\square$ exists $\Rightarrow$ because $H(x,x)$ is the only clause without a premiss, refutation $\leftarrow S(a,b),\ldots, H(t,b), \square$ exists with t some term; by the strong completeness (theorem (6.3)) such a refutation exists where the left-most atom is the selected atom $\Rightarrow$ by the form of P, t must equal b in the above refutation $\Rightarrow$ (lemma (5.3)) $\leftarrow S(a,x),\ldots,\leftarrow H(b,x), \square$ is a refutation $\Rightarrow$ (lemma (9.1)) $(S,a),\ldots,(H,b)$ is a successful computation ■

Theorem (9.3)

Assuming that the nondeterminacy of a finite flowgraph is bounded, 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 (8.8)) for all states σ, $P' \models \neg S(a,\sigma)$.

(if)

For all states σ, $P' \models \neg S(a,\sigma)$ $\Rightarrow$ (Lemma (8.7) and Theorem (2.1)) for all states σ, $S(a,\sigma) \notin gfp(T_P)$

$\Rightarrow$ (Lemma (9.5)) for all states σ, $S(a,\sigma) \notin T_P\downarrow\omega$

$\Rightarrow$ (Corollary (7.13)) 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 ... (9.4)

Let P be the definite sentence representing a finite flowgraph of bounded nondeterminacy. We have contingency B iff

$S(a,\sigma) \in gfp(T_P)$ for at least one state σ and

$S(a,\sigma) \in lfp(T_P)$ for no state σ ∎

Lemma ... (9.5)

If P represents a finite flowgraph of bounded nondeterminacy, then $T_P{\downarrow}w = gfp(T_P)$.

Proof: As we already have $T_P{\downarrow}\omega \supseteq gfp(T_P)$, 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 $\mathbb{Q}(x,z) \leftarrow C(x,y), R(y,z)$ in P such that for infinitely many n, $\mathbb{C}(a,\sigma_n)$ and $R(\sigma_n,c)$ both in $T^{n-1}(U)$; this is because only finitely many clauses resolve with $\leftarrow Q(a,c)$

$\Rightarrow$ there exists a σ and an N such that $C(a,\sigma)$ and $R(\sigma,c)$ both in $T^{n-1}(U)$ for infinitely many n; this because, be bounded nondeterminacy, only finitely many σ_n existssuch 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 ∎

11. Literature References

[ATP] D.W. Loveland: Automated Theorem Proving; North Holland, Amsterdam, etc., 1978.

[BGF] W.P. de Roever: On backtracking and greatest fixpoints; in: Formal Description of Programming Concepts, E. Neuhold (ed.), North Holland, Amsterdam, etc., 1978.

[CDI] M.H. van Emden: Computation and deductive information retrieval; in: Formal Description of Programming Concepts, E. Neuhold (ed.), North Holland, Amsterdam, etc., 1978.

[CWDB] R. Reiter: On closed-world data bases; in: [LDB].

[GDM] A. Colmerauer: Grammaires de métamorphose; "Natural Language Communication with Computers", L. Bolc (ed.), Springer Lecture Notes in Computer Science, 1978.

[IRTP] P. Hitchcock and D. Park: Induction rules and termination proofs; in: Automata, Languages, and Programming, M. Nivat (ed.), North Holland, Amsterdam, etc., 1973.

[LDB] H. Gallaire and J. Minker (eds.): Logic and Data Bases; Plenum, New York, 1978.

[LFF] J.A. Robinson: Logic: Function and Form; Edinburgh University Press, 1979.

[LPS] R.A. Kowalski: Logic for Problem-Solving; North-Holland-Elsevier, New York, 1979.

[LUSH] Robert Hill: LUSH-resolution and its completeness; DCL Memo 78, Department of Computational Logic, University of Edinburgh, 1974.

[MOL] J.A. Robinson: A machine-oriented logic based on the resolution principle; J.ACM 12 (1965), 23-44.

[MTP] C.L. Chang and R.C.T. Lee: Symbolic Logic and Mechanical Theorem-Proving; New York, Academic Press, 1973.

[NAF] K.L. Clark: Negation as Failure, see [LDB].

[PLPL] R.A. Kowalski: Predicate logic as a programming language; Proc. IFIP 1974, 556-574.

[SPL] M.H. van Emden and R.A. Kowalski: The semantics of predicate logic as a programming language; J.ACM 23 (1976), 733-742.

[VCP] M.H. van Emden: Verification conditions as programs; in: Automata, Languages, and Programming, S. Michaelson and R. Milner (eds.), Edinburgh University Press, 1976.