New Generation Computing, 9 (1991) 335-363
OHMSHA, LTD. and Springer-Verlag

Acyclic Programs*

Krzysztof R. APT
*Centre for Mathematics and Computer Science,
Kruislaan 413, 1098 sj Amsterdam, The Netherlands.*
Marc BEZEM
*Department of Philosophy,
State University of Utrecht,
Heidelberglaan 8, 3584 CS Utrecht, The Netherlands.*

Received 15 October 1990
Revised manuscript received 15 April 1991

Abstract We study here a natural subclass of the locally stratified programs which we call acyclic. Acyclic programs enjoy several natural properties. First, they terminate for a large and natural class of general goals, so they could be used as terminating PROLOG programs. Next, their semantics can be defined in several equivalent ways. In particular we show that the immediate consequence operator of an acyclic program P has a unique fixpoint M_P, which coincides with the perfect model of P, is the unique Herbrand model of the completion of P and can be identified with the unique fixpoint of the 3-valued immediate consequence operator associated with P. The completion of an acylic program P is shown to satisfy an even stronger property: addition of a domain closure axiom results in a theory which is complete and decidable with respect to a large class of formulas including the variable-free ones. This implies that M_P is recursive.

On the procedural side we show that SLS-resolution and SLDNF-resolution for acyclic programs coincide, are effective, sound and (non-floundering) complete with respect to the declarative semantics.

Finally, we show that various forms of temporal reasoning, as exemplified by the so-called Yale Shooting Problem, can be naturally described by means of acyclic programs.

Keywords: Logic Programming, Negation, Semantics, Non-Monotonic Reasoning.

* This paper is a revised version of the paper presented at the Seventh International Conference on Logic Programming, 1990, Jerusalem.

§1 Introduction

1.1 Motivation

This paper is about a simple, yet remarkable class of general logic programs. We call them *acyclic* because, given an acyclic program, for a large class of general goals, including the variable-free ones, no infinite SLDNF-derivations exist.

The class of acyclic programs includes the recursion-free general programs and is included in the class of locally stratified programs defined by Przymusinski.[25] It was originally introduced in Cavedon[7] under a rather unattractive name of *ω-locally hierachical programs*. Intuitively, a program is acyclic if a mapping from variable-free literals to natural numbers can be exhibited showing that no recursion on the variable-free level exists.

In this paper we systematically study various approaches to the semantics and proof theory of acyclic programs. Among the main new results we mention the equality between the unique fixpoints of the immediate consequence operator T_P and its 3-valued counterpart Φ_P, decidability of Clark's completion of an acyclic program augmented by the domain closure axiom with respect to a large class of formulas, and identification by means of the concept of boundedness of a large class of goals for which the acyclic programs terminate w. r. t. SLDNF- and SLS-resolution.

These results combined with those previously established by Przymusinski[25] and Cavedon[7] show that several ways of defining the semantics of general logic programs including Clark's completion, perfect model semantics, fixpoint semantics based on the immediate consequence operator T_P and its 3-valued counterpart Φ_P, and two forms of resolution—SLDNF and SLS—coincide in the case of acyclic programs. Thus the class of acyclic programs can be viewed as a common denominator of various approaches to the proof theory and semantics of general logic programs, approaches which in general yield different results.

This striking uniformity can lead the reader to wonder whether acyclic programs are sufficiently strong for modeling non-monotonic reasoning and for computing in general. It has been argued (see e.g. Przymusinski[26]) that Clark's completion, *comp*(P), is in general too weak to model satisfactorily non-monotonic reasoning.

To ward off such a criticism we show that a large class of problems in temporal reasoning, as exemplified by the so-called Yale Shooting Problem of Hanks and McDermott,[14] can be naturally formalized using acyclic programs.

In Bezem[4] it was shown that even without the use of negation every total recursive function can be computed by an acyclic program. Moreover, the guaranteed termination of SLDNF-derivations for a large class of general goals shows that acyclic programs could be used as terminating PROLOG programs.

Thus, after all, acyclic programs form a powerful class.

However, not all things are so rosy. It can be shown that the property of being an acyclic program is highly undecidable—it is Π_2^0 complete in the arithmetical hierachy. In some cases, including the Yale Shooting Problem, we can easily prove that a program is acyclic by exhibiting a simple 'termination function' defined in terms of the arguments of the relations used.

1.2 Plan of the Paper

The paper is organized as follows. In the next subsection we define acyclic programs and introduce the important concept of a bounded general goal. Bounded general goals include the variable-free ones.

In Section 2 we study the declarative semantics of acyclic programs. We show that for every acyclic program P its immediate consequence operator T_P has a unique fixpoint. By the results of Przymusinski[25)] and Apt, Blair and Walker[3)] this fixpoint is the unique perfect Herbrand model of P and the unique Herbrand model of Clark's completion, *comp*(P). Moreover, we show that this fixpoint can be identified with the unique fixpoint of the Φ_P operator due to Fitting[12)], defined on the 3-valued Herbrand interpretations of P.

In Section 3 we study Clark's completion of acyclic programs. We prove that for an acyclic program P, *comp*(P) augmented by a domain closure axiom *DCA*, is a complete and decidable theory for bounded general goals. This implies that the unique perfect Herbrand model of P is recursive.

Then we turn to the procedural semantics of acyclic programs. In Section 4 we show that SLDNF-derivations for a bounded general goal and an acyclic program always terminate. Moreover we show that for acyclic programs SLS-resolution and SLDNF-resolution coincide and are effective. The results of Cavedon[7)] and Przymusinski[27)] imply (non-floundering) completeness of these two resolution methods.

Finally, in Section 5, we show how a well known problem in temporal reasoning, called the Yale Shooting Problem, can easily be formalized and solved using acyclic programs. We also show how a much larger class of problems in temporal reasoning can be solved by analogous means.

1.3 Preliminaries

For definitions, terminology and notation concerning logic programming we refer the reader to Apt[1)] or Lloyd.[17)] or 17). More specifically, for a general logic program P we use U_P, B_P, T_P, *comp*(P) and *ground*(P) as abbreviations of, respectively, the Herbrand Universe of P, the Herbrand Base of P, the immediate consequence operator of P, Clark's completion of P and the set of all variable-free instances of clauses from P. From now on we simply say *program* and *goal* instead of *general program* and *general goal*. We recall the following notions which are due to Przymusinski.[25)]

Definition 1.1
A program P is *locally stratified* if there exists a mapping *stratum* from B_P to the countable ordinals such that for every $A \leftarrow L_1, \ldots, L_n (n \geq 0)$ in *ground*(P) the following conditions hold for every $1 \leq i \leq n$:

- if L_i is positive, say L_i is B for some $B \in B_P$, then *stratum*$(A) \geq$ *stratum*(B);
- if L_i is negative, say L_i is $\neg B$ for some $B \in B_P$, then *stratum*$(A) >$ *stratum*(B).

Definition 1.2
Let P be locally stratified. A Herbrand model M of P is called a *perfect* model of P if there exists no Herbrand model of P which is *preferable* to M. Here *preferable* is the following relation between Herbrand interpretations: I is *preferable* (or *preferred*) to J if for every $A \in I - J$ there exists $B \in J - I$ such that *stratum*$(B) <$ *stratum*(A).

Although the definition of a perfect model seems to depend on *stratum* and it is not obvious from this definition that perfect models exist, Przymusinski has shown in Przymusinski[25] that every locally stratified program has a unique perfect model. It is easily seen that the perfect model of P is a minimal Herbrand model of P (a smaller Herbrand model would be preferable to it). Consequently, if P is a positive logic program, then the perfect model of P equals the least Herbrand model of P.

The following three basic definitions are straightforward generalizations of definitions given in Bezem[4]. The first two definitions can also be found in Cavedon[7].

Definition 1.3
Let P be a program. A *level mapping* for P is a function $| \ |: B_P \rightarrow \mathbf{N}$ of variable-free atoms to natural numbers. We extend $| \ |$ to variable-free literals by putting $|\neg A| = |A|$ for all $A \in B_P$. For $A \in B_P$ we call $|(\neg)A)|$ the *level* of $(\neg)A$.

Definiton 1.4
Let P be a program and $| \ |$ a level mapping for P. We call P *acyclic with respect to* $| \ |$ if for every $A \leftarrow L_1, \ldots, L_n (n \geq 0)$ in *ground* (P), $|A| > |L_i|$ for every $1 \leq i \leq n$. Moreover, P is called *acyclic* if P is acyclic with respect to some level mapping for P.

A simple example which will play a prominent role in this article is provided by a formalization of the so-called Yale Shooting Problem of Hanks and McDermott[14] by the program *YSP* consisting of the clauses (a)-(e) below. In this section *YSP* serves only as an example of an acyclic program. We postpone the discussion of this program to Section 5, where *YSP* is considered as a key example of the special form of non-monotonic reasoning captured by acyclic programs.

$$holds(alive,\ [\,])\leftarrow \tag{a}$$
$$holds(loaded,\ [load\,|\,x_{situation}])\leftarrow \tag{b}$$
$$holds(dead,\ [shoot\,|\,x_{situation}])\leftarrow holds(loaded,\ x_{situation}) \tag{c}$$
$$ab(alive,\ shoot,\ x_{situation})\leftarrow holds(loaded,\ x_{situation}) \tag{d}$$
$$holds(x_{fact},\ [x_{event}\,|\,x_{situation}])\leftarrow \neg\, ab(x_{fact},\ x_{event},\ x_{situation}),\ holds(x_{fact},\ x_{situation}) \tag{e}$$

In this program $x_{situation}$, x_{fact}, x_{event} are variables, *alive*, *dead*, *loaded*, *load*, *shoot* are constants and we have used a representation of lists such as in LISP or PROLOG. The empty list is represented by a constant denoted $[\,]$. If L is a list and t a term, then the list with t as first element (the *head*), followed by the list L (the *tail*) is represented by the term $[t\,|\,L]$, denoting the application of a binary function to t and L. Furthermore $[t_1, ..., t_n\,|\,L]$ abbreviates $[t_1\,|\,[\cdots[t_n\,|\,L]\cdots]]$, and $[t_1, ..., t_n]$ abbreviates $[t_1, ..., t_n\,|\,[\,]](n\geq 1)$. In the alphabet of *YSP* every variable-free term is either a constant, or a term $[t_1\,|\,t_2]$. Hence we can define a mapping l: $U_{YSP}\rightarrow\mathbf{N}$ by $l(t) = 0$ if t is a constant and $l([t_1\,|\,t_2]) = 1 + l(t_2)$. We define a level mapping $|\ |$: $B_{YSP}\rightarrow\mathbf{N}$ by $|holds(t,\ t')| = 2l(t')$ and $|ab(t,\ t',\ t'')| = 2l(t'') + 1$, so that we have

$$|holds(t,\ [h\,|\,l])| > |ab(t',\ t'',\ l)| > |holds(t''',\ l)|$$

for all variable-free terms t, ..., t''', h, l. Now it is not difficult to see that *YSP* is acyclic with respect to $|\ |$.

Definition 1.5
A literal L is called *bounded* with respect to a level mapping $|\ |$ if $|\ |$ is bounded on the set $[L]$ of variable-free instances of L. If L is bounded, then $|[L]|$ denotes the maximum that $|\ |$ takes on $[L]$. We then say that L is bounded by l if $l\geq|[L]|$. A general goal $G = \leftarrow L_1, ..., L_n (n\geq 0)$ is called *bounded* if every $L_i (1\leq i\leq n)$ is bounded. If G is bounded then $|[G]|$ denotes the (finite) multiset (see Dershowitz[9]) consisting of the natural numbers $|[L_1]|, ..., |[L_n]|$.

The following easy lemmas are instrumental in proving the termination of a number of inference procedures for acyclic programs and bounded goals.

Lemma 1.6
Let $|\ |$ be a level mapping and L a bounded literal. Then, for every substitution θ, $L\theta$ is bounded and $|[L\theta]|\leq|[L]|$.

Proof
Follows immediately from $[L\theta]\subseteq[L]$. □

Lemma 1.7
Let P be acyclic with respect to $|\ |$. Then for every clause $A\leftarrow L_1, ..., L_n (n\geq 0)$ from P and every substitution θ we have: if $A\theta$ is bounded, then every $L_i\theta$

is bounded and $|[L_i\theta]| < |[A\theta]| (1 \le i \le n)$.

Proof

For every $L' \in [L_i\theta] (1 \le i \le n)$ there exists a variable-free instance $A' \leftarrow L'_1, \ldots, L'_n$ of $A\theta \leftarrow L_1\theta, \ldots, L_n\theta$, and hence of $A \leftarrow L_1, \ldots, L_n$, such that L' occurs in the body. Since P is acyclic and $A\theta$ is bounded, it follows that $|[A\theta]| \ge |A'| > |L'|$. Now the conclusions of the lemma immediately follow. □

§2 Declarative Semantics of Acyclic Programs

In this section we define the declarative semantics of acyclic programs. We follow the 2-valued and 3-valued approach in successive subsections. In general these approaches lead to different semantics, but, among others, we show that in the case of acyclic programs they lead to the same declarative semantics.

2.1 The 2-Valued Approach

Let $|\ |$ be a level mapping for a program P. We can view $|\ |$ as a way of partitioning the Herbrand Base B_P. Any partition of B_P naturally induces a partition on every Herbrand interpretation $I \subseteq B_P$. Let us denote these partition classes by $I(n)$, so $I(n) = \{A \in I \mid |A| = n\}$ for all $n \in \mathbf{N}$.

Definition 2.1

Let P be acyclic with respect to $|\ |$. The declarative semantics of P is defined as a specific Herbrand interpretation M for P; M is the union of an inductively defined sequence $M(0), M(1), \ldots$ of subsets of B_P such that $M(n)$ contains only atoms of level n (thus conforming to the notation just introduced). This sequence is defined as follows:

$$M(0) = \{A \mid |A| = 0 \text{ and } A \leftarrow \text{ is in } ground(P)\}$$
$$M(n+1) = \{A \mid |A| = n+1 \text{ and there exists } A \leftarrow L_1, \ldots, L_k (k \ge 0) \text{ in } ground(P) \text{ such that } \bigcup_{i \le n} M(i) \models L_1 \wedge \cdots \wedge L_k\}.$$

Alternatively, for all n

$$M(n) = T_P(\bigcup_{i<n} M(i)) \cap B_P(n).$$

At first sight the declarative semantics M of P seems to depend on the level mapping. However, it follows from the Lemmas 2.3 and 2.4 below that this is not the case. Therefore we denote from now on the declarative semantics of an acyclic program P by M_P.

Lemma 2.2

For all interpretations I and variable-free literals L we have $I \models L$ iff $I(|L|) \models L$.

Proof

Trivial. □

Lemma 2.3
Let P be acyclic. Then M_P is a fixpoint of T_P.

Proof
Let P be a program which is acyclic with respect to a level mapping $|\ |: B_P \to \mathbf{N}$. As to $M_P \subseteq T_P(M_P)$, suppose that $A \in M_P$. Then $A \in M_P(|A|)$. If $|A| = 0$, then $A \in T_P(M_P)$ by the definition of $M_P(0)$. If $|A| > 0$, then $A \in T_P(M_P)$ by the construction of $M_P(|A|)$, the acyclicity of P and Lemma 2.2.

Conversely, if $A \in T_P(M_P)$, then there exists $A \leftarrow L_1, \ldots, L_k (k \geq 0)$ in *ground*(P) such that $M_P \models L_1 \wedge \cdots \wedge L_k$. Since P is acyclic, we have $|A| > |L_i|$ for all $1 \leq i \leq k$. Now, again by the construction of M_P and by Lemma 2.2, it follows that $A \in M_P$. □

Lemma 2.4
The T_P operator of an acyclic program has at most one fixpoint.

Proof
Let I and J be fixpoints of T_P for some general program P which is acyclic with respect to a level mapping $|\ |: B_P \to \mathbf{N}$. We shall prove by induction on n that $\bigcup_{i<n} I(n) = \bigcup_{i<n} J(n)$, which immediately implies $I = J$. For $n = 0$ there is nothing to prove. Assume $\bigcup_{i<n} I(n) = \bigcup_{i<n} J(n)$. We have to prove $I(n) = J(n)$. Let $A \in I(n)$; then $A \in I = T_P(I)$, so there exists $A \leftarrow L_1, \ldots, L_k (k \geq 0)$ in *ground*(P) such that $I \models L_1 \wedge \cdots \wedge L_k$. Since P is acyclic, we have $|L_j| < |A| = n$ for all $1 \leq j \leq k$. It follows by Lemma 2.2 above that $\bigcup_{i<n} I(i) \models L_1 \wedge \cdots \wedge L_k$, so by the induction hypothesis we have $\bigcup_{i<n} J(i) \models L_1 \wedge \cdots \wedge L_k$. By Lemma 2.2 $J \models L_1 \wedge \cdots \wedge L_k$ so we have $A \in T_P(J) = J$, hence $A \in J(n)$. We have proved $I(n) \subseteq J(n)$, and the converse follows by symmetry. This completes the induction step. □

Theorem 2.5
Let P be an acyclic program. Then we have:

(i) T_P has a unique fixpoint, M_P;
(ii) M_P is a minimal model of P;
(iii) M_P is the perfect model of P;
(iv) M_P is the unique Herbrand model of *comp*(P).

Proof
(i) By the Lemmas 2.3 and 2.4.
(ii) Assume by contradiction that $N \subset M_P$ is a model of P. Let n be the smallest natural number such that $N(n) \neq M_P(n)$. Now a contradiction follows by inspection of the construction of M_P, the acyclicity of P and Lemma 2.2.
(iii) We first observe that a level mapping naturally induces a local

stratification (see Przymusinski[25]) of the acyclic program. Hence every acyclic program has a unique perfect model. To show that M_P is perfect, assume by contradiction that a model N of P is preferable to M_P. Since M_P is minimal by (ii), it follows that there exists an atom $A \in N$ such that $A \notin M_P$. Let A be such a variable-free atom having the lowest level. By the definition of the preference relation between the models M_P and N there exist $B \in M_P$ such that $B \notin N$ and $|B| < |A|$. Let B be such a variable-free atom having the lowest level. It follows that $\bigcup_{i<|B|} M_P(i) = \bigcup_{i<|B|} N(i)$ and $N(|B|) \subset M_P(|B|)$. Now the desired contradiction follows in a similar way as under (ii).

(iv) We recall that fixpoints of the T_P operator of a general program are exactly the Herbrand models of the completion of that program (see Apt[1], Lemma 7.1 (ii)). □

2.2 The 3-Valued Approach

In this subsection we provide yet another characterization of the model M_P of an acyclic program—in terms of 3-valued models. First we recall the necessary background results, due to Fitting[12], which use a 3-valued logic due to Kleene[15].

In Kleene's logic there are three truth values: **t** for true, **f** for false and **u** for undefined. Every connective takes the value **t** or **f** if it takes that value in 2-valued logic for all possible replacements of **u**'s by **t** or **f**; otherwise it takes value **u**.

A Herbrand interpretation for this logic (called a 3-*valued* Herbrand interpretation) is defined as a pair (T, F) of disjoint sets of variable-free atoms. Given such an interpretation $I = (T, F)$ a variable-free atom A is true in I if $A \in T$, false in I if $A \in F$ and undefined otherwise. Given $I = (T, F)$ we denote T by I^+ and F by I^-. Thus $I = (I^+, I^-)$.

Every (2-valued) Herbrand interpretation I for a program P determines a 3-valued Herbrand interpretation $(I, B_P - I)$. Therefore, in the remainder of this subsection we identify every 2-valued Herbrand interpretation I with its 3-valued counterpart $(I, B_P - I)$.

Given a program P, the 3-valued Herbrand interpretations for P form a complete partial order with the ordering $\subseteq$ defined by

$$I \subseteq J \text{ iff } I^+ \subseteq J^+ \wedge I^- \subseteq J^-$$

and least element (ϕ, ϕ).

Following Fitting[12], given a program P we define an operator Φ_P on this complete partial order of 3-valued Herbrand interpretations for P as follows:

$$\Phi_P(I) = (T, F),$$

where

$T = \{A \mid$ there exists $A \leftarrow L_1, \ldots, L_k$ in *ground*(P) with $L_1 \wedge \cdots \wedge L_k$ true in $I\}$,
$F = \{A \mid$ for all $A \leftarrow L_1, \ldots, L_k$ in *ground*(P), $L_1 \wedge \cdots \wedge L_k$ is false in $I\}$.

It is easy to see that T and F are disjoint, so $\Phi_P(I)$ is indeed a 3-valued Herbrand interpretation. Φ_P is a natural generalization of the operator T_P to the case of 3-valued logic.

The powers of Φ_P are defined in analogy to those of T_P:

$$\Phi_P \uparrow 0 = (\phi, \phi),$$
$$\Phi_P \uparrow (\alpha + 1) = \Phi_P(\Phi_P \uparrow \alpha),$$
$$\Phi_P \uparrow (\alpha) = \bigcup_{\beta < \alpha} \Phi_P \uparrow \beta \text{ for any limit ordinal } \alpha.$$

Φ_P is easily seen to be monotonic, so $\Phi_P \uparrow \alpha \subseteq \Phi_P \uparrow \beta$ whenever $\alpha \leq \beta$.

We have the following result.

Lemma 2.6
Let P be an acylic program. Then $M_P = \Phi_P \uparrow \omega$.

Proof
Let P be acyclic with respect to a level mapping $|.|$. Consider the sequence $M(0)$, $M(1)$, ... of subsets of B_P constructed in Definition 2.1. We prove by simultaneous induction on n the following two claims:

(i) $A \in M(n)$ iff $A \in \Phi_P \uparrow (n+1)^+ \cap B_P(n)$;
(ii) $A \in B_P(n) - M(n)$ iff $A \in \Phi_P \uparrow (n+1)^- \cap B_P(n)$.

Note that by our notational conventions $\Phi_P \uparrow (n+1)^+ \cap B_P(n) = \Phi_P \uparrow (n+1)^+(n)$ and $\Phi_P \uparrow (n+1)^- \cap B_P(n) = \Phi_P \uparrow (n+1)^-(n)$.

The base case is clear since

$$
\begin{aligned}
& A \in M(0) \\
\leftrightarrow\ & A \leftarrow \text{ is in } \textit{ground}\ (P) \text{ and } |A| = 0 \\
\leftrightarrow\ & A \in \Phi_P \uparrow 1^+ \cap B_P(0)
\end{aligned}
$$

and

$$
\begin{aligned}
& A \in B_P(0) - M(0) \\
\leftrightarrow\ & A \leftarrow \text{ is not in } \textit{ground}(P) \text{ and } |A| = 0
\end{aligned}
$$

(since P is acyclic)

$\leftrightarrow$ there exists no $A \leftarrow L_1, \ldots, L_k$ in *ground*(P) such that $L_1 \wedge \cdots \wedge L_k$ is true in $\Phi_P \uparrow 0$ and $|A| = 0$
$\leftrightarrow$ $\Phi_P \uparrow 1^- \cap B_P(0)$.

Assume now that the claim holds for all $m < n$. Then

$A \in M(n)$

$\leftrightarrow$ there exists $A \leftarrow L_1, \ldots, L_k$ in *ground*(P) such that $\bigcup_{i<n} M(i) \models L_1 \wedge \cdots \wedge L_k$, and $|A| = n$

(since P is acyclic)

$\leftrightarrow$ there exists $A \leftarrow L_1, \ldots, L_k$ in *ground*(P) such that $\bigcup_{i<n} M(i) \models L_j$ and $|L_j| < n$ for all $1 \leq j \leq k$, and $|A| = n$

(by the acyclicity of P, the induction hypothesis and the fact that $\Phi_P \uparrow \alpha \subseteq \Phi_P \uparrow \beta$ whenever $\alpha \leq \beta$)

$\leftrightarrow$ there exists $A \leftarrow L_1, \ldots, L_k$ in *ground*(P) such that $L_1 \wedge \cdots \wedge L_k$ is true in $\Phi_P \uparrow n$, and $|A| = n$

$\leftrightarrow$ $A \in \Phi_P \uparrow (n+1)^+ \cap B_P(n)$

and

$A \in B_P(n) - M(n)$

$\leftrightarrow$ there exists no $A \leftarrow L_1 \wedge \cdots \wedge L_k$ in *ground*(P) such that $\bigcup_{i<n} M(i) \models L_1 \wedge \cdots \wedge L_k$, and $|A| = n$

(since P is acyclic)

$\leftrightarrow$ for all $A \leftarrow L_1 \wedge \cdots \wedge L_k$ in *ground*(P) we have $|L_j| < n$ for all $1 \leq j \leq k$, and $\bigcup_{i<n} M(i) \not\models L_1 \wedge \cdots \wedge L_k$, and $|A| = n$

(by the acyclicity of P, the induction hypothesis and the fact that $\Phi_P \uparrow \alpha \subseteq \Phi_P \uparrow \beta$ whenever $\alpha \leq \beta$)

$\leftrightarrow$ for all $A \leftarrow L_1, \ldots, L_k$ in *ground* (P) we have that $L_1 \wedge \cdots \wedge L_k$ is false in $\Phi_P \uparrow n$, and $|A| = n$

$\leftrightarrow$ $A \in \Phi_P \uparrow (n+1)^- \cap B_P(n)$.

This proves the induction step. By induction, for all variable-free atoms A,

$$A \in M_P \text{ iff } A \in \Phi_P \uparrow \omega^+$$

and

$$A \in B_P - M_P \text{ iff } A \in \Phi_P \uparrow \omega^-,$$

i.e. by the identification of 2-valued with 3-valued Herbrand interpretations

$$M_P = \Phi_P \uparrow \omega. \quad \square$$

Corollary 2.7
Let P be an acyclic program. Then M_P is the unique fixpoint of Φ_P.

Proof

We have $\Phi_P \uparrow \omega \subseteq \Phi_P \uparrow (\omega + 1)$, so by Lemma 2.6 $M_P \subseteq \Phi_P(M_P)$. But for no 3-valued Herbrand interpretation I, $M_P \subset I$(otherwise $I^+ \cap I^- \neq \phi$), so $M_P = \Phi_P(M_P)$, i.e. M_P is a fixpoint of Φ_P. Moreover, by the monotonicity of Φ_P, every fixpoint of the form $\Phi_P \uparrow \alpha$ is contained in any other fixpoint, so in fact M_P is the unique fixpoint of Φ_P. □

The advantage of the characterization of M_P by Lemma 2.6 over its original definition is that the construction of $\Phi_P \uparrow \omega$ does not refer to any level mapping.

It is worthwhile to note that even though for an acyclic program P the least fixpoint of Φ_P is reached by iterating Φ_P ω times, the operator Φ_P does not need to be continuous.

Lemma 2.8

For some acyclic program P the operator Φ_P is not continuous.

Proof

Let $P = \{r(s(0)) \leftarrow, \ p \leftarrow q(\mathrm{x})\}$. Then P is clearly acyclic. Define now a sequence of 3-valued Herbrand interpretations I_0, I_1, ... as follows:

$$I_i = (\phi, \{q(s^j(0)) \mid j < i\}),$$

where we identify $s^0(0)$ with 0. Clearly $I_i \subset I_{i+1}$ for $i < \omega$ and

$$\bigcup_{i<\omega} = (\phi, [q(x)]).$$

Now for $i < \omega$ we have $p \notin \Phi_P(I_i)^-$ because for the clause $p \leftarrow q(s^i(0))$ in *ground*(P), the atom $q(s^i(0))$ is not false in I_i. However, $p \in \Phi_P(\bigcup_{i<\omega} I_i)^-$ since for every $i < \omega$ the atom $q(s^i(0))$ is false in $\bigcup_{i<\omega} I_i$. Thus $\bigcup_{i<\omega} \Phi_P(I_i) \neq \Phi_P(\bigcup_{i<\omega} I_i)$.

□

Remarks 2.9

(a) Theorem 2.5 has been found independently by Cavedon[7)] in the slightly stronger version for locally hierarchical programs. Most of the results in Section 2 can be easily generalized to locally hierarchical programs. However, this is not true for the next sections.

(b) Recall that $T_P(I) \subseteq I$ if and only if $I \models P$ (Apt[1)], Proposition 5.12), so that (i) and (ii) of Theorem 2.5 imply that the unique fixpoint of the T_P operator of an acyclic program is also its minimal pre-fixpoint.

(c) In Przymusinski[25)] (Proposition 1) it is proved that every perfect model is minimal, so (iii) implies (ii) in Theorem 2.5 above. Note that in our proof (ii) is used to prove (iii) and that the proof that M is a perfect model is particularly simple as compared to the argument in Przymusinski.[25)]

(d) It is tempting to think that Theorem 2.5 (iv) could be sharpened in the sense that the completion of an acyclic program might be a complete theory (proving or disproving every sentence), or even a categorical theory (all models being isomorphic). This, however, is not the case as shown by the following example. Let P be the acyclic program consisting of the two clauses $p(0) \leftarrow$, $q \leftarrow \neg p(x)$ (with $|p(0)| = 0$ and $|q| = 1$). Then *comp*(P) consists, apart from the axioms of free equality (which do not play a role here and are given in the next section), of the following two completed definitions.

$$p(z) \leftrightarrow z = 0$$
$$q \leftrightarrow \exists x \, \neg p(x)$$

The unique Herbrand model of *comp*(P) is $\{p(0)\}$. However, *comp*(P) has non-Herbrand models in which q is valid, for example $\mathbb{N}$ with p interpreted as *zero*, where q is true since *zero*(1) does not hold. Note that addition of a domain closure axiom $\forall x(x = 0)$ to *comp*(P) yields a categorical theory in the special case of this example. Although this phenomenon does not hold for acyclic programs in general, we show in the sequel that adding a domain closure axiom to the completion of an acyclic program yields a complete theory with respect to formulas in which only bounded atoms occur.

§3 Completion Semantics of Acyclic Programs

In this section we investigate in detail the completion of acyclic programs. We show that any bounded atom can be effectively reduced to an equality formula that is equivalent to that atom modulo the completion of the acyclic program. Apart from suggesting an interpreter for bounded atoms, this reduction enables us to prove that the declarative semantics of an acyclic program is decidable.

Notation 3.1

We use the vector notation $\boldsymbol{t}$(resp. $\boldsymbol{x}$) to denote a sequence of zero or more terms (resp. distinct variables). Furthermore, $\boldsymbol{t} = \boldsymbol{s}$ abbreviates the conjunction $t_1 = s_1 \wedge \ldots \wedge t_n = s_n$, where $\boldsymbol{t} = t_1, \ldots, t_n$ and $\boldsymbol{s} = s_1, \ldots, s_n$. Similarly, $\boldsymbol{L}$ abbreviates the conjunction $L_1 \wedge \cdots \wedge L_n$ of the literals occurring in $\boldsymbol{L}$. Also $\forall \boldsymbol{x}$ abbreviates $\forall x_1 \ldots \forall x_n$. The empty conjunction stands for *verum*, a true proposition, dually to the convention that an empty disjunction, such as the empty goal, stands for *falsum*, a false proposition. If, for any syntactic expression E, we write $E(\boldsymbol{x})$, then no other variables occur in E than those explicitly shown in $\boldsymbol{x}$. If we abbreviate sequences of variables in a different way, say $\boldsymbol{x}$ and $\boldsymbol{y}$, then all the variables occurring in $\boldsymbol{x}$ and $\boldsymbol{y}$ are supposed to be distinct. We do not use this convention for abbreviations of sequences of terms and literals. Syntactic identity is denoted by $\equiv$.

Definition 3.2
The theory of free equality, denoted by EQ, is defined by the following axiom schemata.

$$f(\boldsymbol{x}) = f(\boldsymbol{y}) \rightarrow \boldsymbol{x} = \boldsymbol{y} \text{ for all function symbols } f,$$
$$\neg(f(\boldsymbol{x}) = g(\boldsymbol{y})) \text{ whenever } f \not\equiv g,$$
$$\neg(x = t) \text{ for all terms } t \not\equiv x \text{ such that } x \text{ occurs in } t.$$

As usual for first order logic with equality we interpret $=$ as the identity relation on the domain of interpretation. Consequently, we do not have to axiomatize $=$ as a congruence relation. □

Lemma 3.3[8]
(i) If $\boldsymbol{t}(\boldsymbol{x})$ and $\boldsymbol{s}(\boldsymbol{y})$ do not unify, then $EQ \models \forall \boldsymbol{x}\ \forall \boldsymbol{y}\ \neg(\boldsymbol{t}(\boldsymbol{x}) = \boldsymbol{s}(\boldsymbol{y}))$.
(ii) If $\boldsymbol{t}(\boldsymbol{x})$ and $\boldsymbol{s}(\boldsymbol{y})$ do unify, then there exists an mgu $\theta = \{\cdots, x_i/u_i, \cdots, \cdots, y_j/v_j, \cdots\}$ of $\boldsymbol{t}(\boldsymbol{x})$ and $\boldsymbol{s}(\boldsymbol{y})$ such that all the variables occurring in θ are among $\boldsymbol{x}, \boldsymbol{y}$ and $EQ \models \forall \boldsymbol{x}\ \forall \boldsymbol{y}\ [\boldsymbol{t}(\boldsymbol{x}) = \boldsymbol{s}(\boldsymbol{y}) \leftrightarrow (\boldsymbol{x} = \boldsymbol{u} \wedge \boldsymbol{y} = \boldsymbol{v})]$.

Here and below it is understood that $u_i \equiv x_i$ (resp. $v_j \equiv y_j$) if θ does not contain a binding for x_i (resp. y_j). □

For a simple proof of the above lemma, based on the use of the Martelli-Montanari unification algorithm, see Apt[1](Lemma 5.21).

Theorem 3.4 (Equivalence Theorem, or substitutivity for logical equivalents)
Let T be a theory and ϕ' a formula obtained from a formula ϕ by replacing some occurrences of formulas $\phi_1, ..., \phi_n$ by $\phi_1', ..., \phi_n'$ respectively. If $T \models \phi_1 \leftrightarrow \phi_1', ..., T \models \phi_n \leftrightarrow \phi_n'$, then $T \models \phi \leftrightarrow \phi'$.

Proof
This is just a mild generalization of the Equivalence Theorem in Shoenfield[29] (3.4). It should be noted that the replacement of formulas may involve renaming of variables to avoid variable clashes. □

Lemma 3.5
Let P be acyclic with respect to $|\ |: B_P \rightarrow \mathbb{N}$. For every bounded atom A there exists a formula ϕ_A, all whose free variables occur in A, such that $comp(P) \models A \leftrightarrow \phi_A$ and all atoms A' occurring in ϕ_A are either equality atoms, or are bounded with $|[A']| < |[A]|$.

Proof
The proof is essentially by unfolding completed definitions. The decrease in the bound on the level of variable-free instances is ensured since the atom is bounded and the program is acyclic, but the price is the introduction of equality formulas which express the unification process. Let $A \equiv p(\boldsymbol{s}(\boldsymbol{y}))$ be a bounded atom. Consider the completed definition

$$p(\boldsymbol{z}) \leftrightarrow (F_1(\boldsymbol{z}) \vee \cdots \vee F_n(\boldsymbol{z}))\ (n \geq 0)$$

of p in *comp(P)*. Fix $1 \leq i \leq n$ and assume that $F_i(\boldsymbol{z})$ originates from the program clause $p(\boldsymbol{t}(\boldsymbol{x})) \leftarrow \boldsymbol{L}(\boldsymbol{x})$ from P. (The denotation $p(\boldsymbol{t}(\boldsymbol{x})) \leftarrow \boldsymbol{L}(\boldsymbol{x})$ is meant to express that $\boldsymbol{x}$ are all the variables occurring in the clause, and not that these variables occur all both in the head and in the body of the clause.) We have

$$F_i(\boldsymbol{z}) \equiv \exists \boldsymbol{x}(\boldsymbol{z} = \boldsymbol{t}(\boldsymbol{x}) \wedge \boldsymbol{L}(\boldsymbol{x}))$$

and distinguish the following two cases.

Case 1

$p(\boldsymbol{t}(\boldsymbol{x}))$ and $p(\boldsymbol{s}(\boldsymbol{y}))$ do not unify. Then by Lemma 3.3 (i), we have

$$EQ \models \forall \boldsymbol{x}\ \forall \boldsymbol{y}\ \neg(\boldsymbol{t}(\boldsymbol{x}) = \boldsymbol{s}(\boldsymbol{y}))$$

and so

$$comp(P) \models \forall \boldsymbol{y}\ \neg F_i(\boldsymbol{s}(\boldsymbol{y})).$$

Case 2

$p(\boldsymbol{t}(\boldsymbol{x}))$ and $p(\boldsymbol{s}(\boldsymbol{y}))$ do unify. Let θ be as in Lemma 3.3 (ii). Then by the Theorem 3.4 we have

$$comp(P) \models \forall \boldsymbol{y}[F_i(\boldsymbol{s}(\boldsymbol{y})) \leftrightarrow \exists \boldsymbol{x}(\boldsymbol{x} = \boldsymbol{u} \wedge \boldsymbol{y} = \boldsymbol{v} \wedge \boldsymbol{L}(\boldsymbol{x}))].$$

Since $=$ is interpreted as identity, we obviously have

$$\models \forall \boldsymbol{x}\ \forall \boldsymbol{y}[(\boldsymbol{x} = \boldsymbol{u} \wedge \boldsymbol{y} = \boldsymbol{v} \wedge \boldsymbol{L}(\boldsymbol{x})) \leftrightarrow (\boldsymbol{x} = \boldsymbol{u} \wedge \boldsymbol{y} = \boldsymbol{v} \wedge \boldsymbol{L}(\boldsymbol{u}))].$$

Since $\boldsymbol{L}(\boldsymbol{u}) \equiv \boldsymbol{L}(\boldsymbol{x})\theta$, it follows that

$$comp(P) \models \forall \boldsymbol{y}[F_i(\boldsymbol{s}(\boldsymbol{y})) \leftrightarrow \exists \boldsymbol{x}(\boldsymbol{x} = \boldsymbol{u} \wedge \boldsymbol{y} = \boldsymbol{v} \wedge \boldsymbol{L}(\boldsymbol{x})\theta)].$$

This completes the second case of the case distinction.

After these preparations the construction of ϕ_A can be given. We have

$$comp(P) \models A \leftrightarrow (F_1(\boldsymbol{s}(\boldsymbol{y})) \vee \cdots \vee F_n(\boldsymbol{s}(\boldsymbol{y}))). \qquad (\dagger)$$

Let $1 \leq i \leq n$ and consider $F_i(\boldsymbol{s}(\boldsymbol{y}))$. In Case 1 we simply delete $F_i(\boldsymbol{s}(\boldsymbol{y}))$ from (†). In Case 2 we replace $F_i(\boldsymbol{s}(\boldsymbol{y}))$ by $\exists \boldsymbol{x}(\boldsymbol{x} = \boldsymbol{u} \wedge \boldsymbol{y} = \boldsymbol{v} \wedge \boldsymbol{L}(\boldsymbol{x})\theta)$. Let ϕ_A be the resulting right hand side. By Theorem 3.4 it follows that *comp* $(P) \models A \leftrightarrow \phi_A$. It remains to show that all atoms A' occurring in ϕ_A that are not equality atoms are bounded and satisfy $|[A']| < |[A]|$. This can be seen as follows. Recall that $A \equiv p(\boldsymbol{s}(\boldsymbol{y}))$ and $A' \equiv L_j(\boldsymbol{x})\theta$ for some L_j occurring in the body of a program clause $p(\boldsymbol{t}(\boldsymbol{x})) \leftarrow \boldsymbol{L}(\boldsymbol{x})$. We obviously have $|[p(\boldsymbol{s}(\boldsymbol{y}))]| \geq |[p(\boldsymbol{s}(\boldsymbol{y}))\theta]|$. Moreover, $p(\boldsymbol{s}(\boldsymbol{y}))\theta \equiv p(\boldsymbol{t}(\boldsymbol{x}))\theta$. Finally,, by Lemma 1.7, $|[p(\boldsymbol{t}(\boldsymbol{x}))\theta]| > |[L_j(\boldsymbol{x})\theta]|$ since $p(\boldsymbol{t}(\boldsymbol{x})) \leftarrow \boldsymbol{L}(\boldsymbol{x})$ is a clause of the acyclic program P. It follows that $|[L_j(\boldsymbol{x})\theta]| < |[p(\boldsymbol{s}(\boldsymbol{y}))]|$, i.e. $|[A']| < |[A]|$. □

The following theorem is to be compared to Lemma 3 in the addendum of Apt and Blair[2]. The class of acyclic programs is considerably larger than the class of recursion-free programs. On the other hand, the reduction to equality formulas can no longer be obtained for arbitrary atoms, but only for bounded atoms.

Theorem 3.6
Let P be acyclic. For every bounded atom A there exists a formula ϕ_A, all whose free variables occur in A, such that ϕ_A contains only equality atoms and $comp(P) \models A \leftrightarrow \phi_A$.

Proof
By induction on $|[A]|$, using Theorem 3.4 and Lemma 3.5 above. □

Corollary 3.7
Let P be acyclic. For every formula F in which only bounded atoms occur there exists a formula ϕ_F, all whose free variables occur in F, such that ϕ_F contains only equality atoms and $comp(P) \models F \leftrightarrow \phi_F$.

Proof
By induction on the length of the formula using Theorem 3.6 □

The above results are constructive in the sense that they yield an effective procedure to reduce a bounded atom, or even a bounded formula (i.e. a formula F such as in Corollary 3.7), to a formula in the language of equality which is equivalent to the original formula modulo *comp*(P). It is interesting to view this reduction as an abstract interpretation. It should be noted that the equality formula ϕ_F from Corollary 3.7 can become arbitrarily complex and one can dispute whether indeed we achieve a *reduction* of F. In the sequel we argue that, due to a result of Malcev[21], recently rediscovered independently by Maher[20], we can indeed claim to have achieved a reduction. Let us first give a simple example of the proposed reduction technique.

The completion of the example program *YSP* from Section 1 consists of *EQ* and the following two completed definitions.

$$\begin{aligned}
&ab(x, y, z) \leftrightarrow \exists u(x = alive \wedge y = shoot \wedge z = u \wedge holds(loaded, u)),\\
&holds(x, y) \leftrightarrow ((x = alive \wedge y = []) \vee\\
&\qquad \exists u(x = loaded \wedge y = [load \mid u]) \vee\\
&\qquad \exists u(x = dead \wedge y = [shoot \mid u] \wedge holds(loaded, u)) \vee\\
&\qquad \exists u, v, w(x = u \wedge y = [v \mid w] \wedge \neg ab(u, v, w) \wedge holds(u, w)).
\end{aligned}$$

We shall reduce the bounded atom *holds*(*alive*, $[x, y]$) to an equality formula equivalent to it modulo *comp*(*YSP*). The reduction technique will essentially be the one described in the proof of Lemma 3.5, but from time to time we shall simplify the formulas to enhance readability. For example, we write $F(t)$ instead of $\exists x(x = t \wedge F(x))$ for a formula F and a term t in which x does not occur. Moreover, we move quantifiers to the innermost position. The first unfolding operation yields that

$$holds(alive, [x, y]) \tag{1}$$

is equivalent (modulo *comp*(*YSP*)) to

$$\exists u, v, w(u = alive \wedge v = x \wedge w = [y] \wedge \neg ab(alive, x, [y]) \wedge holds(alive, [y])), \tag{2}$$

since only the last disjunct of the completed definition of *holds* can apply here on the account of the equality axioms *EQ*. Formula (2) simplifies to

$$\neg ab(alive, x, [y]) \wedge holds(alive, [y]). \tag{3}$$

Unfolding again and simplifying these two atoms yields

$$\neg(x = shoot \wedge holds(loaded, [y])) \wedge \neg ab(alive, y, []) \wedge holds(alive, []). \tag{4}$$

Obviously $holds(alive, [])$ unfolds to the empty conjunction or *verum*, so that it can be deleted from (4). Moreover $ab(alive, y, [])$ is unfolded and simplified as $y = shoot \wedge holds(loaded, [])$. Now $holds(loaded, [])$ is unfolded to the empty disjunction or *falsum*, so that $ab\ (alive, y, [])$ is seen to be equivalent to *falsum*. Hence its negation is equivalent to *verum* and consequently can be deleted from (4). It remains to unhold $holds(loaded, [y])$, which after simplification yields

$$y = load \vee (\neg ab(loaded, y, []) \wedge holds(loaded, [])). \tag{5}$$

Although $ab(loaded, y, [])$ unfolds to *falsum*, the conjunction in (5) is equivalent to *falsum*, since the right conjunct has already been shown to be equivalent to *falsum*. So this conjunction can be deleted, which yields $y = load$. Summarizing all unfolding and simplification we get the following result:

$$holds(alive, [x, y]) \leftrightarrow \neg(x = shoot \wedge y = load), \tag{6}$$

which is exactly in accordance with the intended meaning of *YSP*. Note that the goal $\leftarrow holds(alive, [x, y])$ flounders, so that neither by SLDNF- nor by SLS-resolution one can obtain a correct answer substitution, and certainly not an equivalent equality formula such as the one given by (6).

It is well known that domain closure in first order logic is only possible for finite domains. Then the domain closure axiom reads $\forall x(x = c_0 \vee \cdots \vee x = c_n)$. It is tempting to generalize this formula to the case of infinite domains by

$$\bigvee_{t \in U_P} x = t,$$

but this is an infinitary axiom which goes beyond first order logic. By the Löwenheim-Skolem Theorem (see Shoenfield[29](5.3)), every theory having an infinite model admits models of arbitrary infinite cardinality. This simply excludes that an infinite domain can be characterized by a first order theory. However, an interesting first order approximation of a domain closure axiom can be formulated provided that the alphabet of the language contains only finitely many function symbols (cf. Maher[20]). *Par abus de langage* this axiom is usually called domain closure axiom and abbreviated by *DCA*.

Definition 3.8
DCA is the axiom

$$\forall x \bigvee_{f \in L} \exists y_1 \cdots \exists y_{r_f}\ x = f(y_1, \ldots, y_{r_f}).$$

In this definition constants are taken as function symbols of arity 0 and r_f denotes the arity of f. Thus, *DCA* depends on the alphabet of the language L and hence on the program P. Since it will always be clear from the context which P is meant, we do not express this dependence in the denotation. Note that *DCA* is satisfied in all Herbrand interpretations of P.

Theorem 3.9
Let P be acyclic. Then for every bounded atom A we have either $comp(P) \cup DCA \models \forall A$ or $comp(P) \cup DCA \models \neg \forall A$. Moreover, it is decidable which of these two possibilities holds.

Proof
Follows from the Theorem 3.6 and the result of Malcev[21], recently rediscovered independently in Maher[20], that $EQ \cup DCA$ is a complete and decidable theory.
□

Before we finish this section with a simple corollary of this theorem we show that both *DCA* and the condition that the goal is bounded are necessary. Consider the program P from Remarks 2.9(d). Then $comp(P) \models \neg q \leftrightarrow \forall z(z = 0)$. Now q is true in the non-Herbrand model given in Remarks 2.9(d), but false in any model satisfying $DCA \equiv \forall z(z = 0)$, such as M_P.

Regarding the condition that the goal is bounded, consider $P = \{p(0) \leftarrow, p(f(x)) \leftarrow p(x)\}$. Then P is obviously acyclic and the goal $\leftarrow p(x)$ is not bounded. Furthermore, the completed definition of p is $p(z) \leftrightarrow (z = 0 \vee \exists x(z = f(x) \wedge p(x)))$. We have $M_P \models comp(P) \cup DCA \cup \{\forall p(x)\}$, but for M with domain $\mathbf{N} \cup \mathbf{Z}$, with $p(x)$ interpreted as $x \in \mathbf{N}$, 0 as $0 \in \mathbf{N}$ and f as the successor function on both $\mathbf{N}$ and $\mathbf{Z}$, we have $M \models comp(P) \cup DCA \cup \{\neg \forall p(x)\}$.

Corollary 3.10
Let P be acyclic. Then M_P is recursive and satisfies for all $A \in B_P$

$$A \in M_P \text{ iff } comp(P) \cup DCA \models A.$$

Proof
By Theorem 2.5, Theorem 3.9 and the fact that variable-free atoms are bounded. □

§4 Procedural Semantics of Acyclic Programs

Among the various approaches to the procedural semantics of logic programming with negation, the most prominent are SLDNF-resolution, see Lloyd[17], and SLS-resolution from Przymusinski.[26][27] One of the difficulties concerning SLDNF-resolution is that it cannot be defined in a top down manner. This problem was overlooked in the first edition of Lloyd[17] and was solved in Ref. 17) by defining SLDNF-trees inductively bottom up. As a result, for certain programs and goals like $P = \{p \leftarrow p\}$ and $G = \leftarrow \neg p$, or $P = \{p \leftarrow \neg p\}$ and $G = \leftarrow p$, no SLDNF-derivation needs to exist. We show that this problem does not arise for acyclic programs.

SLS-resolution as defined in Przymusinski[26] for stratified programs in a top down manner. This definition can be naturally extended to the case of locally stratified programs (see Bol[6] for a rigorous definition), so a fortiori to the case of acyclic programs.

The major distinction between SLDNF- and SLS-resolution lies in the way they treat negation. SLS uses a *negation as failure* rule, whereas SLDNF uses negation as *finite* failure. A minor distinction between SLDNF and SLS is the way they treat *floundering*, i.e. the appearance of a goal consisting entirely of negative literals containing variables. Since floundering is not our main concern here, we shall simply ignore this distinction. More precisely, by SLDNF we mean a variant of SLDNF in which floundering is treated in the same systematic way as done in SLS. The following results can be established about these forms of resolution for acyclic programs.

Theorem 4.1
Let P be an acyclic program and G a bounded goal. Then every SLS-tree as well as every SLDNF-tree of G contains only bounded goals and is finite.

Proof
We argue in a way similar to Bezem[4](Lemma 2.5 and Corollary 2.6). For the multiset ordering we refer to Dershowitz[9]. Let G be a bounded goal. We distinguish the following three cases. If a positive literal of G is selected, then it follows by Lemma 1.6 and Lemma 1.7 that every resolvent G' of G is bounded and that indeed $|[G']|$ is smaller than $|[G]|$ in the multiset ordering. If G consists entirely of negative literals containing variables, then there is no resolvent at all. If a variable-free negative literal is selected, then both for SLS- and SLDNF-resolution we trivially have that the resolvent (if any) of G is bounded and smaller than G in the multiset ordering. Now use the fact that the

multiset ordering over $\mathbf{N}$ is well-founded. □

Corollary 4.2
Let P be an acyclic program and G a goal. Then, for any selection rule, the SLS-tree and the SLDNF-tree of G coincide.

Proof
We recall that the difference between SLS and SLDNF amounts to negation as failure versus negation as *finite* failure. Since all goals $\leftarrow A$ with $A \in B_P$ are bounded, it follows by Theorem 4.1 that $\leftarrow A$ fails if and only if $\leftarrow A$ fails *finitely*. Now the corollary easily follows. □

Since SLS-derivations always exist for locally stratified programs (so a fortiori for acyclic programs), this corollary implies that for all acyclic programs and goals SLDNF-derivations exist. Moreover, for acyclic programs SLDNF-resolution can be defined in a top down manner, as it coincides then with SLS-resolution. Additionally we have:

Corollary 4.3
Let P be an acyclic program. Then both SLS- and SLDNF-resolution are decidable rules of inference.

Proof
If a positive literal is selected in a goal, then every inference step is obviously decidable. If a goal consists entirely of negative literals containing variables, then there is no resolvent at all. Now assume a negative literal $\neg A$ with $A \in B_P$ is selected. Then $\leftarrow A$ is bounded, so for any selection rule both the SLS- and the SLDNF-tree of $\leftarrow A$ are finite. So $\leftarrow A$ either succeeds, flounders or (finitely) fails. Moreover, it is decidable which of these cases hold. It follows that the inference step (if any) is decidable. □

Theorem 4.1 implies in particular that all variable-free goals terminate when the program is acyclic. One would expect a converse, stating that a program is acyclic when all variable-free goals terminate, analogous to Bezem[4] (Theorem 2.10) for the positive case. However, by the presence of negation the situation is delicate.

One source of termination is floundering as illustrated by the following small program P:

$$p(0) \leftarrow \neg p(x).$$

The only variable-free literals are $p(0)$ and $\neg p(0)$; every variable-free goal consists solely of occurrences of these two atoms. These variable-free goals all terminate, since they flounder, whereas P is obviously not acyclic.

Another source of termination is safety. SLDNF-resolution is only sound with *safe* selection rules, i.e. those not selecting negative literals containing variables. Consider the following program Q as an example of this phenomenon:

$$a \leftarrow c, \ \neg d(x),$$
$$d(s(x)) \leftarrow d(x),$$
$$d(0) \leftarrow.$$

We have the following SLDNF-derivation: $\leftarrow a$, $\leftarrow c, \ \neg d(x)$. Only the literal c can be selected safely now, so the SLDNF-derivation ends with failure since there is no program clause with c as head. It can be easily verified that every variable-free goal terminates. But Q is not acyclic.

When these sources of (abnormal) termination are excluded by imposing appropriate conditions on a logic program P, then it becomes possible to prove that termination of all variable-free goals implies that P is acyclic. However, these conditions are quite constraining (ruling out e.g. program Q above), so the result is rather weak. Therefore we refrain from giving the details which are quite involved.

We close this section by combining results previously obtained in this paper with results from Cavedon[7] and Przymusinski[27] to obtain the following characterizations of the model M_P.

Theorem 4.4
Let P be an acyclic program. Then we have:

(i) T_P has a unique fixpoint, M_P;
(ii) M_P is the perfect model of P;
(iii) M_P is the unique Herbrand model of *comp*(P);
(iv) M_P is the unique fixpoint of Φ_P;
(v) for all variable-free atoms A,
$A \in M_P$ iff *comp*(P) $\cup$ $DCA \models A$;
(vi) for all variable-free atoms A that do not flounder,
$A \in M_P$ iff there exists an SLDNF-refutation of $P \cup \{\leftarrow A\}$;
(vii) for all variable-free atoms A that do not flounder,
$A \in M_P$ iff there exists an SLS-refutation of $P \cup \{\leftarrow A\}$;
(viii) M_P is recursive.

Proof
(i), (ii) and (iii) follow from Theorem 2.5, (iv) from Corollary 2.7, (v) from Corollary 3.10, (vi) from Cavedon[7] and (vii) is implied by results from Przymusinski[27]. In fact (vi) and (vii) are special cases of more general completeness results. Note that (vi) and (vii) are equivalent by Corollary 4.2. Finally, (viii) follows from Theorem 3.9 and Corollary 3.10. □

It is worthwhile to note here that some of the results listed in the above theorem can also be derived using more general results concerning general programs and their subclasses, proved by Kunen[16], Shepherdson[28], and Przymusinski[27]. However, our proofs are more direct and simpler.

§5 Application—Temporal Reasoning

5.1 Yale Shooting Problem

In Hanks and McDermott[14] a simple problem in temporal reasoning is discussed. It became known in the literature as the 'Yale Shooting Problem'. Hanks and McDermott's interest in this problem arose from the fact that apparently all theories about non-monotonic reasoning, when used to formalize this problem, led to too weak conclusions. The problem has been extensively discussed in the literature and several solutions to it have been proposed, e.g., by means of circumscription (see Lifschtz[19]) or epistemic logic (see Gelfond[13]). In Hanks and McDermott[14] some of these solutions are discussed and critically evaluated.

In this section we present a particularly simple solution to the above problem by means of acyclic programs. First, let us explain the problem. We closely follow here Hanks and McDermott[14] (p. 387). Consider a single individual who in any situation can be either *alive* or *dead*, and a gun that can be either *loaded* or *unloaded*. The following statements are stipulated.

(1) At some specific situation s_0 the person is alive.
(2) The gun becomes loaded any time a *load* event happens.
(3) Any time the person is shot with a loaded gun, he becomes dead. Moreover, the fact of staying alive is abnormal with respect to the event of being shot with a loaded gun.
(4) Facts which are not abnormal with respect to an event remain true.

To formalize these statements Hanks and McDermott[14] use McCarthy and Hayes'[23] situation calculus in which one distinguishes three entities: *facts*, *events* and *situations*.

Facts can hold true in situations and situations can be changed by the occurrence of events. To express statements involving facts, events and situations, relation symbols t and ab and a function symbol *result* are used.

Given a fact f, event e and a situation s

- $t(f, s)$ means that fact f is true in situation s,
- $result(e, s)$ denotes the situation resulting from occurrence of event e in situation s,
- $ab(f, e, s)$ means that 'fact f is abnormal with respect to event e occurring in situation s' or 'occurrence of event e in situation s causes f to stop being true in $result(e, s)$'.

Using this notation[14] formulate the above statements (1)-(4) as the following formulas:

$$t(alive, s_0), \tag{1}$$

$$\forall s\ t(loaded,\ result(load,\ s)), \tag{2}$$
$$\forall s(t(loaded,\ s) \rightarrow (ab(alive,\ shoot,\ s) \wedge t(dead,\ result(shoot,\ s)))), \tag{3}$$
$$\forall f\ \forall e\ \forall s((t(f,\ s) \wedge \neg ab(f,\ e,\ s)) \rightarrow t(f,\ result(e,\ s))). \tag{4}$$

Thus

- *alive*, *dead* and *loaded* are interpreted as constants 'of type *fact*',
- *load* and *shoot* are interpreted as constants 'of type *event*',
- s_0 is interpreted as a constant 'of type *situation*'.

(While an explicit use of types in the underlying first order language would result in a more rigorous description, their use is not needed for the purpose at hand.) The last formula (4) is often called *inertia axiom*. It is a formalization in the situation calculus of the *frame problem*.

To draw the desired conclusions from the above formulas (1)-(4), Hanks and McDermott[14] used the circumscription method of McCarthy[22] to circumscribe over the relation *ab*.

Circumscription of a set of formulas S over a relation, say p, is equivalent to computing the set of all formulas true in all models of S in which the circumscribed relation p is minimal (see Lifschitz[18]. Hanks and McDermott[14] consider the following sequence of situations:

$$\begin{aligned}
&s_0,\\
&s_1 = result(load,\ s_0),\\
&s_2 = result(wait,\ s_1),\\
&s_3 = result(shoot,\ s_2)\\
&\quad = result(shoot,\ result(wait,\ result(load,\ s_0))),
\end{aligned}$$

where *wait* is a new event whose occurrence is supposed to have no effect on the truth of the considered facts. This is taken care of by formula (4) used for $e =$ *wait* and by the lack of any formula stating that a fact is abnormal with respect to the event *wait* in a situation.

In their analysis Hanks and McDermott[14] notice that there exist two (Herbrand) models in which the circumscribed relation *ab* is minimal. In one of them, say M_1, among others the following variable-free atoms are true:

'in s_0': $t(alive,\ s_0)$,
'in s_1': $t(alive,\ s_1)$, $t(loaded,\ s_1)$, $ab(alive,\ shoot,\ s_1)$,
'in s_2': $t(alive,\ s_2)$, $t(loaded,\ s_2)$, $ab(alive,\ shoot,\ s_2)$,
'in s_3': $t(dead,\ s_3)$.

In another model, say M_2, among others the following variable-free atoms are true:

'in s_0': $t(alive,\ s_0)$,
'in s_1': $t(alive, s_1)$, $t(loaded, s_1)$, $ab(alive, shoot, s_1)$, $ab(loaded,\ wait, s_1)$,

'in s_2': $t(alive, s_2)$,
'in s_3': $t(alive, s_3)$.

It is easy to see that in every model of the formulas (1)-(4) the formulas $ab(alive, shoot, s_1)$ and $ab(loaded, wait, s_1) \vee ab(alive, shoot, s_2)$ are true. Thus M_1 and M_2 are models of (1)-(4) in which the relation ab is indeed minimal.

Now, circumscribing the formulas (1)-(4) over ab we obtain a theory from which $t(dead, s_3)$ cannot be deduced since $\neg t(dead, s_3)$ holds in M_2.

On the other hand, an intuitive analysis of the problem seems to support the formulas true in the model M_1. Indeed, we assumed that the event *wait* has no effect on the truth of the considered fact. This suggests that $\forall f\ \forall s\ \neg ab(f, wait, s)$ holds. This formula was supposed to be 'enforced' by the circumscription over the relation ab. Unfortunately, this turned out not to be the case and led to unexpected conclusions.

5.2 A Solution Using Acyclic Programs

Our solution to the Yale Shooting Problem is embarrassingly trivial: first split formula (3) into

$$\forall s(t(loaded, s) \to ab(alive, shoot, s)), \tag{3a}$$
$$\forall s(t(loaded, s) \to t(dead, result(shoot, s))). \tag{3b}$$

This obviously does not affect the description of the discussed problem. Then interpret the resulting set of formulas as a logic program. That is all.

Since for logic programs we adopted a different vocabulary (s and t denote expressions etc.), we rewrite the formulas (1), (2), (3a), (3b), (4) by

- using the relation *holds* instead of t,
- using variables x_{fact}, x_{event} and $x_{situation}$ instead of, respectively, the variables f, e and s.

Also, we write the empty list [] for s_0 and $[t \mid L]$ for $result(t, L)$, and use the clausal form as customary in logic programming. Thus, by using the abbreviations concerning lists as given in Subsection 1.3, for example $[shoot, wait, load]$ stands for the situation s_3. To be formally correct we add a constant *wait* to the alphabet of *YSP*.

As a result the formulas (1), (2), (3a), (3b), (4) translate into the program *YSP* given in Section 1. We proved there that *YSP* is an acyclic program by exhibiting a simple level mapping. Consequently, to analyze it we can use any of the theorems concerning acyclic programs which are proved in Sections 2, 3 and 4.

By virtue of Theorem 4.4 and the observation that goals of the form $\leftarrow A$, where A is a variable-free atom, do not flounder with respect to *YSP*, we have:

Corollary 5.1

(i) T_{YSP} has a unique fixpoint, M_{YSP};
(ii) M_{YSP} is the perfect model of YSP;
(iii) M_{YSP} is the unique Herbrand model of $comp(YSP)$;
(iv) M_{YSP} is the unique fixpoint of Φ_{YSP};
(v) for all variable-free atoms A,
$A \in M_{YSP}$ iff $comp(YSP) \cup DCA \models A$;
(vi) for all variable-free atoms A,
$A \in M_{YSP}$ iff there exists an SLDNF-refutation of $YSP \cup \{\leftarrow A\}$;
(vii) for all variable-free atoms A,
$A \in M_{YSP}$ iff there exists an SLS-refutation of $YSP \cup \{\leftarrow A\}$;
(viii) M_{YSP} is recursive. □

This corollary provides overwhelming evidence that among all Herbrand models of YSP, M_{YSP} is the preferred one. This model is characterized in several, vastly different ways and naturally arises when studying both declarative and procedural semantics of the program YSP.

It is useful to see that M_{YSP} coincides with the model M_1 considered in the previous subsection. Thanks to Corollary 5.1 there are several ways of checking it. Perhaps the simplest is the one using the SLDNF-resolution. We only concentrate on the crucial statement $t(dead, s_3)$ or, using the notation adopted in this section, $holds(dead, [shoot, wait, load])$.

We have the following SLDNF-refutation:

$$\begin{array}{l}
\leftarrow holds(dead, [shoot, wait, load]) \\
\qquad\qquad\qquad\qquad | \,(c) \\
\leftarrow holds(loaded, [wait, load]) \\
\qquad\qquad\qquad\qquad | \,(e) \\
\leftarrow \neg\, ab(loaded, wait, [load]), holds(loaded, [load]) \\
\qquad\qquad\qquad\qquad | \\
\leftarrow holds(loaded, [load]) \\
\qquad\qquad\qquad\qquad | \,(b) \\
\qquad\qquad\qquad\qquad \square
\end{array}$$

The subsidiary derivation of $\neg\, ab(loaded, wait, [load])$ by means of negation as failure is trivial as $ab(loaded, wait, [load])$ does not unify with any head of the clauses (a)-(e).

It is also easy to check that the statement $t(alive, s_3)$, or in other words $holds(alive, [shoot, wait, load])$, cannot be derived using SLDNF-resolution since $\neg\, ab(alive, shoot, [wait, load])$ cannot be established by means of negation as failure. In fact, it is easy to prove the converse by exhibiting an SLDNF-refutation of $YSP \cup \{\leftarrow \neg\, holds(alive, [shoot, wait, load])\}$.

It should be pointed out here that the idea of using logic progrmming to solve the Yale Shooting Problem has been proposed independently by others. In particular, Elkan[10)] showed that the translation of the Yale Shooting Problem to a logic program results in a locally stratified program and Evans[11)] observed that SLDNF-resolution can be used to compute desired consequences of the formulas (1)-(4). Moreover, as pointed out by one of the referees of this paper, the logic programming solution to the Yale Shooting Problem coincides with a natural translation of the solution proposed in Morris[24)], which is based on the use of non-normal default theory. His solution can be translated to a locally stratified program using the appropriate equivalence result established in Bidoit and Froidevaux[5)](Theorem 4.1.4).

However, our characterization of the logic programming solution as an acyclic program leads to several additional characterizations of and insights in the model M_{YSP} which are collected in Corollary 5.1.

5.3 Temporal Reasoning Using Acyclic Programs

How general are the considerations concerning the Yale Shooting Problem? It is an instance of a problem in temporal reasoning and it is by no means clear that our proposed solution also applies to other problems of a similar kind. In this subsection we exhibit a large class of problems in temporal reasoning which can be solved by analogous means.

Let us adopt the notation used in the previous subsection. In case of a temporal reasoning we can naturally identify the following four types of statements.

(1) In some set of situations a certain fact holds unconditionally. Each such statement can be represented by an unconditional clause

$$holds(f,\ t) \leftarrow$$

for some fixed fact represented by a constant f and a term t (possibly containing variables) representing a set of situations.

(2) In a certain situation a certrain fact holds provided some other fact holds in a previous situation. Each such statement can be represented by a clause

$$holds(f,\ [e\,|\,s]) \leftarrow holds(f',\ s)$$

for some facts f, f', event e and situation s.

(3) In a given situation a certain event affects certain facts unconditionally. Each such statement can be represented by an unconditional clause

$$ab(f,\ e,\ s) \leftarrow$$

for some fact f, event e and situation s.

(4) In a given situation a certain event affects a certain fact provided some other fact holds in this situation. Each such statement can be represented by a clause

$$ab(f,\ e,\ s) \leftarrow holds(f',\ s)$$

for some facts f, f', event e and a situation s.

Denote claue (e) of the program *YSP* by *IA* (for *inertia axiom*). The following observation is crucial.

Lemma 5.2
Let P be a program consisting of clauses of the form (1)-(4). Then $P \cup \{IA\}$ is acyclic.

Proof
We can use here the same level mapping as the one used for the general program *YSP*, i.e.,

$$|holds(t,\ t')| = 2l(t'),$$
$$|ab(t,\ t',\ t'')| = 2l(t'') + 1$$

where $l(t) = 0$ if t is a constant, and $l([t_1 \mid t_2]) = 1 + l(t_2)$ otherwise. It is easy to check that $P \cup \{IA\}$ is acyclic w. r. t. $|\ |$. □

This lemma allows us to apply our theory of acyclic programs to *any* temporal reasoning problem which can be described by means of statements (1)-(4). Thus any such problem naturally yields a model which can be viewed as a solution to the problem. This model—the perfect Herbrand model of the corresponding acyclic program $P \cup \{IA\}$—can be characterized in a number of equivalent ways, both semantically and proof theoretically.

Moreover, by virtue of Theorem 4.1, for a large number of questions, namely those which can be expressed as bounded and non-floundering goals, we can use SLDNF-resolution to compute the desired answers. More precisely, we have the following result.

Lemma 5.3
Let P be a program consisting of clauses of the form (1)-(4) and G a goal such that the last argument in each of its literals is a list. Then all SLDNF-derivations of $P \cup \{IA\} \cup \{G\}$ are finite.

Proof
Consider the level mapping $|\ |$ exhibited in the proof of Lemma 5.2. By assumption each of the literals of G is bounded w. r. t. $|\ |$, since for any list t and substitution θ we have $l(t) = l(t\theta)$. Thus G is bounded w. r. t. $|\ |$ and the conclusion follows by Theorem 4.1 □

By our choice of using the list notation instead of the repeated application of the *result* function, a list appearing in the last argument of a literal of G stands for a sequence of situations (of a fixed length). Thus the condition put on G in the above lemma amounts to saying that all of its literals refer to situations that are bounded in time. When additionally such a goal does not flounder we can thus use SLDNF-resolution (or PROLOG) to effectively compute *all* answers to it.

We can extend the use of acyclic programs to temporal reasoning even further by assuming the existence of a well-founded ordering on facts, say $<$, and allowing additionally to clauses of the form (1)-(4), clauses of the form

$$holds(f, s) \leftarrow holds(f', s) \tag{5}$$

where f and f' are facts such that $f' < f$ and s is a situation,

$$ab(f, e, s) \leftarrow holds(f', s), ab(f', e, s) \tag{6}$$

where f and f' are facts such that $f' < f$, e is an event and s is a situation.

Such clauses express naturally arising statements. For example, Gelfond[13] considers an extension of the original formulation of the Yale Shooting Problem by assuming two new statements expressed by the following clauses:

$$holds(breath, s) \leftarrow holds(alive, s),$$
$$ab(breath, e, s) \leftarrow holds(alive, s), ab(alive, e, s).$$

Adding these clauses to the problem *YSP* yields an acyclic program. Indeed, we have the following lemma.

Lemma 5.4
Let P be a program consisting of clauses of the form (1)-(6). Then $P \cup \{IA\}$ is acyclic.

Proof
Let $<$ be the well-founded ordering on the facts used in the clauses of the form (5) and (6). Let *norm* be a function from facts to natural numbers such that $norm(f') < norm(f)$ whenever $f' < f$. Since there are only finitely many facts used in P, the function *norm* has a maximum, say $k - 1$. Define now the following bound function

$$|holds(t, t')| = norm(t) + 2k \cdot l(t'),$$
$$|ab(t, t', t'')| = norm(t) + 2k \cdot l(t'') + k.$$

It is easy to see that $P \cup \{IA\}$ is acyclic w. r. t. $| \ |$. □

Acknowledgements

We thank V. Lifschitz, P. Mancarella, D. Pedreschi and one of the referees for helpful comments on the subject of this paper. This research has been partially supported by ESPRIT BRA 3020 'Integration'.

References

1) Apt, K. R., "Logic Programming," in *Handbook of Theoretical Computer Science*,

Vol. B (J. van Leeuwen, ed.), Elsevier, Amsterdam, pp. 493-574, 1990.

2) Apt, K. R. and Blair, H. A., "Arithmetic Classification of Perfect Models of Stratified Programs," *Fundamenta Informaticae, 13*, pp. 1-18, 1990 (with an addendum in vol. 14, pp. 339-343, 1991).

3) Apt, K. R., Blair H. A. and Walker, A., "Towards a Theory of Declarative Knowledge," in *Foundations of Deductive Databases and Logic Programming* (J. Minker, ed.), Morgan Kaufmann, Los Altos, pp. 89-148, 1988.

4) Bezem, M., "Characterizing Termination of Logic Programs with Level Mappings," in *Proc. North American Conference on Logic Programming*, Cleveland, Ohio, pp. 69-80, 1989.

5) Bidoit, N. and Froidevaux, C., "General Logical Databases and Programs: Default Logic Semantics and Stratification," *Information and Computation, 91*, pp. 15-54, 1991.

6) Bol, R., "Loop Checking and Negation," *Report CS-R9075*, Centre for Mathematics and Computer Science, Amsterdam, 1990.

7) Cavedon, L., "Continuity, Consistency, and Completeness Properties for Logic Programs," in *Proc. of the 6th International Conference on Logic Programming*, The MIT Press, pp. 571-584, 1989.

8) Clark, K. L., "Negation as Failure," in *Logic and Data Bases*, (H. Gallaire and J. Minker, eds.), Plenum Press, New York, pp. 293-322, 1978.

9) Dershowitz, N., "Termination of Rewriting," *Journal of Symbolic Computation, 3*, pp. 69-116, 1987.

10) Elkan, C., "A Perfect Logic for Reasoning about Action," *manuscript*, University of Toronto, 1989.

11) Evans, C., "Negation-as-Failure as an Approach to the Hanks and McDermott Problem," in *Proc. of the 2nd International Symposium on Artificial Intelligence*, Monterrey, Mexico, 1989.

12) Fitting, M., "A Kripke-Kleene Semantics for General Logic Programs," *Journal of Logic Programming, 2*, pp. 295-312, 1985.

13) Gelfond, M., "Autoepistemic Logic and Formalization of Commonsense Reasoning," in *Proc. 2nd Workshop on Nonmonotonic Reasoning, Lecture Notes in Artificial Intelligence*, Springer-Verlag, Berlin, 1988.

14) Hanks, S. and McDermott, D., "Nonmonotonic Logic and Temporal Projection," *Artificial Intelligence, 33*, pp. 379-412, 1987.

15) Kleene, S. C., *Introduction to Metamathematics*, van Nostrand, New York, 1952.

16) Kunen, K., "Signed Data Dependencies in Logic Programs," *Journal of Logic Programming, 7*, pp. 231-246, 1989.

17) Lloyd, J. W., *Foundations of Logic Programming, 2nd Edition*, Springer-Verlag, Berlin, 1987.

18) Lifschitz, V., "Computing Circumscription," in *Proc. IJCAI-85*, pp. 121-127, 1985.

19) Lifschitz, V., "Formal Theories of Action," in *The Frame Problem in Artificial Intelligence* (F. Brown, ed.), Morgan Kaufmann, Los Altos, California, 1987.

20) Maher, M. J., "Complete Axiomatizations of the Algebras of Finite, Rational and Infinite Trees," in *Proc. of the 3rd Annual Symposium on Logic in Computer Science*, Edinburgh, pp. 348-357, 1988.

21) Malcev, A., "Axiomatizable Classes of Locally Free Algebras of Various Types," in *The Metamathematics of Algebraic Systems: Collected Papers, Chapter 23*, North-Holland, Amsterdam, pp. 262-281, 1971.

22) McCarthy, J., "Circumscription—A Form of Non-Monotonic Reasoning," *Artificial Intelligence, 13*, pp. 27-39, 1980.

23) McCarthy, J. and Hayes, P. J., "Some Philosophical Problems from the Standpoint of

Artificial Intelligence," in *Machine Intelligence 4* (B. Meltzer and D. Mitchie, eds.), Edinburgh University Press, Edinburgh, pp. 463-502, 1969.

24) Morris, P., "Curing Anomalous Extensions," in *Proc. of AAAI*, pp. 437-442, 1987.

25) Przymusinski, T. C., "On the Declarative Semantics of Deductive Databases and Logic Programs," in *Foundations of Deductive Databases and Logic Programming* (J. Minker, ed.), Morgan Kaufmann Publishers, Los Altos, pp. 193-216, 1988.

26) Przymusinski, T. C., "On the Declarative and Procedural Semantics of Logic Programs," *Journal of Automated Reasoning*, *5*, pp. 167-205, 1989.

27) Przymusinski, T. C., "Every Logic Program Has a Natural Stratification and an Iterated Least Fixed Point Model," in *Proc. Principles of Database Systems* (*PODS'89*), Philadelphia, 1989.

28) Shepherdson, J. C., "Negation in Logic Programming," in *Foundations of Deductive Databases and Logic Programming* (J. Minker, ed.), Morgan Kaufmann Publishers, Los Altos, pp. 19-88, 1988.

29) Shoenfield, J. R., *Mathematical Logic*, Addison-Wesley, Reading MA, 1967.