Acyclic logic programs and the completeness of SLDNF-resolution

Lawrence Cavedon*

Centre for Cognitive Science, University of Edinburgh, Edinburgh EH8 9LW, UK

Abstract


This paper investigates the class of acyclic programs, programs with the usual hierarchical condition imposed on ground instances of atoms rather than predicate symbols. The acyclic condition seems to naturally capture much of the recursion that occurs in practice and many programs that arise in practical programming satisfy this condition. We prove completeness of SLDNF-resolution for the acyclic programs and discuss several other desirable properties exhibited by programs belonging to this class.

1. Introduction

The class of locally hierarchical programs generalizes that of the hierarchical programs by imposing the hierarchical constraint on ground instances of atoms rather than on predicate symbols. Unlike the hierarchical condition, which prevents any recursion and is too strong for practical programming, the locally hierarchical condition is general enough to allow many programs arising in practice.

The locally hierarchical programs exhibit desirable properties similar to several of those possessed by the hierarchical programs; in particular, we show that the completion semantics and the perfect model semantics coincide for Herbrand interpretations. Unfortunately, however, these desirable properties do not extend to the completeness of SLDNF-resolution [12]. We define a subclass of the locally hierarchical programs by imposing the condition that every atom is assigned a finite level, and show that many programs written in practice satisfy this condition. This condition was introduced by Cavedon [5], who called it locally $\omega$-hierarchical. Apt and Bezem [1] renamed this condition to acyclic, and we adopt their nomenclature here. We prove the completeness of SLDNF-resolution for the acyclic programs, under the condition of allowedness [13].

Bezem [4] and Apt and Bezem [1] have also investigated the acyclic programs, identifying interesting properties related to both the termination behaviour of

* This paper was written while the author was at the Australian AI Institute, Carlton 3053, Australia.
SLDNF-resolution and the decidability of the completion semantics. We briefly review their results below. Given the many desirable properties of the acyclic programs and the simple yet very general acyclic condition, we conclude that the acyclic programs constitute a very interesting class of programs.

2. Preliminaries

In this section, we define the class of locally hierarchical programs, as well as the related classes of locally stratified and locally call-consistent programs. The terminology throughout follows that of Lloyd [12]. We use letters of the Greek alphabet, e.g., $\alpha, \beta, \gamma, \ldots$ to denote ordinals, with $\omega$ being the first limit ordinal greater than 0.

**Definition.** A *program clause* is a clause of the form $A \leftarrow L_1, \ldots, L_m$, where $A$ is an atom and $L_1, \ldots, L_m$ are literals. A *normal program* is a finite set of program clauses. A *normal goal* is a clause of the form $\leftarrow L_1, \ldots, L_m$, where $L_1, \ldots, L_m$ are literals.

We often refer to normal programs and normal goals simply as *programs* and *goals*, respectively. Throughout, we use $U_P$ to denote the Herbrand universe, $B_P$ the Herbrand base, $T_P$ the immediate consequence operator, and $\text{comp}(P)$ the completion of a program $P$ (see also [12]).

We use the concept of a *level mapping* on ground atoms to define the following classes of programs. The *locally stratified* programs were introduced by Przymusinski [15] and the *locally call-consistent* programs by Sato [17], who called them *order-consistent*.

**Definition.** An *atomic level mapping* of a normal program is a mapping from its Herbrand base to the countable ordinals. We refer to the value of a ground atom $A$ under this mapping as the *level* of $A$ and denote it by $\text{level}(A)$.

**Definition.** A normal program $P$ is *locally hierarchical* if it has an atomic level mapping such that, for every ground instance $A \leftarrow B_1, \ldots, B_n, \neg C_1, \ldots, \neg C_m$ of a clause in $P$, we have $\text{level}(B_i) < \text{level}(A)$, $1 \leq i \leq n$, and $\text{level}(C_j) < \text{level}(A)$, $1 \leq j \leq m$.

**Definition.** A normal program $P$ is *locally stratified* if it has an atomic level mapping such that, for every ground instance $A \leftarrow B_1, \ldots, B_n, \neg C_1, \ldots, \neg C_m$ of a clause in $P$, we have $\text{level}(B_i) \leq \text{level}(A)$, $1 \leq i \leq n$, and $\text{level}(C_j) < \text{level}(A)$, $1 \leq j \leq m$.

Before defining the next class of programs, we define the concept of dependency between atoms. The *dependency graph* of a program was introduced by Apt et al. [2]. We modify this concept to one in which nodes of the graph are ground atoms rather than predicate symbols.
Definition. Let \( P \) be a normal program. The atomic dependency graph for \( P \) is the directed graph defined as follows:

(i) each node in the graph is a ground atom in \( B_P \);
(ii) let \( A \) and \( B \) be ground atoms in \( B_P \). There is an edge from \( A \) to \( B \) if there is a ground instance of a clause in \( P \) such that \( A \) is the head and \( B \) occurs in the body. The edge is marked positive (resp., negative) if \( B \) occurs in a positive (resp., negative) literal in the body of the clause instance. An edge may be both positive and negative.

Definition. Let \( P \) be a normal program and let \( A, B \) be ground atoms in \( B_P \). We say \( A \) depends positively (resp., negatively) on \( B \) if, in the atomic dependency graph for \( P \), there is a path (possibly of length zero) from \( A \) to \( B \) containing an even (resp., odd) number of negative edges. We say \( A \) depends on \( B \) if \( A \) depends positively or negatively on \( B \).

Allowing paths of length zero in the above definition ensures that every ground atom depends on itself positively. We now define the locally call-consistent programs. Sato [17] identifies (via a more complicated definition) this same class of programs, which he calls order-consistent programs.

Definition. A normal program \( P \) is locally call-consistent if it has an atomic level mapping such that, for any atoms \( A, B \) in \( B_P \), if \( A \) depends on \( B \) then \( \text{level}(A) \geq \text{level}(B) \), and if \( A \) depends both positively and negatively on \( B \) then \( \text{level}(A) > \text{level}(B) \).

Using Cavedon's [5] alternative characterization of the locally call-consistent programs, it can be shown that the above definition defines the same class of programs as does Sato's order-consistency condition.

The above classes of programs generalize the classes of hierarchical, stratified and call-consistent programs as we would hope: every hierarchical program is locally hierarchical, every stratified program is locally stratified, and every call-consistent program is locally call-consistent. Furthermore, every locally hierarchical program is locally stratified, and every locally stratified program is locally call-consistent. Unfortunately, Cholak [7] shows that each of these classes of programs is undecidable.

Sato [17] has shown that every locally call-consistent program has a consistent completion. A significantly shorter proof of this result was later given by Cavedon [5]. On examination of either the proof of Sato or that of Cavedon, we see that we can actually infer the stronger result that, if \( P \) is a locally call-consistent program, then \( \text{comp}(P) \) has a Herbrand model. Obvious corollaries to this result are the existence of a Herbrand model for \( \text{comp}(P) \), for \( P \) a locally hierarchical or locally stratified program. The former of these corollaries is used later.
Recently, Przymusinski [15] has introduced the \textit{perfect model} semantics as an alternative declarative semantics for logic programs. He argues that a program’s intended semantics is represented by its perfect models, defined as follows.

\textbf{Definition.} Let \( P \) be a program and \( I \) and \( J \) Herbrand interpretations for \( P \). We say \( I \) is \textit{preferable} to \( J \) if, for every \( A \in I - J \), there exists some \( B \in J - I \) such that there is a path in the atomic dependency graph for \( P \) from \( A \) to \( B \) containing a negative edge. A Herbrand model \( M \) of \( P \) is called a \textit{perfect model} of \( P \) if there does not exist any Herbrand model of \( P \) preferable to it.

Przymusinski proves that every locally stratified program has a unique perfect Herbrand model [15, Theorem 4]. A result of the next section relates Herbrand models of a locally hierarchical program’s completion to its perfect Herbrand model.

\section{Locally hierarchical programs}

Locally hierarchical programs exhibit, within the context of Herbrand interpretations and models, analogues to some of the desirable properties possessed by hierarchical programs. In particular, as noted by Cholak [7], if \( P \) is hierarchical, then every 3-valued model [11] for \( \text{comp}(P) \) is 2-valued, whereas if \( P \) is locally hierarchical, then every 3-valued Herbrand model for \( \text{comp}(P) \) is 2-valued. Also, if \( P \) is a hierarchical program, then there exists, for any pre-interpretation \( J \) of \( P \), a single model based on \( J \) for \( \text{comp}(P) \), whereas if \( P \) is locally hierarchical, then we have the following result.

\textbf{Proposition 3.1.} For every locally hierarchical program \( P \), there exists exactly one Herbrand model for \( \text{comp}(P) \).

\textbf{Proof.} We have seen that \( \text{comp}(P) \) has at least one Herbrand model. Let \( M_1 \) and \( M_2 \) be any two Herbrand models for \( \text{comp}(P) \). \( M_1 \) and \( M_2 \) clearly agree on atoms of level 0. Now suppose \( M_1 \) and \( M_2 \) agree on atoms of level \( < \gamma \), where \( \gamma \geq 0 \) and can be either a limit or a successor ordinal. Consider any \( A \in M_1 \) such that \( A \) has atomic level \( \gamma \). Now, there exists a ground instance \( A = B_1, \ldots, B_n, \neg C_1, \ldots, \neg C_m \) of a clause in \( P \) such that \( B_i \in M_1, 1 \leq i \leq n \), and \( C_j \notin M_1, 1 \leq j \leq m \). But then, by the induction hypothesis, \( B_i \in M_2, 1 \leq i \leq n \), and \( C_j \notin M_2, 1 \leq j \leq m \). Hence \( A \in M_2 \), and, by reversing the roles of \( M_1 \) and \( M_2 \), the result follows. \( \square \)

Using Proposition 3.1 and the result that every locally stratified program has a unique perfect Herbrand model, we infer the following.
Proposition 3.2. If $P$ is a locally hierarchical program, then the unique Herbrand model of $\text{comp}(P)$ and the unique perfect Herbrand model of $P$ are the same.

Proof. Cavedon [5] uses a continuity property for the $T_p$ operator of a locally call-consistent program to give the following characterization of the perfect Herbrand model $M_p$ of a locally stratified program (cf. Apt et al.'s [2] definition of the perfect Herbrand model of a stratified program). Let $P^*_g$ denote the set of ground instances of clauses in $P$ such that the head of the clause instance has atomic level $\preceq \gamma$. Let

\[
M_0 = (T_{p_g})^\omega(0), \\
M_\gamma = (T_{p_g})^\omega(M_{\gamma-1}) \quad \text{if } \gamma \text{ is a successor ordinal}, \\
M_\gamma = (T_{p_g})^\omega(\bigcup_{\alpha < \gamma} M_\alpha) \quad \text{if } \gamma \text{ is a limit ordinal and } \gamma \neq 0.
\]

Finally, let $M_p = \bigcup_{\gamma \geq 0} M_\gamma$. It can be shown that $M_p$ is both a perfect model of $P$ and a model of $\text{comp}(P)$, and the result then follows. \qed

This result has been independently proved by Apt and Bezem [1] for the slightly smaller class of acyclic programs, defined in the following section.

The equivalence between perfect models and models of the completion does not extend to non-Herbrand models; for example, the completion of the following locally hierarchical program

\[
p(f(x)) \leftarrow p(x)
\]

has a non-Herbrand model in which $\exists x(p(x))$ holds (the proof of the completeness of the negation-as-failure rule [10] indicates how to construct one such model), whereas $\forall x(\neg p(x))$ holds in every perfect model of this program.

Unfortunately, the desirable properties of locally hierarchical programs do not extend to the completeness of SLDNF-resolution. If we consider the following locally hierarchical program $P$

\[
p(a) \leftarrow q(a) \\
p(a) \leftarrow \neg q(a) \\
q(a) \leftarrow r(x) \\
r(f(x)) \leftarrow r(x)
\]

then $\text{comp}(P) \vdash p(a)$ but $P \cup \{\leftarrow p(a)\}$ does not have an SLDNF-refutation.

4. Acyclic programs

The incompleteness demonstrated above arises because the definition of an atomic level mapping allows an atom to be assigned a level greater than or equal to $\omega$. For instance, $q(a)$ and $p(a)$ must each be assigned a transfinite level in the example.
above. In this section, we introduce the acyclic programs, a subclass of the locally hierarchical programs defined by restricting atoms to be assigned a finite atomic level, i.e. the relevant level mapping must effectively be a mapping from the Herbrand base to the natural numbers. We show that the acyclic programs possess desirable computability properties (in particular, we prove a completeness result for SLDNF-resolution) while, unlike the hierarchical programs, still being general enough to include many programs written in practice. We also discuss further properties of the acyclic programs.

**Definition.** A normal program $P$ is *acyclic* if it is locally hierarchical via an atomic level mapping that assigns each ground atom a finite level.

The following examples demonstrate that the acyclic programs are a very general class of programs. In particular, every hierarchical program is clearly acyclic. Some interesting non-hierarchical programs that are acyclic include many recursive list-processing programs, such as `member`, `append` and `reverse`. The `even` program

\[
\text{even}(0) \leftarrow \\
\text{even}(s(x)) \leftarrow \neg \text{even}(x)
\]

is not call-consistent, yet is acyclic. The following logic programming interpreter is "effectively" acyclic if the object program on which it operates is an acyclic program, and we only consider instantiations of `clause((x \leftarrow y))` consistent with the object program.

\[
\text{interp(true)} \leftarrow \\
\text{interp}(x) \leftarrow \text{clause}(x \leftarrow y), \text{interp}(y) \\
\text{interp}((x, y)) \leftarrow \text{interp}(x), \text{interp}(y) \\
\text{interp}(-x) \leftarrow \neg \text{interp}(x)
\]

(The intended interpretation of `clause((x \leftarrow y))` is that there exists a clause with head matching $x$ and body $y$, where $y$ is a conjunction of literals, in the object program on which the interpreter operates. An empty body is represented by the constant `true`.)

As for the locally hierarchical programs, the class of acyclic programs is also undecidable [1].

4.1. **Completeness of SLDNF-resolution**

Unlike hierarchical programs, acyclic programs do not, in general, satisfy the finite tree property [8] (every SLDNF-tree for the program, with respect to any goal, is finite). For example, the program $P$ consisting of the clause

\[
p(f(x)) \leftarrow p(x)
\]
is acyclic, but the SLDNF-tree for \( P \cup \{ \text{p}(x) \} \) contains an infinite branch. However, we are able to prove completeness of SLDNF-resolution for programs satisfying the following condition.

**Definition.** A normal program \( P \) is **atomically decidable** if, for every ground atom \( A \in B_P \), \( P \cup \{ \texttt{p} \} \) has either an SLDNF-refutation or a finitely failed SLDNF-tree.

Examples of classes of programs that are atomically decidable are the hierarchical programs and the **structured** programs [3]. The completeness of SLDNF-resolution for the hierarchical and structured programs ([8, 18] and [3], respectively) follows from the general completeness theorem below.

To prove the completeness of SLDNF-resolution for atomically decidable programs, we impose the following **allowedness** condition of Lloyd and Topor [13] to ensure a program \( P \) and goal \( G \) do not flounder (we say \( P \cup \{ G \} \) flounders if some attempt to construct an SLDNF-derivation of \( P \cup \{ G \} \) results in a goal consisting of only non-ground negative literals).

**Definition.** Let \( P \) be a normal program and \( G \) a normal goal. A program clause in \( P \) is **admissible** if every variable that occurs in the clause occurs either in the head or in a positive literal in the body of that clause. A program clause in \( P \) is **allowed** if every variable that occurs in the clause occurs in a positive literal in the body of that clause. \( G \) is **allowed** if every variable that occurs in \( G \) occurs in a positive literal in \( G \). \( P \cup \{ G \} \) is **allowed** if all of the following conditions are satisfied:

(a) every clause in \( P \) is admissible;
(b) every clause in the definition of a predicate symbol occurring in a positive literal in either \( G \) or the body of a clause in \( P \) is allowed;
(c) \( G \) is allowed.

Lloyd and Topor [13] prove the following result for allowed programs and goals.

**Proposition 4.1.** Let \( P \) be a normal program and \( G \) a normal goal such that \( P \cup \{ G \} \) is allowed. Then the following properties hold:

(a) \( P \cup \{ G \} \) does not flounder;
(b) each computed answer for \( P \cup \{ G \} \) is a ground substitution for all variables in \( G \).

The following lifting lemma, extending [12, Lemma 8.2] to the case of normal programs and goals, is proved by Cavedon and Lloyd [6]. Their result is proved using a stronger allowedness condition, but the identical proof holds for the result below.

**Lemma 4.2.** Let \( P \) be a normal program and \( G \) a normal goal such that \( P \cup \{ G \} \) is allowed, and let \( \theta \) be a substitution. Suppose there exists an SLDNF-refutation of \( P \cup \{ G \theta \} \). Then there exists an SLDNF-refutation of \( P \cup \{ G \} \) of the same length such
that, if \( \theta_1, \ldots, \theta_n \) are the mgu's from the SLDNF-refutation of \( P \cup \{G\theta\} \) and \( \theta'_1, \ldots, \theta'_n \) are the mgu's from the SLDNF-refutation of \( P \cup \{G\} \), then there exists a substitution \( \gamma \) such that \( \theta_1 \ldots \theta_n = \theta'_1 \ldots \theta'_n \gamma \).

The following completeness theorem for atomically decidable programs is used to prove the completeness result for acyclic programs.

**Theorem 4.3.** Let \( P \) be a normal program such that \( P \) is atomically decidable and \( \text{comp}(P) \) is consistent, and let \( G \) be a normal goal such that \( P \cup \{G\} \) is allowed. If \( \theta \) is a correct answer for \( \text{comp}(P) \cup \{G\} \), then \( \theta \) is a computed answer for \( P \cup \{G\} \).

**Proof.** Let \( G \) be the goal \( +A_1, \ldots, A_n, \sim B_1, \ldots, \sim B_m \). Shepherdson [19] shows that, if every clause in \( P \) is allowed, then every correct answer for \( \text{comp}(P) \cup \{G\} \) is a ground substitution. In fact, by a slight modification of Shepherdson's proof, it can be shown that this holds even if \( P \cup \{G\} \) is allowed, i.e. some clauses in \( P \) may be admissible. By this result, we infer that \( G\theta \) is ground. Now, since \( \text{comp}(P) \models A_i\theta, 1 \leq i \leq n \), and \( P \) is atomically decidable, then, by the soundness of SLDNF-resolution and the assumption that \( \text{comp}(P) \) is consistent, \( P \cup \{+A_i\theta\} \) has an SLDNF-refutation, \( 1 \leq i \leq n \). Similarly, \( P \cup \{-B_j\theta\} \) is finitely failed, \( 1 \leq j \leq m \), and hence, \( P \cup \{G\theta\} \) has an SLDNF-refutation. By the lifting lemma above, \( P \cup \{G\} \) has an SLDNF-refutation with computed answer \( \theta \).

We now show that any acyclic program \( P \) is atomically decidable, provided it does not flounder against any ground atomic goal. It is easily shown that this non-floundering condition is satisfied if \( P \cup \{G\} \) is allowed, for some goal \( G \).

**Proposition 4.4.** Let \( P \) be an acyclic, normal program and suppose there exists a normal goal \( G \) such that \( P \cup \{G\} \) is allowed. Then \( P \) is atomically decidable.

**Proof.** Consider any \( A \in B_p \). As mentioned above, \( P \cup \{G\} \) being allowed, for some goal \( G \), ensures that \( P \cup \{\leftarrow A\} \) is non-floundered. We prove the proposition by induction on \( n \), the atomic level of \( A \). If \( n = 0 \), then any clause in \( P \) that unifies with \( A \) must be a unit clause, and the result follows. Now suppose \( n > 0 \) and assume the result holds for every \( k < n \). Consider any computation of \( P \cup \{\leftarrow A\} \). Since any selected negative literal is ground and has atomic level less than \( n \), we can apply the induction hypothesis to show that an SLDNF-tree for \( P \cup \{\leftarrow A\} \) exists. It then easily follows that \( P \cup \{\leftarrow A\} \) either succeeds or finitely fails if \( P^+ \cup \{\leftarrow A\} \) either succeeds or finitely fails, where \( P^+ \) denotes the set of clauses formed by removing any negative literals from clauses in \( P \). By [12, Theorems 8.3 and 13.6], the latter holds if and only if \( A \notin (T_p \uparrow \omega) \setminus (T_p \downarrow \omega) \).

Let \( B \) be any ground atom in \( B_p \), and suppose \( B \) has atomic level \( m \). We show by induction on \( m \) that

\[
B \notin (T_p \downarrow (m + 1)) \setminus (T_p \uparrow (m + 1)).
\]
If $m = 0$, then either there is a ground instance $B \leftarrow$ of a unit clause in $P^+$, and $B \in (T_{p \leftarrow} \uparrow 1)$, or $B$ does not unify with any clause in $P^+$, and $B \not \in (T_{p \leftarrow} \downarrow 1)$. Now suppose $m > 0$ and the result holds for $k < m$. If $B \in (T_{p \leftarrow} \downarrow (m + 1))$, then there exists a ground instance $B \leftarrow C_1, \ldots, C_h$ of a clause in $P^+$ such that $\{C_1, \ldots, C_h\} \subseteq (T_{p \leftarrow} \downarrow m)$. But each $C_i$ has atomic level $< m$. Hence, by the induction hypothesis, $\{C_1, \ldots, C_h\} \subseteq (T_{p \leftarrow} \uparrow m)$, and $B \in (T_{p \leftarrow} \uparrow (m + 1))$.

It follows that $A \not \in (T_{p \leftarrow} \downarrow \omega) \setminus (T_{p \leftarrow} \uparrow \omega)$ and therefore $P^+ \cup \{\leftarrow A\}$ either succeeds or finitely fails, from which we infer that $P \cup \{\leftarrow A\}$ either succeeds or finitely fails, i.e. $P$ is atomically decidable.

Finally, we show the completeness of SLDNF-resolution for acyclic programs.

**Theorem 4.5.** Let $P$ be an acyclic, normal program and $G$ a normal goal such that $P \cup \{G\}$ is allowed. If $\theta$ is a correct answer for $\text{comp}(P) \cup \{G\}$, then $\theta$ is a computed answer for $P \cup \{G\}$.

**Proof.** The result follows from the consistency of the completion of a locally call-consistent program [17], Theorem 4.3, and Proposition 4.4. □

We believe that this completeness result is a very useful one. Unlike the hierarchical condition, which prevents any recursion, the acyclic condition seems to allow much of the recursion that arises in practice. In particular, programs defined "inductively" on some object, for example, the **append** program

\[
\text{append}(\text{nil}, x, x) \leftarrow \\
\text{append}(x, y, u, x, v) \leftarrow \text{append}(y, u, v),
\]

naturally seem to fit the condition that some value decreases with each recursive step (in the case of append, it is the length of the list in the first argument). Furthermore, for the recursion to terminate and the relation to be computable, we require less than $\omega$ steps to be performed. Hence, the acyclic condition seems to naturally capture much of the recursion that arises in practical programming. For this reason, we consider the above completeness result to be a very general one for programs written in practice.

### 4.2. Further properties of acyclic programs

The acyclic programs possess several other interesting properties besides the completeness of SLDNF-resolution. Bezem [4] investigates the termination behaviour of SLDNF-resolution for acyclic programs without negation. Apt and Bezem [1] extend Bezem's results on termination by considering the more general setting of acyclic programs with negation, i.e. the same class as defined above. They also present some interesting results regarding the declarative semantics of acyclic programs. We briefly review their results here.
Define a literal $L$ to be \textit{bounded}, with respect to an atomic level mapping, if there exists some finite $n$ such that every ground instance of the atom in $L$ has atomic level $\leq n$. Similarly, a goal $G$ is \textit{bounded} if every literal in $G$ is bounded. Apt and Bezem show that, for $P$ an acyclic program and $G$ a bounded goal (both with respect to the same atomic level mapping), every SLDNF-derivation for $P \cup \{G\}$ is finite; in particular, this implies the atomic decidability property of Proposition 4.4. From this result they infer that, for $P$ an acyclic program and $G$ a non-floundering goal, the SLDNF-tree and the SLS-tree [16] for $P \cup \{G\}$ coincide.

As done above for the locally hierarchical programs, Apt and Bezem independently identify the coincidence of the unique perfect Herbrand model of an acyclic program with the unique Herbrand model of its completion. They also further investigate properties of an acyclic program's completion. They show that the completion of an acyclic program, augmented by a first-order formula $DCA$ that approximates a domain closure axiom for the program's Herbrand universe (see [14]), is complete and decidable for (the universal closure of) bounded atoms: i.e. for an acyclic program $P$ and bounded atom $A$, $\text{comp}(P) \cup DCA \models \forall(A)$ or $\text{comp}(P) \cup DCA \models \neg \forall(A)$, and it is decidable which of these holds. From this result, they infer that the unique Herbrand model of an acyclic program's completion is recursive.

The following theorem of Apt and Bezem summarizes the properties of the Herbrand model of an acyclic program's completion.

\textbf{Theorem} (Apt and Bezem [1, Theorem 4.4]). \textit{Let $P$ be an acyclic program. Then the following hold:}

(i) the unique Herbrand model $M_P$ of $\text{comp}(P)$ is also the unique perfect Herbrand model of $P$;

(ii) for any formula $F$ in which only bounded atoms occur, $M_P \models F$ iff $\text{comp}(P) \cup DCA \models F$;

(iii) for any ground atom $A$ such that $P \cup \{\neg A\}$ does not flounder,

$M_P \models A$ if and only if there exists an SLDNF-refutation of $P \cup \{\neg A\}$,

iff there exists an SLS-refutation of $P \cup \{\neg A\}$.

Given the many desirable properties of the acyclic programs, relating to both termination behaviour of SLDNF- and SLS-resolution and decidability of the declarative semantics, one may wonder whether this class of programs may, in fact, be too restrictive. However, we believe that the example programs presented earlier show that the acyclic programs form a very general class. Apt and Bezem present a further interesting example of an acyclic program, one that solves the \textit{Yale shooting problem} [9], a temporal reasoning problem that has recently received much attention in the non-monotonic reasoning literature. Also, Bezem [4] proves that every total recursive function can be computed by an acyclic program without negation.
5. Conclusions

The class of acyclic programs seems to be a very general one, containing many practical logic programs. In particular, the acyclic condition seems to naturally capture much of the recursion that arises in practice, yet is expressed very simply. The main result of this paper was to prove the completeness of SLDNF-resolution for the acyclic programs—it was the investigation of this property that originally led to the identification of this class. Furthermore, several desirable properties proved by Bezem [4] and Apt and Bezem [1], relating to both the behaviour of the SLDNF operational semantics and the decidability of the program completion declarative semantics, identify the acyclic programs as a possibly important class of programs. It is also interesting to note that the acyclic programs form a large class of programs for which two important alternative declarative semantics for logic programs with negation, i.e. the completion and the perfect model semantics, coincide for Herbrand interpretations.

Note (added in proof)

It has recently been pointed out to me by A Cortesi that there is a slight error in the continuity property in [5]. Fortunately, this error is easily rectified, and all results of the current paper hold without modification.

Acknowledgment

I would like to thank the referees for suggestions on presentation.

References


