Proving Termination of General Prolog Programs

Krzysztof R. Apt
Centre for Mathematics and Computer Science
Kruislaan 413, 1098 SJ Amsterdam, The Netherlands

Dino Pedreschi
Dipartimento di Informatica, Università di Pisa
Corso Italia 40, 56125 Pisa, Italy

Abstract

We study here termination of general logic programs with the Prolog selection rule. To this end we extend the approach of Apt and Pedreschi [AP90] and consider the class of *left terminating* general programs. These are general logic programs that terminate with the Prolog selection rule for all ground goals. We introduce the notion of an *acceptable program* and prove that acceptable programs are left terminating. This provides us with a practical method of proving termination.

The converse implication does not hold but we show that under the assumption of non-floundering from ground goals every left terminating program is acceptable. Finally, we prove that various ways of defining semantics coincide for acceptable programs. The method is illustrated by giving simple proofs of termination of a "game" program and the transitive closure program for the desired class of goals.

Note. First author's work was partly supported by ESPRIT Basic Research Action 3020 (Integration). Second author's work was partly supported by ESPRIT Basic Research Action 3012 (Compulog).

1 Introduction

Motivation

Prolog is a programming language based on logic programming. However, the use of a fixed selection rule combined with the depth first search in the resulting search trees makes Prolog and logic programming different. As a consequence various completeness results linking the procedural and declarative interpretation of logic programs cannot be directly applied to Prolog programs. This mismatch makes it difficult to study Prolog programs using only the logic programming theory. Clearly the main problem is the issue of termination: a Prolog interpreter will miss a solution if all success nodes lie to the right of an infinite path in the search tree.

In our previous paper we proposed to study pure Prolog programs that terminate for all ground goals. We called such programs left terminating and claimed that most pure Prolog programs are left terminating. Then we offered a characterization of left terminating programs which allowed us to provide simple termination proofs of various "troublesome" pure Prolog programs.

The aim of this paper is to extend this approach to termination to general Prolog programs, i.e. programs allowing negative literals. More precisely, we consider here general logic programs executed with the leftmost selection rule used in Prolog. Our approach uses the concept of a level mapping (a function assigning natural numbers to ground atoms) in combination with a limited declarative knowledge about the program embodied in some interpretation I. I should be a model of the considered program P *and* a model of Clark's completion of the "negative" fragment of P.

These two concepts are combined in the notion of an acceptable program. Intuitively, a general program P is acceptable w.r.t. a level mapping and a model I if for all ground instances of the clauses of P the level of the head is greater than the level of the atoms in a certain prefix of the body. Which prefix is considered is determined by the model I. We prove that acceptable general programs are left terminating. Consequently, to prove left termination it suffices to prove acceptability.

The converse implication does not hold due to the possibility of floundering. On the other hand, we show that for programs that do not flounder from ground goals the concepts of left termination and acceptability do coincide. Also, we prove that various ways of defining semantics coincide for acceptable programs.

Once the left termination of a general Prolog program is established, non-ground terminating goals can be identified by using the concept of a bounded goal. We illustrate the use of this method by providing simple proofs of termination of a "game" program and the transitive closure program for the desired class of goals.

The problem of termination of Prolog programs attracted a lot of attention in the literature. A short overview can be found in Apt and Pedreschi [AP90]. In particular, it is interesting to contrast our approach with that of Ullman and Van Gelder [UvG88], later improved by Plümer [Plü90b, Plü90a], aimed at the automatic verification of termination of a pure Prolog program and a goal. In their approach, some sufficient conditions for termination are identified, which can be statically checked. Obviously, such an approach cannot be complete due to the undecidability of the halting problem.

We propose instead a complete method, which characterizes precisely the left terminating, non floundering programs. Additionally, in the present paper and in [AP90] we provide simple proofs of termination for programs and goals which cannot be handled using the cited approach. On the other hand, we do not determine here any conditions under which our method could be automated. This should form part of a future research.

We are aware of only one paper in which a method of proving termination of general Prolog programs is proposed — Baudinet [Bau88]. In her proposal negation is treated indirectly by dealing with termination in presence of the *cut* operator using which negation can be simulated. The present paper seems to be the first one in which negation is treated in a direct way. By virtue of our approach the termination proofs can be built in a modular way and the limited declarative knowledge ensuring termination of the program can be identified. This results in our opinion in simple arguments which formalize the reasoning used informally.

Preliminaries

Throughout this paper we use the standard notation and terminology of Lloyd [Llo87] or Apt [Apt90]. Recall that a *general clause* is a construct of the form

$$A \leftarrow L_1, \ldots, L_n$$

($n \geq 0$) where A is an atom and $L_1, \ldots, L_n$ are literals. In turn, a *general goal* is a construct of the form

$$\leftarrow L_1, \ldots, L_n$$

($n \geq 0$) where $L_1, \ldots, L_n$ are literals. A *general program* is a finite set of general clauses.

From now on we simply say *clause, goal* and *program* instead of *general clause, general goal* and *general program.* When each L_i is positive, we call a clause a *positive clause* and a goal a *positive goal.* A program whose all clauses are positive is called a *positive program.*

We use the following abbreviations for a program P:
B_P for the Herbrand Base of P,
T_P for the immediate consequence operator of P,
ground(P) for the set of all ground instances of clauses from P,
comp(P) for Clark's completion of P.

Also, we use Prolog's convention identifying in the context of a program each string starting with a capital letter with a variable, reserving other strings for the names of constants, terms or relations.

In the programs we use the usual list notation. The constant $[\,]$ denotes the empty list and $[\,.\mid.\,]$ is a binary function which given a term x and a list xs produces a new list $[\,x \mid xs\,]$ with head x and tail xs. The standard notation $[\,x_1, \ldots, x_n\,]$, for $n \geq 0$, is used as an abbreviation of $[x_1 \mid [\ldots [x_n \mid [\,]] \ldots]]$. Given a list $[\,x_1, \ldots, x_n\,]$, each x_i is called an *element* of $[\,x_1, \ldots, x_n\,]$. In general, the Herbrand Universe will also contain "impure" elements that contain $[\,]$ or $[\,.\mid.\,]$ but are not lists - for example $s([\,])$ or $[s(0) \mid 0]$ where 0 is a constant and s a unary function symbol. They will not cause any complications.

Left Termination

In this paper we consider $SLDNF$-resolution with one selection rule only – namely that of Prolog, usually called the leftmost selection rule. As S in $SLDNF$ stands for "selection rule", we denote this form of resolution by $LDNF$ (*L*inear resolution for *D*efinite clauses with *N*egation as *F*ailure).

When studying termination of general Prolog programs, i.e. programs executed using the $LDNF$-resolution it is necessary to revise the standard definitions of Lloyd [Llo87]. Indeed, according to his definitions there is no $LDNF$-derivation for $\{p \leftarrow \neg p\} \cup \{\leftarrow p\}$ whereas the corresponding Prolog execution diverges.

The appropriate revision is achieved by viewing the $LDNF$-resolution as a top down interpreter which given a program P and a goal G attempts to build a search tree for $P \cup \{G\}$ by constructing its branches in parallel. The branches in this tree are called $LDNF$-derivations for $P \cup \{G\}$ and the tree itself is called the $LDNF$-tree for $P \cup \{G\}$.

Negative literals are resolved using the negation as failure rule which calls for the construction of a subsidiary search tree. If during this subsidiary construction the interpreter diverges, the main $LDNF$-derivation *is considered* to be infinite. Adopting this view the $LDNF$-derivation for $\{p \leftarrow \neg p\} \cup \{\leftarrow p\}$ diverges because the goal $\leftarrow p$ is resolved to $\leftarrow \neg p$ and the subsequent construction of the subsidiary $LDNF$-tree for $\{p \leftarrow \neg p\} \cup \{\leftarrow p\}$ diverges.

Summarizing, by termination of a general Prolog program we actually mean termination of the underlying interpreter. By choosing variables of the input clauses and the used mgu's in a fixed way we can assume that for every program P and goal G there exists exactly one $LDNF$-tree for $P \cup \{G\}$. The subsidiary $LDNF$-trees formed during the construction of this tree are called *subsidiary* $LDNF$-trees *for* $P \cup \{G\}$.

The following notion plays an important role in our considerations.

Definition 1.1 A program P is called *left terminating* if all $LDNF$-derivations of P starting in a ground goal are finite. □

In other words, a program is left terminating if all $LDNF$-trees for P with a ground root are finite. When studying Prolog programs, one is actually interested in proving termination of a given program not only for all ground goals but also for a class of non-ground goals constituting the intended queries. Our method of proving left termination will allow us to identify for each program such a class of non-ground goals.

The following lemma will be of use later.

Lemma 1.2 *Suppose that all $LDNF$-derivations of P starting in a ground positive goal are finite. Then P is left terminating.*

Proof. It suffices to show that for all ground literals L all $LDNF$-derivations of $P \cup \{\leftarrow L\}$ are finite. When L is positive it is a part of the assumptions and when L is negative, say $L = \neg A$, it follows from the fact that by assumption the subsidiary $LDNF$-tree for $P \cup \{\leftarrow A\}$ is finite.

□

2 Acceptable Programs

Definitions

The subject of termination of Prolog programs has been studied in several articles (see Apt and Pedreschi [AP90] for a short overview). Our approach to termination of general Prolog programs is based on a generalization of the approach of Apt and Pedreschi [AP90]. We begin by recalling the relevant notions.

A *level mapping* for a positive program P (see Bezem [Bez89] and Cavedon [Cav89]) is a function $|\ | : B_P \rightarrow N$ from ground atoms to natural numbers. For $A \in B_P$, $|A|$ is the level of A.

Definition 2.1 Let P be a positive program, $|\ |$ a level mapping for P and I a (not necessarily Herbrand) model of P. P is called *acceptable with respect to* $|\ |$ *and* I if for every clause $A \leftarrow B_1, \ldots, B_n$ in $ground(P)$

$$|A| > |B_i| \text{ for } i \in [1, \bar{n}],$$

where

$$\bar{n} = \min(\{n\} \cup \{i \in [1,n] \mid I \not\models B_i\}).$$

Alternatively, we may define $\bar{n}$ by

$$\bar{n} = \begin{cases} n & \text{if } I \models B_1 \wedge \ldots \wedge B_n, \\ i & \text{if } I \models B_1 \wedge \ldots \wedge B_{i-1} \text{ and } I \not\models B_1 \wedge \cdots \wedge B_i. \end{cases}$$

P is called *acceptable* if it is acceptable with respect to some level mapping and a model of P. □

Our aim is to generalize the above concept of acceptability to general Prolog programs. First , we extend in a natural way a level mapping to a mapping from ground literals to natural numbers by putting $|\neg A| = |A|$. Next, given a program P, we define its subset P^-. In P^- we collect the definitions of the negated relations and relations on which these relations depend. More precisely, we define P^- as follows.

Definition 2.2 Let P be a program and p, q relations.

(i) We say that p *refers to* q iff there is a clause in P that uses p in its head and q in its body.

(ii) We say that p *depends on* q iff (p, q) is in the reflexive, transitive closure of the relation *refers to.*

□

Of course, not every relation needs to refer to itself, but by reflexivity every relation depends on itself.

Definition 2.3 Let P be a program. Denote by Neg_P the set of relations in P which occur in a negative literal in a body of a clause from P and by Neg_P^* the set of relations in P on which the relations in Neg_P depend on. We define P^- to be the set of clauses in P in whose head a relation from Neg_P^* occurs. □

We can now introduce the desired generalization of the notion of acceptability.

Definition 2.4 Let P be a program, $|\ |$ a level mapping for P and I a model of P whose restriction to the relations from Neg_P^* is a model of $comp(P^-)$. P is called *acceptable with respect to* $|\ |$ *and* I if for every clause $A \leftarrow L_1, \ldots, L_n$ in $ground(P)$

$$|A| > |L_i| \text{ for } i \in [1, \bar{n}],$$

where

$$\bar{n} = \min(\{n\} \cup \{i \in [1,n] \mid I \not\models L_i\}).$$

P is called *acceptable* if it is acceptable with respect to some level mapping and a model of P whose restriction to the relations from Neg_P^* is a model of $comp(P^-)$. □

Note that for a positive program P we have $Neg_P^* = \emptyset$, so P^- is empty and the above definition coincides with the definition of acceptability for positive programs.

The concept of an acceptable program also generalizes that of an acyclic program studied in Cavedon [Cav89] and Apt and Bezem [AB90].

Definition 2.5 Let P be a program, $|\ |$ a level mapping for P. P is called *acyclic with respect to* $|\ |$ if for every clause $A \leftarrow L_1, \ldots, L_n$ in $ground(P)$

$$|A| > |L_i| \text{ for } i \in [1, n].$$

P is called *acyclic* if it is acyclic with respect to some level mapping. □

Lemma 2.6 *Every acyclic program is acceptable.*

Proof. Let P be acyclic w.r.t. some level mapping $|\ |$. By Theorem 4.1 of Apt and Bezem [AB90] $comp(P)$ has a unique Herbrand model, M_P. Then P is acceptable w.r.t. $|\ |$ and M_P. □

Apt and Bezem [AB90] proved among others that all $SLDNF$-derivations of an acyclic program starting in a ground goal are finite. This implies that all acyclic programs are left terminating, so the concept of acyclicity is of obvious importance when studying termination of Prolog programs. Indeed, in Apt and Bezem [AB90] the usefulness of this concept was demonstrated by proving termination of a program which formalizes the Yale Shooting problem of Hanks and McDermott [HM87]. However, as we shall see in the final section of this paper, there exist natural left terminating programs which are not acyclic. Thus the concept of acyclicity is of limited applicability when considering Prolog programs.

Multiset ordering

In our considerations below we use the multiset ordering. A *multiset*, sometimes called *bag*, is an unordered sequence. Given a (non-reflexive) ordering $<$ on a set W, the *multiset ordering over* $(W, <)$ is an ordering on finite multisets of the set W. It is defined as the transitive closure of the relation in which X is smaller than Y if X can be obtained from Y by replacing an element a of Y by a finite (possibly empty) multiset each of whose elements is smaller than a in the ordering $<$.

In symbols, first we define the relation $\prec$ by

$$X \prec Y \text{ iff } X = Y - \{a\} \cup Z \text{ for some } Z \text{ such that } b < a \text{ for } b \in Z,$$

where X, Y, Z are finite multisets of elements of W, and then define the multiset ordering over $(W, <)$ as the transitive closure of the relation $\prec$.

It is well-known (see e.g. Dershowitz [Der87]) that multiset ordering over a well-founded ordering is again well-founded. Thus it can be iterated while maintaining well-foundedness. What we need here is, as in Apt and Pedreschi [AP90], two fold iteration. We start with the set of natural numbers N ordered by $<$ and apply the multiset ordering twice. We call the first iteration multiset ordering and the second *double multiset ordering.* Both are well-founded. The double multiset ordering is defined on the finite *multisets* of finite multisets of natural numbers, but we shall use it only on the finite *sets* of finite multisets of natural numbers. The following simple lemma (see Apt and Pedreschi [AP90]) will be of help when using the double multiset ordering.

Lemma 2.7 *Let X and Y be two finite sets of finite multisets of natural numbers. Suppose that*

$$\forall x \in X\, \exists y \in Y\ (y\ \textit{majorizes}\ x),$$

where y majorizes x means that x is smaller than y in the multiset ordering.
Then X is smaller than Y in the double multiset ordering.

Proof. We call an element $y \in Y$ *majorizing* if it majorizes some $x \in X$. X can be obtained from Y by first replacing each majorizing $y \in Y$ by the multiset M_y of elements of X it majorizes and then removing from Y the non-majorizing elements. This proves the claim. □

Below we use the notation bag $(a_1, \ldots, a_n)$ to denote the multiset consisting of the unordered sequence $a_1, \ldots, a_n$.

Boundedness

Another important concept is that of boundedness, originally introduced in Bezem [Bez89]. It allows us to identify goals from which no divergence can arise. Recall that an atom A is called *bounded* w.r.t. a level mapping $|\ |$ if $|\ |$ is bounded on the set $[A]$ of ground instances of A. If A is bounded, then $|[A]|$ denotes the maximum that $|\ |$ takes on $[A]$. Note that every ground atom is bounded.

Our concept of a bounded general goal directly generalizes that of a bounded goal given in Apt and Pedreschi [AP90].

Definition 2.8 Let P be a program, $|\ |$ a level mapping for P, I model of P whose restriction to the relations from Neg_P^* is a model of $comp(P^-)$ and $k \geq 0$.

(i) With each ground general goal $G = \leftarrow L_1, \ldots, L_n$ we associate a finite multiset $|G|_I$ of natural numbers defined by

$$|G|_I \ = \ \text{bag}\,(|L_1|, \ldots, |L_{\bar{n}}|),$$

where

$$\bar{n} = \min(\{n\} \cup \{i \in [1,n] \mid I \not\models L_i\}).$$

(ii) With each general goal G we associate a set of multisets $|[G]|_I$ defined by

$$|[G]|_I = \{|G'|_I \mid G' \text{ is a ground instance of } G\}.$$

(iii) A general goal G is called *bounded by k* w.r.t. $|\ |$ and I if $k \geq \ell$ for $\ell \in \cup|[G]|_I$, where $\cup|[G]|_I$ stands for the set-theoretic union of the elements of $|[G]|_I$.

(iv) A general goal is called *bounded* w.r.t. $|\ |$ and I if it is bounded by some $k \geq 0$ w.r.t. $|\ |$ and I.

□

It is useful to note the following.

Lemma 2.9 *Let P be a program, $|\ |$ a level mapping for P and I a model of P whose restriction to the relations from Neg_P^* is a model of comp(P^-). A general goal G is bounded w.r.t. $|\ |$ and I iff the set $|[G]|_I$ is finite.*

Proof. Consider a general goal G that is bounded by some k. Suppose that G has n atoms. Then each element of $|[G]|_I$ is a multiset of at most n numbers selected from $[0, k]$. The number of such multisets is finite.

The other implication is obvious. □

The following lemma is an analogue of Lemma 3.7 of Apt and Pedreschi [AP90]. Recall that a goal is called *positive* if it contains only positive literals.

Lemma 2.10 *Let P be a program that is acceptable w.r.t. a level mapping $|\ |$ and an interpretation I. Let G be a goal which is a descendant of a positive goal and which is bounded (w.r.t. $|\ |$ and I) and let H be an LDNF-resolvent of G from P. Then*

(i) H is bounded,

(ii) $|[H]|_I$ is smaller than $|[G]|_I$ in the double multiset ordering.

Proof. The proof is analogous to the proof of Lemma 3.7 of Apt and Pedreschi [AP90]. Due to the presence of negative literals we only have to consider one additional case.

Let $G = \leftarrow L_1, \ldots, L_n\ (n \geq 1)$. For some literals $M_1, \ldots, M_k\ (k \geq 0)$ and a substitution θ we have $H = \leftarrow (M_1, \ldots, M_k, L_2, \ldots, L_n)\theta$.

First we show that for every ground instance H_0 of H there exists a ground instance G' of G such that $|H_0|_I$ is smaller that $|G'|_I$ in the multiset ordering.

Case 1 H is obtained from G by the negation as failure rule.
Then L_1 is a ground negative literal, say $L_1 = \neg A$, and $H = \leftarrow L_2, \ldots, L_n$, i.e. $k = 0$ and $\theta = \epsilon$ (ϵ stands for the empty substitution).

Denote by T the finitely failed $LDNF$-tree for $P \cup \{\leftarrow A\}$. By the definition of Neg_P and the fact that G is a descendant of a positive goal, the relation occurring in A is in Neg_P. Thus all relations which occur in the goals of the tree T are elements of Neg_P^*. So T is in fact a finitely failed $LDNF$-tree for $P^- \cup \{\leftarrow A\}$. By the soundness of the $SLDNF$-resolution, $comp(P^-) \models \neg A$, so $I \models L_1$.

Let H_0 be a ground instance of H. For some substitution δ

$$H_0 = \leftarrow L'_2, \ldots, L'_n,$$

where L'_i denotes $L_i\delta$. Thus

$$G' = \leftarrow L_1, L'_2, \ldots, L'_n,$$

is a ground instance of G. Then

$$|H_0|_I = \text{bag}\,(|L'_2|, \ldots, |L'_n|)$$

where

$$\bar{n} = \min(\{n\} \cup \{i \in [2,n] \mid I \not\models L'_i\}).$$

and, since $I \models L_1$,

$$|G'|_I = \text{bag}\,(|L_1|,\ |L'_2|, \ldots, |L'_{\bar{n}}|).$$

This shows that $|H_0|_I$ is smaller than $|G'|_I$ in the multiset ordering. □

Case 2 H is obtained from G by the proper resolution step.
Then L_1 is a positive literal, so for some atom A, $C = A \leftarrow M_1, \ldots, M_k$ is an input clause of P and θ is an mgu of A and L_1. Let H_0 be a ground instance of H. For some substitution δ

$$H_0 = \leftarrow M'_1, \ldots, M'_k, L'_2, \ldots, L'_n,$$

where for brevity for any atom, clause or goal M, M' denotes $M\theta\delta$. Note that

$$C' = L'_1 \leftarrow M'_1, \ldots, M'_k$$

and

$$G' = \leftarrow L'_1, \ldots, L'_n,$$

since $A' = L'_1$ as $A\theta = L_1\theta$.

Subcase 1 For $i \in [1,k]$ $I \models M'_i$.
Then

$$|H_0|_I = \text{bag}\,(|M'_1|, \ldots, |M'_k|,\ |L'_2|, \ldots, |L'_{\bar{n}}|)$$

where

$$\bar{n} = \min(\{n\} \cup \{i \in [2,n] \mid I \not\models L'_i\}).$$

Additionally $I \models L'_1$ because I is a model of P and a fortiori a model of the clause C'. Thus

$$|G'|_I = \text{bag}\,(|L'_1|,\ |L'_2|, \ldots, |L'_{\bar{n}}|).$$

This means that $|H_0|_I$ is obtained from $|G'|_I$ by replacing $|L'_1|$ by $|M'_1|, \ldots, |M'_k|$. But by the definition of acceptability

$$|M'_i| < |L'_1|$$

for $i \in [1,k]$, so $|H_0|_I$ is smaller than $|G'|_I$ in the multiset ordering. □

Subcase 2 For some $i \in [1,k]$ $I \not\models M'_i$.
Then

$$|H_0|_I = \text{bag}(|M'_1|, \ldots, |M'_{\bar{k}}|)$$

where

$$\bar{k} = \min(\{i \in [1,k] \mid I \not\models M'_i\}).$$

Also, by the definition of acceptability

$$|M'_i| < |L'_1|$$

for $i \in [1, \bar{k}]$, so $|H_0|_I$ is smaller than $|G'|_I$ in the multiset ordering. □
□

The statement we just proved implies claim (i) since G is bounded. By Lemma 2.9 $|[H]|_I$ is finite and claim (ii) now follows by Lemma 2.7. □

Corollary 2.11 *Let P be an acceptable program and G a bounded positive goal. Then all LDNF-derivations of $P \cup \{G\}$ are finite.*

Proof. The double multiset ordering is well-founded. □

Corollary 2.12 *Every acceptable program is left terminating.*

Proof. By the fact that every ground goal is bounded, Corollary 2.11 and Lemma 1.2. □

Thus to prove that a program is left terminating it suffices to show that it is acceptable.

To apply Corollaries 2.11 and 2.12 we need a method for verifying that an interpretation is a model of $comp(P^-)$. In the case of Herbrand interpretations this task becomes much simpler thanks to the following theorem due to Apt, Blair and Walker [ABW88]. Here an interpretation is *supported* if for all ground atoms A, $I \models A$ implies that for some clause $A \leftarrow L_1, \ldots, L_n$ in $ground(P)$ we have $I \models L_1 \wedge \ldots \wedge L_n$.

Theorem 2.13 *A Herbrand interpretation I is a model of $comp(P)$ iff it is a supported model of P.* □

3 Acceptability versus Left Termination

The converse of Corollary 2.12 does not hold. This is in contrast to the case of positive programs. Below we say that an $LDNF$-derivation *flounders* if there occurs in it or in any of its subsidiary $LDNF$-trees a goal with the first literal being non-ground and negative. An $LDNF$-tree is called *non-floundering* if none of its branches flounders.

Example 3.1 Consider the program P which consists of only one clause: $p(0) \leftarrow \neg p(X)$. Then the only $LDNF$-derivation of $P \cup \{ \leftarrow p(0)\}$ flounders, so it is finite. By the definition of $SLDNF$-resolution the only $LDNF$-derivation of $P \cup \{ \leftarrow \neg p(0)\}$ flounders, as well. Thus P is left terminating, since the only ground goals are of the form $G = \leftarrow L_1, \ldots, L_n$ $(n \geq 1)$ where each L_i is either $p(0)$ or $\neg p(0)$. On the other hand P is not acceptable since $p(0) \leftarrow \neg p(0)$ is in $ground(P)$ and by definition for any level mapping $|p(0)| = |\neg p(0)|$. □

The above example exploits the fact that $SLDNF$-derivations may terminate by floundering. We now show that in the absence of floundering Corollary 2.12 can be reversed. We proceed analogously to the case of positive programs and study the size of finite $LDNF$-trees. We need the following lemma, where $nodes_P(G)$ for a program P and a goal G denotes the total number of nodes in the $LDNF$-tree for $P \cup \{G\}$ and in all the subsidiary $LDNF$-trees for $P \cup \{G\}$.

Lemma 3.2 *Let P be a program and G a goal such that the $LDNF$-tree for $P \cup \{G\}$ is finite and non-floundering. Then*

(i) for all substitutions θ, the $LDNF$-tree for $P \cup \{G\theta\}$ is finite and non-floundering and $nodes_P(G\theta) \leq nodes_P(G)$,

(ii) for all prefixes H of G, the $LDNF$-tree for $P \cup \{H\}$ is finite and non-floundering and $nodes_P(H) \leq nodes_P(G)$,

(iii) for all non-root nodes H in the $LDNF$-tree for $P \cup \{G\}$, $nodes_P(H) < nodes_P(G)$.

Proof.
(i) The proof proceeds by structural induction on the $LDNF$-tree T for $P \cup \{G\}$.

The Base Case. Then T is formed by the only node G. The following three subcases arise.

Subcase 1 $G = \square$. Then $G = G\theta$, and the claim trivially holds.
Subcase 2 $G = \;\leftarrow A, L_2, \ldots, L_k$. Then A does not unify with the head of any clause in P and neither $A\theta$ does. As a consequence, the goal $G\theta$ also immediately fails, and the $LDNF$-tree T for $P \cup \{G\theta\}$ is formed by the only node $G\theta$.
Subcase 3 $G = \;\leftarrow \neg A, L_2, \ldots, L_k$. By the fact that T has no floundering derivation, A is ground. The goal G immediately fails, so by the definition of the $LDNF$-resolution there is an $LDNF$-refutation of $P \cup \{\leftarrow A\}$. Then $G\theta$ also immediately fails as $A = A\theta$. Hence the $LDNF$-tree T for $P \cup \{G\theta\}$ is formed by the only node $G\theta$. By definition

$$nodes_P(G\theta) = 1 + nodes_P(\leftarrow A\theta) = 1 + nodes_P(\leftarrow A) = nodes_P(G).$$

The Induction Case. Two subcases arise here.

Subcase 1 $G = \;\leftarrow A, L_2, \ldots, L_k$. Assume that $H_1, \ldots, H_m$ are the resolvents of G from P. Consider $G\theta = \;\leftarrow (A, L_2, \ldots, L_k)\theta$, and let $H_1', \ldots, H_l'$ be the resolvents of $G\theta$ from P. Clearly, for all i in $[1, l]$ there exist j in $[1, m]$ and a substitution δ such that $H_i' = H_j\delta$. By the induction hypothesis, $nodes_P(H_i') \leq nodes_P(H_j)$. Hence:

$$\begin{aligned}&nodes_P(G\theta) = 1 + nodes_P(H_1') + \ldots + nodes_P(H_l') \leq \\ &1 + nodes_P(H_1) + \ldots + nodes_P(H_m) = nodes_P(G).\end{aligned}$$

Moreover, the $LDNF$-tree for $P \cup \{G\theta\}$ is finite and non-floundering and by the induction hypothesis the $LDNF$-trees for the resolvents of $G\theta$ are finite and non-floundering.
Subcase 2 $G = \;\leftarrow \neg A, L_2, \ldots, L_k$. By the fact that T has no floundering derivation, A is ground. The fact that G is not a terminal node in T implies that there exists an $LDNF$-refutation of $P \cup \{\leftarrow \neg A\}$, i.e. the $LDNF$-tree for $P \cup \{\leftarrow A\}$ is finitely failed. Then G

has only one resolvent, namely $\leftarrow L_2,\ldots,L_k$. Moreover, $G\theta = \leftarrow \neg A,(L_2,\ldots,L_k)\theta$, since A is ground, so $\leftarrow (L_2,\ldots,L_k)\theta$ is the only resolvent of $G\theta$. By the induction hypothesis, $nodes_P(\leftarrow (L_2,\ldots,L_k)\theta) \le nodes_P(\leftarrow L_2,\ldots,L_k)$. Hence:

$$nodes_P(G\theta) = 1 + nodes_P(\leftarrow A) + nodes_P(\leftarrow (L_2,\ldots,L_k)\theta) \le$$
$$1 + nodes_P(\leftarrow A) + nodes_P(\leftarrow L_2,\ldots,L_k) = nodes_P(G).$$

Moreover, the *LDNF*-tree for $P \cup \{G\theta\}$ is finite and non-floundering, since by the induction hypothesis the *LDNF*-tree for the resolvent of $G\theta$ is finite and non-floundering.

(ii) Consider a prefix $H = \leftarrow L_1,\ldots,L_k$ of $G = \leftarrow L_1,\ldots,L_n$ $(n \ge k)$. By an appropriate renaming of variables (formally justified by a straightforward extension to the *LDNF*-resolution of the Variant Lemma 2.8 in Apt [Apt90]) we can assume that all input clauses used in the *LDNF*-tree for $P \cup \{H\}$ have no variables in common with G. We can now transform the *LDNF*-tree for $P \cup \{H\}$ into an initial subtree of the *LDNF*-tree for $P \cup \{G\}$ by replacing in it a node $\leftarrow M_1,\ldots,M_l$ by $\leftarrow M_1,\ldots,M_l, L_{k+1}\theta,\ldots,L_n\theta$, where θ is the composition of the mgu's used on the path from the root H to the node $\leftarrow M_1,\ldots,M_l$. This implies the claim, since every subsidiary *LDNF*-tree for $P \cup \{H\}$ is also a subsidiary *LDNF*-tree for $P \cup \{G\}$.

(iii) Immediate by the definition. □

The following definition will now be useful.

Definition 3.3 We call a program P *non-floundering* if all its *LDNF*-derivations starting in a ground goal are non-floundering.

Theorem 3.4 *Let P be a left terminating, non-floundering program. Then for some level mapping $|\ |$ and a model I of $comp(P)$*

(i) P is acceptable w.r.t. $|\ |$ and I,

(ii) for every goal G, G is bounded w.r.t. $|\ |$ and I iff all LDNF-derivations of $P \cup \{G\}$ are finite.

Proof. Define the level mapping by putting for $A \in B_P$

$$|A| = nodes_P\,(\leftarrow A).$$

Since P is left terminating, this level mapping is well defined. Note that by definition, for $A \in B_P$

$$nodes_P(\leftarrow \neg A) > nodes_P(\leftarrow A) = |A| = |\neg A|,$$

so

$$nodes_P(\leftarrow \neg A) \ge |\neg A|.$$

Next, choose

$$I = \{A \in B_P \mid \text{there is an } LDNF\text{-refutation of } P \cup \{\leftarrow A\}\}.$$

Let us show that I is a model of *comp*(P). To this end, we use Theorem 2.13 and show that I is a supported model of P.

To establish that I is a model of P, assume by contradiction that some ground instance $A \leftarrow L'_1, \ldots, L'_n$ of a clause C from P is false in I. Then $I \models L'_1 \wedge \ldots \wedge L'_n$ and $I \not\models A$. Since P is left terminating and non-floundering, $I \not\models A$ implies that the *LDNF*-tree for $P \cup \{\leftarrow A\}$ is finitely failed and non-floundering.

For some ground substitution γ, $A = B\gamma$ where B is the head of the clause C. Thus $A\gamma = B\gamma\gamma = B\gamma$, so A and B unify.

Let $\leftarrow L_1, \ldots, L_n$ be the resolvent of $\leftarrow A$ from the clause C. The *LDNF*-tree for $P \cup \{\leftarrow L_1, \ldots, L_n\}$ is also finitely failed and non-floundering. As $L'_1, \ldots, L'_n = (L_1, \ldots, L_n)\theta$ for some substitution θ, we have by Lemma 3.2(i) that the *LDNF*-tree for $P \cup \{\leftarrow L'_1, \ldots, L'_n\}$ is non-floundering. Moreover, it is finitely failed, since a direct consequence of the proof of Lemma 3.2(i) is that the goals present in the *LDNF*-tree for $P \cup \{\leftarrow L'_1, \ldots, L'_n\}$ are all instances of the goals present in the *LDNF*-tree for $P \cup \{\leftarrow L_1, \ldots, L_n\}$. But the fact that the *LDNF*-tree for $P \cup \{\leftarrow L'_1, \ldots, L'_n\}$ is finitely failed and non-floundering contradicts the hypothesis that $I \models L'_1 \wedge \ldots \wedge L'_n$.

To establish that I is a supported interpretation of P, consider $A \in B_P$ such that $I \models A$, and let C be the first input clause used in the leftmost *LDNF*-refutation of $P \cup \{\leftarrow A\}$. Let $\leftarrow L_1, \ldots, L_n$ be the resolvent of $\leftarrow A$ from the clause C. Clearly, an *LDNF*-refutation for $P \cup \{\leftarrow L_1, \ldots, L_n\}$, with a computed answer substitution θ, can be extracted from the *LDNF*-refutation of $P \cup \{\leftarrow A\}$. Let $L'_1, \ldots, L'_n$ be a ground instance of $(L_1, \ldots, L_n)\theta$. By a straightforward generalization of Lemma 3.20 in [Apt90] to the *LDNF*-resolution there exists an *LDNF*-refutation for $P \cup \{\leftarrow L'_1, \ldots, L'_n\}$. We conclude that $I \models L'_1 \wedge \ldots \wedge L'_n$. This establishes that I is a supported interpretation of P.

We are now in the position to prove (i) and (ii). First we prove one implication of (ii).

(ii1) Consider a goal G such that all *LDNF*-derivations of $P \cup \{G\}$ are finite. We prove that G is bounded by $nodes_P(G)$ w.r.t. $|\;|$ and I.

To this end take $\ell \in \cup|[G]|_I$. For some ground instance $\leftarrow L_1, \ldots, L_n$ of G and $i \in [1, \bar{n}]$, where

$$\bar{n} = \min(\{n\} \cup \{i \in [1,n] \mid I \not\models L_i\}),$$

we have $\ell = |L_i|$. We now calculate

$$\begin{array}{ll}
 & nodes_P(G) \\
\geq & \quad \{\text{Lemma 3.2 (i)}\} \\
 & nodes_P(\leftarrow L_1, \ldots, L_n) \\
\geq & \quad \{\text{Lemma 3.2 (ii)}\} \\
 & nodes_P(\leftarrow L_1, \ldots, L_{\bar{n}}) \\
\geq & \quad \{\text{Lemma 3.2 (iii), noting that for } j \in [1, \bar{n}-1] \\
 & \quad \text{there is an } LDNF\text{-refutation of } P \cup \{\leftarrow L_1, \ldots, L_j\}\} \\
 & nodes_P(\leftarrow L_i, \ldots, L_{\bar{n}})
\end{array}$$

$$\begin{array}{ll}
\geq & \{\text{Lemma 3.2 (ii)}\} \\
& nodes_P\,(\leftarrow L_i) \\
\geq & \{\text{definition of } |\ |,\ L_i \text{ is ground}\} \\
& |L_i| \\
= & \ell.
\end{array}$$

(i) We now prove that P is acceptable w.r.t. $|\ |$ and I. We showed that I is a model of $comp(P)$, so the restriction of I to the relations in Neg_P^* is trivially a model of $comp(P^-)$. To complete the proof, take a clause $A \leftarrow L_1, \ldots, L_n$ in P and its ground instance $A\theta \leftarrow L_1\theta, \ldots, L_n\theta$. We need to show that

$$|A\theta| > |L_i\theta| \text{ for } i \in [1, \bar{n}],$$

where

$$\bar{n} = \min(\{n\} \cup \{i \in [1, n] \mid I \not\models L_i\theta\}).$$

We have $A\theta\theta \equiv A\theta$, so $A\theta$ and A unify. Let $\mu = \text{mgu}(A\theta, A)$. Then $\theta = \mu\delta$ for some δ. By the definition of $LDNF$-resolution, $\leftarrow L_1\mu, \ldots, L_n\mu$ is an $LDNF$-resolvent of $\leftarrow A\theta$.

Then for $i \in [1, \bar{n}]$

$$\begin{array}{ll}
& |A\theta| \\
= & \{\text{definition of } |\ |\} \\
& nodes_P\,(\leftarrow A\theta) \\
> & \{\text{Lemma 3.2(iii)},\ \leftarrow L_1\mu, \ldots, L_n\mu \text{ is a resolvent of } \leftarrow A\theta\} \\
& nodes_P\,(\leftarrow L_1\mu, \ldots, L_n\mu) \\
\geq & \{\text{part (ii1), noting that } L_i\theta \in \cup|[\leftarrow L_1\mu, \ldots, L_n\mu]|_I\} \\
& |L_i\theta|.
\end{array}$$

(ii2) Consider a goal G which is bounded w.r.t. $|\ |$ and I. Then by (i) and Corollary 2.10 all $LDNF$-derivations of $P \cup \{G\}$ are finite. □

Corollary 3.5 *A non-floundering program is left terminating iff it is acceptable.*

Proof. By Corollary 2.12 and Theorem 3.4. □

4 Semantic Considerations

In this section we study semantics of acceptable programs. We show here that various ways of defining their semantics coincide.

We recall first the relevant definitions and results. We use below Fitting's approach to the semantics of general programs. Fitting [Fit85] uses a 3-valued logic based on a logic due to Kleene [Kle52]. In Kleene's logic there are three truth values: **t** for true, **f** for false and **u** for undefined.

A Herbrand interpretation for this logic (called a 3-*valued* Herbrand interpretation) is defined as a pair (T, F) of disjoint sets of ground atoms. Given such an interpretation $I = (T, F)$ a ground atom A is true in I if $A \in T$, false in I if $A \in F$ and undefined otherwise; $\neg A$ is true in I if A is false in I and $\neg A$ is false in I if A is true in I.

Every binary connective takes the value **t** or **f** if it takes that value in 2-valued logic for all possible substitutions of **u**'s by **t** or **f**; otherwise it takes value **u**.

Given a formula ϕ and a 3-valued Herbrand interpretation I, we write ϕ is $true_3$ in I (respectively ϕ is $false_3$ in I) to denote the fact that ϕ is true in I (respectively that ϕ is false in I) in the above defined sense.

Given $I = (T, F)$ we denote T by I^+ and F by I^-. Thus $I = (I^+, I^-)$. If $I^+ \cup I^- = B_P$, we call I a *total* 3-valued Herbrand interpretation for the program P.

Every (2-valued) Herbrand interpretation I for a program P determines a total 3-valued Herbrand interpretation $(I, B_P - I)$ for P. This allows us to identify every 2-valued Herbrand interpretation I for a program P with its 3-valued counterpart $(I, B_P - I)$. For uniformity, given a (2-valued) Herbrand interpretation I we write ϕ is $true_2$ in I instead of $I \models \phi$ and ϕ is $false_2$ in I instead of $I \not\models \phi$. The following proposition relates truth in 3- and 2-valued intepretations and will be useful later.

Proposition 4.1 *Let I be a 3-valued interpretation and L a literal. Then*

(i) L is $true_3$ in I implies L is $true_2$ in I^+,

(ii) L is $true_2$ in I^+ implies L is not $false_3$ in I, i.e. L is either $true_3$ or undefined in I.

Proof.
(i) If $L = A$, L is $true_3$ in I implies $A \in I^+$, hence A is $true_2$ in I^+. If $L = \neg A$, $\neg A$ is $true_3$ in I implies $A \in I^-$, which implies $A \notin I^+$. Hence $\neg A$ is $true_2$ in I^+.
(ii) If $L = A$, L is $true_2$ in I^+ implies $A \in I^+$, hence A is $true_3$ in I. If $L = \neg A$, $\neg A$ is $true_2$ in I^+ implies $A \notin I^+$. Hence $\neg A$ is either $true_3$ or undefined in I. □

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

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

and with the least element $(\emptyset, \emptyset)$. Note that in this ordering every total 3-valued Herbrand interpretation is $\subseteq$-maximal. Intuitively, $I \subseteq J$ if J decides both truth and falsity for more atoms than I does.

Following Fitting [Fit85], given a program P we define an operator Φ_P on the complete partial ordering of 3-valued Herbrand interpretations for P as follows:

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

where

$$T = \{A \mid \text{for some } A \leftarrow L_1, \ldots, L_k \text{ in } \mathit{ground}(P),\ L_1 \wedge \ldots \wedge L_k \text{ is } \mathit{true}_3 \text{ in } I\},$$
$$F = \{A \mid \text{for all } A \leftarrow L_1, \ldots, L_k \text{ in } \mathit{ground}(P),\ L_1 \wedge \ldots \wedge L_k \text{ is } \mathit{false}_3 \text{ in } I\}.$$

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 usual immediate consequence operator T_P to the case of 3-valued logic. Φ_P is easily seen to be monotonic.

The *upward ordinal powers* of Φ_P, denoted by $\Phi_P \uparrow \alpha$, are defined in the usual way starting the iteration at the $\subseteq$-least 3-valued Herbrand interpretation, $(\emptyset, \emptyset)$. In particular

$$\Phi_P \uparrow \omega = \bigcup_{n<\omega} \Phi_P \uparrow n.$$

Before studying semantics of acceptable programs we prove a number of auxiliary results about the operators T_P and Φ_P. The following lemma relates these two operators.

Lemma 4.2 *Let I be a 3-valued interpretation and P a program. Then*

$$\Phi_P(I)^+ \subseteq T_P(I^+) \subseteq B_P - \Phi_P(I)^-.$$

Moreover, if I is total then $\Phi_P(I)^+ = T_P(I^+) = B_P - \Phi_P(I)^-$.

Proof. By definition of T_P and Φ_P we obtain:

$A \in \Phi_P(I)^+$ iff for some $A \leftarrow L_1, \ldots, L_k$ in $\mathit{ground}(P)$ $L_1 \wedge \ldots \wedge L_k$ is true_3 in I,
$A \in T_P(I^+)$ iff for some $A \leftarrow L_1, \ldots, L_k$ in $\mathit{ground}(P)$ $L_1 \wedge \ldots \wedge L_k$ is true_2 in I^+,
$A \in B_P - \Phi_P(I)^-$ iff for some $A \leftarrow L_1, \ldots, L_k$ in $\mathit{ground}(P)$ $L_1 \wedge \ldots \wedge L_k$ is not false_3 in I.

Hence, the implication $A \in \Phi_P(I)^+ \Rightarrow A \in T_P(I^+)$ (respectively $A \in T_P(I^+) \Rightarrow A \in B_P - \Phi_P(I)^-$) directly follows from Proposition 4.1(i) (respectively Proposition 4.1(ii)).

If I is total, then $L_1 \wedge \ldots \wedge L_k$ is true_3 in I iff $L_1 \wedge \ldots \wedge L_k$ is true_2 in I^+ iff $L_1 \wedge \ldots \wedge L_k$ is not false_3 in I. □

The following corollaries relate the fixpoints of the operators T_P and Φ_P.

Corollary 4.3 *Let $I = (I^+, B_P - I^+)$ be a total 3-valued interpretation and P a program. Then I^+ is a fixpoint of T_P if and only if I is a fixpoint of Φ_P.*

Proof.
($\Rightarrow$) Assume $I^+ = T_P(I^+)$. By Lemma 4.2 we have $\Phi_P(I)^+ = T_P(I^+) = B_P - \Phi_P(I)^-$. Hence $I^+ = \Phi_P(I)^+$ and $I^- = B_P - I^+ = \Phi_P(I)^-$, i.e. $I = \Phi_P(I)$.
($\Leftarrow$) Assume $I = \Phi_P(I)$. Then by Lemma 4.2 we have

$$I^+ = \Phi_P(I)^+ \subseteq T_P(I^+) \subseteq B_P - \Phi_P(I)^- = B_P - I^- = I^+.$$

Hence I^+ is a fixpoint of T_P. □

Corollary 4.4 *If Φ_P has exactly one fixpoint I and I is total, then I^+ is the unique fixpoint of T_P.*

Proof. By Corollary 4.3. □

The fixpoints of the operator T_P are of interest for us because of the following result of Apt, Blair and Walker [ABW88].

Theorem 4.5 *A Herbrand interpretation I is a model of comp(P) iff it is a fixpoint of T_P.* □

Corollary 4.6 *If I is a Herbrand model of comp(P) then $\Phi_P \uparrow \omega \subseteq (I, B_P - I)$.*

Proof. Suppose I is a Herbrand model of *comp*(P). Then by Theorem 4.5 I is a fixpoint of T_P, so by Corollary 4.3 $(I, B_P - I)$ is fixpoint of Φ_P. By the monotonicity of Φ_P the least fixpoint of Φ_P, $lfp(\Phi_P)$, exists and $\Phi_P \uparrow \omega \subseteq lfp(\Phi_P)$. But $lfp(\Phi_P) \subseteq (I, B_P - I)$, so $\Phi_P \uparrow \omega \subseteq (I, B_P - I)$.

□

We are now ready to analyze the semantics of acceptable programs.

Theorem 4.7 *Let P be an acceptable program w.r.t. $|\,|$ and I. Then $\Phi_P \uparrow \omega$ is total.*

Proof. To establish that $\Phi_P \uparrow \omega$ is total we prove that, for $n \in \omega$ and $A \in B_P$, $|A| = n$ implies that A is not undefined in $\Phi_P \uparrow (n+1)$, i.e. A is either $true_3$ or $false_3$ in $\Phi_P \uparrow (n+1)$. The proof proceeds by induction on n. Fix $A \in B_P$.

In the base case we have $|A| = 0$ and since P is acceptable, two possibilities arise: (i) there is a unit clause $A \leftarrow$ in *ground*(P) and (ii) there is no clause in *ground*(P) with A as conclusion. In case (i) A is $true_3$ in $\Phi_P \uparrow 1$, and in case (ii) A is $false_3$ in $\Phi_P \uparrow 1$.

In the induction case we have $|A| = n > 0$. Consider the set C_A of the clauses in *ground*(P) with A as conclusion. If C_A is empty then A is $false_3$ in $\Phi_P \uparrow 1$ and, by the monotonicity of Φ_P, it is $false_3$ in $\Phi_P \uparrow (n+1)$. If C_A is non-empty, take a clause $A \leftarrow L_1, \ldots, L_k$ from C_A, and let $\bar{k} = \min(\{k\} \cup \{i \in [1,k] \mid L_i \text{ is } false_2 \text{ in } I\}$. We now prove that $L_1 \wedge \ldots \wedge L_{\bar{k}}$ is not undefined in $\Phi_P \uparrow n$. To this end we consider two subcases.

Subcase 1. $\bar{k} = k$ and L_k is $true_2$ in I. Then, by the acceptability of P, $n = |A| > |L_k|$ for $i \in [1,k]$. By the induction hypothesis L_i is either $true_3$ or $false_3$ in $\Phi_P \uparrow n$, for $i \in [1,k]$.

Subcase 2. $\bar{k} \leq k$ and $L_{\bar{k}}$ is $false_2$ in I. Then $n = |A| > |L_k|$ for $i \in [1,\bar{k}]$. By the induction hypothesis, L_i is either $true_3$ or $false_3$ in $\Phi_P \uparrow n$, for $i \in [1,\bar{k}]$. Moreover, we claim that $L_{\bar{k}}$ is $false_3$ in $\Phi_P \uparrow n$. To establish this point, the following two possibilities have to be taken into account.

Suppose the relation occurring in $L_{\bar{k}}$ is in Neg_P^*. A simple proof by induction on n shows that $\Phi_P \uparrow n$ and $\Phi_{P^-} \uparrow n$ coincide on the relations in Neg_P^*. Thus $L_{\bar{k}}$ is $true_3$ in $\Phi_P \uparrow n$ implies $L_{\bar{k}}$ is $true_3$ in $\Phi_{P^-} \uparrow n$. Hence, by Corollary 4.6 and Proposition 4.1(i), $L_{\bar{k}}$ is $true_2$ in the restriction of I to the relations in Neg_P^* which is a model of *comp*(P^-). This contradicts the fact that $L_{\bar{k}}$ is $false_2$ in I.

If the relation occurring in $L_{\bar{k}}$ is not in Neg_P^*, then $L_{\bar{k}}$ is a positive literal. We show that in this case $L_{\bar{k}}$ is $true_3$ in $\Phi_P \uparrow n$ implies $L_{\bar{k}}$ is $true_2$ in I by induction on the stage i at which $L_{\bar{k}}$ becomes $true_3$ in $\Phi_P \uparrow i$. For $i = 0$ there is nothing to prove. If $L_{\bar{k}}$ becomes $true_3$ in $\Phi_P \uparrow i$, then there is a clause $L_{\bar{k}} \leftarrow M_1, \ldots, M_m$ in $ground(P)$ with $M_1 \wedge \ldots \wedge M_m$ being $true_3$ in $\Phi_P \uparrow (i-1)$. For $j \in [1, m]$, if the relation occurring in M_j is in Neg_P^*, then M_j is $true_3$ in $\Phi_P \uparrow (i-1)$ implies M_j is $true_2$ in I by Corollary 4.6 and Proposition 4.1(i). If the relation occurring in M_j is not in Neg_P^*, then M_j is $true_3$ in $\Phi_P \uparrow (i-1)$ implies M_j is $true_2$ in I by the induction hypothesis. Hence $M_1 \wedge \ldots \wedge M_m$ is $true_2$ in I, which implies $L_{\bar{k}}$ is $true_2$ in I, since I is a model of $L_{\bar{k}} \leftarrow M_1, \ldots, M_m$. This contradicts the fact that $L_{\bar{k}}$ is $false_2$ in I.

In both Subcase 1 and 2, we have that $L_1 \wedge \ldots \wedge L_k$ is not undefined in $\Phi_P \uparrow n$, as it is either $true_3$ or $false_3$ in Subcase 1, and $false_3$ in Subcase 2. As a consequence, A is either $true_3$ or $false_3$ in $\Phi_P \uparrow (n+1)$, which establishes the claim. □

Corollary 4.8 *Let P be an acceptable program. Then $\Phi_P \uparrow \omega$ is the unique fixpoint of Φ_P.*

Proof. We have $\Phi_P \uparrow \omega \subseteq \Phi_P \uparrow (\omega + 1)$, i.e. $\Phi_P \uparrow \omega \subseteq \Phi_P(\Phi_P \uparrow \omega)$. By Theorem 4.7 $\Phi_P \uparrow \omega$ is total, so in fact $\Phi_P \uparrow \omega = \Phi_P(\Phi_P \uparrow \omega)$, i.e. $\Phi_P \uparrow \omega$ is a fixpoint of Φ_P. Moreover, by the monotonicity of Φ_P, every fixpoint of Φ_P of the form $\Phi_P \uparrow \alpha$ is contained in any other fixpoint, so in fact $\Phi_P \uparrow \omega$ is the unique fixpoint of Φ_P. □

The following corollary summarizes the relevant properties of $M_P = \Phi_P \uparrow \omega$.

Corollary 4.9 *Let P be an acceptable program. Then*

(i) M_P is total,

(ii) M_P is the unique fixpoint of Φ_P,

(iii) M_P is the unique 3-valued Herbrand model of comp(P),

(iv) M_P^+ is the unique fixpoint of T_P,

(v) M_P^+ is the unique Herbrand model of comp(P),

(vi) for all ground atoms A such that no LDNF-derivation of $P \cup \{\leftarrow A\}$ flounders,

$$A \in M_P^+ \text{ iff there exists an LDNF-refutation of } P \cup \{\leftarrow A\}.$$

In particular, this equivalence holds for all ground atoms A when P is non-floundering.

Proof.
(i) By Theorem 4.7.
(ii) By Corollary 4.8.
(iii) By (ii) and the result of Fitting [Fit85] stating that a 3-valued Herbrand interpretation is a model of *comp(P)* iff it is a fixpoint of Φ_P.

(iv) By Theorem 4.7 and Corollaries 4.8 and 4.4.
(v) By Theorem 4.5.
(vi) Consider a ground atom A such that no $LDNF$-derivation of $P \cup \{\leftarrow A\}$ flounders. By the soundness of the $SLDNF$-resolution and (v) if there exists an $LDNF$-refutation of $P \cup \{\leftarrow A\}$ then $A \in M_P^+$. To prove the converse implication assume $A \in M_P^+$. By Corollary 2.11 all $LDNF$-derivations of $P \cup \{\leftarrow A\}$ are finite. Suppose by contradiction that none of them is successful. Then the $LDNF$-tree for $P \cup \{\leftarrow A\}$ is non-floundering and finitely failed. By the soundness of the $SLDNF$-resolution and (v), $M_P^+ \models \neg A$, i.e. $A \notin M_P^+$ which is a contradiction.

□

Clause (vi) of the above Corollary can be seen as a completeness result for acceptable programs relating the $LDNF$-resolution to the model M_P^+.

5 Applications

Theorem 3.4 shows that our method of proving termination based on the concepts of acceptability and boundedness is complete for left terminating, non-floundering general Prolog programs. In this section we illustrate its use by proving termination of two simple, well-known programs. None of them can be handled within the framework of Apt and Bezem [AB90].

A GAME Program

Suppose that $\mathcal{G}$ is an acyclic finite graph. Consider the following program GAME:

```
win(X) ← move(X,Y), ¬ win(Y).
move(a,b) ←        for (a,b) ∈ 𝒢.
```

Lemma 5.1 GAME *is not acyclic.*

Proof. For any ground instance $win(a) \leftarrow move(a,a), \neg win(a)$ of the first clause and a level mapping $|\ |$ we have $|win(a)| = |\neg win(a)|$. □

We now proceed to show that GAME is acceptable. Since $\mathcal{G}$ is acyclic and finite, there exists a function f from the elements of its domain to natural numbers such that for $a \in dom(\mathcal{G})$

$$f(a) = \begin{cases} 0 & \text{if for no } b,\ (a,b) \in \mathcal{G} \\ 1 + max\ \{f(b) \mid (a,b) \in \mathcal{G}\} & \text{otherwise.} \end{cases}$$

We define appropriate level mapping by putting for all $(a,b) \in dom(\mathcal{G})$

$$|move(a,b)| = f(a)$$

and for $a \in dom(\mathcal{G})$

$$|win(a)| = f(a) + 1.$$

Next, since $\mathcal{G}$ is acyclic and finite, there exists a function g from the elements of its domain to $\{0,1\}$ such that for $a \in dom(\mathcal{G})$

$$g(a) = \begin{cases} 0 & \text{if for no } b,\ (a,b) \in \mathcal{G} \\ 1 - min\ \{g(b) \mid (a,b) \in \mathcal{G}\} & \text{otherwise.} \end{cases}$$

Let

$$\begin{array}{rcl} I & = & \{move(a,b) \mid\ (a,b) \in \mathcal{G}\} \\ & \cup & \{win(a) \mid\ g(a) = 1\}. \end{array}$$

Lemma 5.2 *I is a model of comp(*`GAME`*).*

Proof. The following two statements hold.
(a) I is a model of `GAME`.
Indeed, consider a ground instance

$$win(a) \leftarrow move(a,b), \neg win(b)$$

of the first clause of `GAME` and suppose that

$$I \models move(a,b)\ \land\ \neg win(b).$$

Then $(a,b) \in \mathcal{G}$ and $g(b) = 0$, so $g(a) = 1$ and consequently

$$I \models win(a).$$

Additionally, I is a model for all `move` clauses.

(b) I is a supported interpretation of `GAME`.
Indeed, consider an atom $win(a) \in I$. Then $g(a) = 1$, so for some $b \in \mathcal{G}$ we have $(a,b) \in \mathcal{G}$ and $g(b) = 0$. We conclude that

$$I \models move(a,b)\ \land\ \neg win(b).$$

By Theorem 2.13 we conclude that I is a model of *comp*(`GAME`). □

We can now prove the desired result.

Theorem 5.3 `GAME` *is acceptable w.r.t.* $|\ |$ *and* I.

Proof. For a program P every model of $comp(P)$ is also a model of P, thus I is a model of `GAME`. Moreover, $\text{GAME}^- = \text{GAME}$.
Consider a ground instance

$$win(a) \leftarrow move(a,b), \neg win(b)$$

of the first clause of GAME. Then by definition

$$|win(a)| = f(a) + 1 > f(a) = |move(a,b)|.$$

Suppose now that $I \models move(a,b)$. Then $move(a,b) \in I$, so $(a,b) \in \mathcal{G}$ and consequently $f(a) > f(b)$. Thus

$$|win(a)| = f(a) + 1 > f(b) + 1 = |\neg win(b)|.$$

□

Corollary 5.4 GAME *is left terminating.*

Proof. By Corollary 2.12. □

Corollary 5.5 *For all terms t, the goal* $\leftarrow win(t)$ *is bounded w.r.t.* $|\;|$ *and* I.

Proof. The goal $\leftarrow win(t)$ is bounded by $max\ \{f(a) + 1 \mid a \in dom(\mathcal{G})\}$. Note that because of the syntax of GAME, t is either a variable or a constant. In the latter case we can improve the bound to $f(t) + 1$. □

Corollary 5.6 *For all terms t, all LDNF-derivations of* GAME $\cup \{\leftarrow win(t)\}$ *are finite.*

Proof. By Corollary 2.11. □

Transitive Closure

Consider the following program computing the transitive closure of a graph.

```
(r1)  r(X,Y,E,V) ←
        member([X,Y],E).
(r2)  r(X,Z,E,V) ←
        member([X,Y],E),
        ¬ member(Y,V),
        r(Y,Z,E,[Y|V]).

(m1)  member(X,[X|T]) ←.
(m2)  member(X,[Y|T]) ←
        member(X,T).
```

In a typical use of this program one evaluates a goal $\leftarrow r(x,y,e,[\,])$ where x,y are nodes and e is a graph specified by a list of its edges. The nodes of e belong to a finite set $\mathcal{A}$. This goal is supposed to succeed when $[x,y]$ is in the transitive closure of e. The last argument of $r(x,y,e,v)$ acts as an accumulator in which one maintains the list of nodes which should not be reused when looking for a path connecting x with y in e (to keep the path acyclic).

To ensure that the elements of $\mathcal{A}$ are in the Herbrand Universe of the program we add to the program the clauses (*e*) `element(a) ←` *for* a $\in \mathcal{A}$,

and call the resulting program `TRANS`.

Lemma 5.7 `TRANS` *is not acyclic.*

Proof. By Lemma 4.1 of Apt and Bezem [AB90] all *SLDNF*-derivations of an acyclic program P starting with a ground goal are finite. Thus it suffices to exhibit an infinite *SLDNF*-derivation of `TRANS` starting in a ground goal. Such a derivation is obtained by using the rightmost selection rule and starting with the ground goal $\leftarrow r(x, z, e, v)$ repeatedly using clause (r_2). □

We now prove that `TRANS` is acceptable. Below we call a list consisting of two elements a *pair.*

First, we define by structural induction two functions on ground terms. We denote the first function by $|\ |$:

$$\begin{aligned} |[x|xs]| &= |xs| + 1, \\ |f(x_1, \ldots, x_n)| &= 0 \text{ if } f \neq [\,.\,|\,.\,]. \end{aligned}$$

Then for a list xs, $|xs|$ equals its length. We denote the second function by *set*:

$$\begin{aligned} set([x|xs]) &= \{x\} \cup set(xs), \\ set(f(x_1, \ldots, x_n)) &= \emptyset \text{ if } f \neq [\,.\,|\,.\,]. \end{aligned}$$

Then for a list xs, $set(xs)$ is the set of its elements.

Define now a Herbrand interpretation I by

$$I = [r(X, Y, E, V)] \cup I_1 \cup \{element(x) \mid x \in \mathcal{A}\}$$

where

$$I_1 = \{member(x, xs) \mid x \in set(xs)\}.$$

Recall that for an atom A, $[A]$ stands for the set of all ground instances of A.

We now prove two lemmata about I and I_1.

Lemma 5.8 *I is a model of* `TRANS`.

Proof. I is clearly a model of (r_1), (r_2) and of the clauses (e). I is also a model of the clauses (m_1) and (m_2) because by definition $x \in set([x|t])$ holds and $x \in set(t)$ implies $x \in set([y|t])$. □

Lemma 5.9 *I_1 is a model of comp*(`TRANS`$^-$).

Proof. Note that TRANS$^-$ $= \{(m_1), (m_2)\}$. We prove that I_1 is a supported interpretation of $\{(m_1), (m_2)\}$. Consider an atom $member(x, xs) \in I_1$. We prove that there exists a ground instance $member(x, xs) \leftarrow L_1, \ldots, L_n$ of (m_1) or (m_2) such that $I \models L_1 \wedge \ldots \wedge L_n$.

By definition $x \in set(xs)$, so for some y and t we have $xs = [y|t]$ and $x \in \{y\} \cup set(t)$. If $x = y$, then $xs = [x|t]$, and the desired clause is an instance of (m_1). Otherwise $x \in set(t)$, so $member(x, t) \in I$, i.e. $I \models member(x, t)$. In this case the desired clause is an instance of (m_2).

By Lemma 5.8 I_1 is a model of $\{(m_1), (m_2)\}$, so by Theorem 2.13 we now conclude that I_1 is a model of $comp(\{(m_1), (m_2)\})$. □

We now define an appropriate level mapping. It is clear that by putting

$$|member(x, y)| = |y|$$

we obtain the desired decrease for clause (m_2). Having made this choice in order to obtain the desired decrease for clause (r_1) we need to have

$$|r(x, z, e, v)| > |e|. \tag{1}$$

Additionally, to obtain the desired decrease for clause (r_2) we need to have (assuming that $I \models member([x, y], e)$)

$$|r(x, z, e, v)| > |v| \tag{2}$$

and, assuming

$$I \models member([x, y], e) \wedge \neg member(y, v), \tag{3}$$

we need to prove

$$|r(x, z, e, v)| > |r(y, z, e, [y|v])|. \tag{4}$$

To define $|r(x, z, e, v)|$ we first define two auxiliary functions. Let

$$nodes(e) = \{x \mid \text{for some pair } b,\ x \in set(b) \ \text{ and } \ b \in set(e)\}.$$

If e is a list of pairs that specifies the edges of a graph $\mathcal{G}$, then $nodes(e)$ is the set of nodes of $\mathcal{G}$.

Let

$$out(e, v) = \{x \mid x \in nodes(e) \ \text{ and } x \notin set(v)\}.$$

If e is a list of pairs that specify the edges of a graph $\mathcal{G}$ and v is a list, then $out(e, v)$ is the set of nodes of $\mathcal{G}$ that are not elements of v.

We now put

$$|r(x, z, e, v)| = |e| + |v| + 2 \cdot card\ out(e, v) + 1,$$

where $card\ X$ stands for the cardinality of the set X.

Then (1) and (2) hold. Assume now (3). Then $[x,y] \in set(e)$ and $y \notin set(v)$. Thus $y \in nodes(e)$ and consequently $y \in out(e,v)$.

On the other hand $set([y|v]) = \{y\} \cup set(v)$. Thus $y \notin out(e,[y|v])$ and $out(e,v) = \{y\} \cup out(e,[y|v])$ so $card\ out(e,v) = card\ out(e,[y|v]) + 1$.

We now have

$$\begin{aligned} |r(x,z,e,v)| &= |e| + |v| + 2 \cdot card\ out(e,v) + 1 \\ &= |e| + |v| + 2 \cdot card\ out(e,[y|v]) + 3 \\ &> |e| + |[y|v]| + 2 \cdot card\ out(e,[y|v]) + 1 \\ &= |r(y,z,e,[y|v])| \end{aligned}$$

which proves (4).

Summarizing, we proved the following result.

Theorem 5.10 TRANS *is acceptable w.r.t.* $|\ |$ *and* I. □

Corollary 5.11 TRANS *is left terminating.*

Proof. By Corollary 2.12. □

Corollary 5.12 *For all terms* x, y *and lists* e, v, *the goal* $\leftarrow r(x,y,e,v)$ *is bounded w.r.t.* $|\ |$ *and* I.

Proof. The goal $\leftarrow r(x,y,e,v)$ is bounded by $|e| + |v| + 2 \cdot card\ out(e,v) + 1$. □

Corollary 5.13 *For all terms* x, y *and lists* e, v, *all* $LDNF$*-derivations of* TRANS $\cup \{\leftarrow r(x,y,e,v)\}$ *are finite.*

Proof. By Corollary 2.11. □

Acknowledgement

Marc Bezem made us aware of the importance of including subsidiary $LDNF$-trees in the definition of $nodes_P(G)$.

References

[AB90] K. R. Apt and M. Bezem. Acyclic programs. In D. H. D. Warren and P. Szeredi, editors, *Proceedings of the Seventh International Conference on Logic Programming*, pages 617–633. The MIT Press, 1990.

[ABW88] K. R. Apt, H. A. Blair, and A. Walker. Towards a theory of declarative knowledge. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 89–148. Morgan Kaufmann, 1988.

[AP90] K. R. Apt and D. Pedreschi. Studies in pure Prolog: termination. In J.W. Lloyd, editor, *Symposium on Computional Logic*, pages 150–176, Berlin, 1990. Springer-Verlag.

[Apt90] K. R. Apt. Logic programming. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 493–574. Elsevier, 1990. Vol. B.

[Bau88] M. Baudinet. Proving termination properties of PROLOG programs. In *Proceedings of the 3rd Annual Symposium on Logic in Computer Science (LICS)*, pages 336–347, Edinburgh, Scotland, 1988.

[Bez89] M. Bezem. Characterizing termination of logic programs with level mappings. In E. L. Lusk and R. A. Overbeek, editors, *Proceedings of the North American Conference on Logic Programming*, pages 69–80. The MIT Press, 1989.

[Cav89] L. Cavedon. Continuity, consistency, and completeness properties for logic programs. In G. Levi and M. Martelli, editors, *Proceedings of the Sixth International Conference on Logic Programming*, pages 571–584. The MIT Press, 1989.

[Der87] N. Dershowitz. Termination of rewriting. *Journal of Symbolic Computation*, 8:69–116, 1987.

[Fit85] M. Fitting. A Kripke-Kleene semantics for general logic programs. *Journal of Logic Programming*, 2:295–312, 1985.

[HM87] S. Hanks and D. McDermott. Nonmonotonic logic and temporal projection. *Artificial Intelligence*, 33:379–412, 1987.

[Kle52] S. C. Kleene. *Introduction to Metamathematics*. van Nostrand, New York, 1952.

[Llo87] J. W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, Berlin, second edition, 1987.

[Plü90a] L. Plümer. *Termination Proofs for Logic Programs*. Lecture Notes in Artificial Intelligence 446, Springer-Verlag, Berlin, 1990.

[Plü90b] L. Plümer. Termination proofs for logic programs based on predicate inequalities. In D. H. D. Warren and P. Szeredi, editors, *Proceedings of the Seventh International Conference on Logic Programming*, pages 634–648. The MIT Press, 1990.

[UvG88] J. D. Ullman and A. van Gelder. Efficient tests for top-down termination of logical rules. *J. ACM*, 35(2):345–373, 1988.