Characterizations of Classes of Programs by Three-Valued Operators

Pascal Hitzler[1] and Anthony Karel Seda[2]

[1] National University of Ireland, Cork, Ireland,
phitzler@ucc.ie,
WWW home page: http://maths.ucc.ie/~pascal/index.html
[2] National University of Ireland, Cork, Ireland,
aks@ucc.ie,
WWW home page: http://maths.ucc.ie/~seda/index.html

Abstract. Several important classes of normal logic programs, including the classes of acyclic, acceptable, and locally hierarchical programs, have the property that every program in the class has a unique two-valued supported model. In this paper, we call such classes *unique supported model classes*. We analyse and characterize these classes by means of operators on three-valued logics. Our studies will motivate the definition of a larger unique supported model class which we call the class of Φ^**-accessible* programs. Finally, we show that the class of Φ^*-accessible programs is computationally adequate in that every partial recursive function can be implemented by such a program.

Proceedings of the 5th International Conference on Logic Programming and Non-Monotonic Reasoning (LPNMR'99), El Paso, Texas, December, 1999. Springer Lecture Notes in Artificial Intelligence Vol. 1730, 1999, pp. 357-371.

1 Introduction

A good deal of recent research in logic programming has been put into the determination of standard, or intended, models for normal logic programs. Some standard semantics, such as the well-founded semantics ([14]) or the stable model semantics ([15]), are applicable to very large classes of programs. However, whilst the general applicability of these semantics is certainly desirable, the study of these large classes of programs has a natural practical limitation: it is possible to assign standard models to logic programs for which useful interpreters have not yet been implemented, and for which it is questionable whether or not this ever will be possible. It is therefore reasonable to study smaller classes of programs whose behaviour is more controlled, so long as these classes are large enough for practical purposes.

On the other hand, certain classes of logic programs have been defined purely in order to study termination and computability properties. For instance, the acyclic programs of Cavedon [8] (initially called locally ω-hierarchical programs by him) are precisely the terminating programs, and were shown by Bezem [7] to be able to compute all the total computable functions, see also [1]. Next, the class of acceptable programs ([3]) was introduced by Apt and Pedreschi. Such programs are left-terminating and, conversely, left-terminating non-floundering programs are acceptable. In fact, the class of all acceptable programs strictly contains the acyclic programs but, nevertheless, is not computationally adequate, i.e. not every partial recursive function can be implemented by such a program. Finally, the class of all locally hierarchical programs was introduced in [8]. However, this class, which also contains all acyclic programs, is computationally adequate under Prolog if the use of safe cuts is allowed ([23]).

All the programs contained in the classes mentioned in the previous paragraph have a common property: they have unique supported models. These classes will be called here *unique supported model classes*. In fact, they even have unique three-valued models under Fitting's Kripke-Kleene semantics ([11]). Thus, the programs in question leave little doubt about the semantics, i.e. the model, which is to be assigned to them as standard model and, in addition, they have interesting computational properties under existing interpreters, as noted above.

In this paper, we will analyse and characterize unique supported model classes by means of certain three-valued logics, and study computability properties of these. In particular, in Section 2 we will introduce three different three-valued logics and their associated consequence operators, and study the relationships between them. In Sections 3.1 and 3.2, we will characterize acceptable and locally hierarchical programs by means of the behaviour of these operators. We will also give constructions of their canonical level mappings.

Prompted by the studies of acceptable and locally hierarchical programs, we will define a new class of programs which we call the Φ^*-*accessible* programs. We study this class in Section 3.3, where it is shown that the Φ^*-*accessible* programs contain the acceptable and the locally hierarchical programs. Moreover, we will show that each Φ^*-*accessible* program has a unique supported model, that each has a canonical level mapping, and that the class of Φ^*-*accessible* programs is computationally adequate under SLDNF-resolution.

Many-valued logics have been employed in several studies of the semantics of logic programs. In particular, they have been used to assign special truth values to atoms which possess certain computational behaviour such as being non-terminating ([11, 20]), being ill-typed ([21]), being floundering ([4]), or failing when backtracking ([6]). The motivation for the definitions of the three-valued logics we will be using in this paper comes from a couple of sources. Primarily, these logics are formulated in order to allow for easy analysis and characterization of the programs or classes of programs in question by using the logic to mimic the defining property of the program or class of programs. This idea is akin to some of those considered in the papers just cited, see also [6], and is a component of work being undertaken by the authors in [16] where a program transformation which outputs a locally hierarchical program, when input an acceptable one, is used in the characterization of acceptable programs given in [16]. Natural questions, partly answered here, then arise as to the different ways that different classes of programs can be characterized. On the other hand, the present work can also be viewed as a contribution to the asymmetric semantics proposed by Fitting and Ben-Jacob in [13] where it is noted that certain differences between Pascal, LISP and Prolog, for example, are easily described in terms of three-valued logic. Thus, [13] is also a source of motivation for our definitions. However, we note that all programs analysed herein do have unique supported models, therefore the third truth value *undefined* will only be used for obtaining the unique supported two-valued model. Hence, interpretations of *undefined* from the point of view of computation (such as non-halting) are not actually necessary in this paper.

Preliminaries and Notation

Our notation basically follows [18], but we will include next a short review of the main terminology used. Given a normal logic program P, we work over an arbitrary preinterpretation J (complete generality is needed in [16] and hence also in this companion paper). We refer to variable assignments which map into the domain $\mathcal{D}$ of J as *J-variable assignments*; the underlying first order language of P will be denoted by $\mathcal{L}$. By $B_{P,J}$, we denote the set of all J-ground instances of atoms in $\mathcal{L}$. Thus, $B_{P,J}$ is the set of all $p(d_1, \ldots, d_n)$, where p is an n-ary predicate symbol in $\mathcal{L}$ and $d_1, \ldots, d_n \in \mathcal{D}$. An element $A = p(d_1, \ldots, d_n)$ of $B_{P,J}$ is called a *J, v-(ground) instance* or *J-(ground) instance* of an atomic formula $A' = p(t_1, \ldots, t_n)$ in $\mathcal{L}$ if there exists a J-variable assignment v such that $A' \mid v = A$, meaning that $t_i \mid v = d_i$ for $i = 1, \ldots, n$, where $t \mid v$ is the denotation of a term t relative to J and v. Since each $t_i \mid v \in \mathcal{D}$, any J-instance of A' is variable free. This extends easily to literals L, where $L = \neg A' = \neg p(t_1, \ldots, t_n)$, say. Thus, the symbol $\neg p(d_1, \ldots, d_n)$ is called a *J, v-(ground) instance* or *J-(ground) instance* of the literal L if there exists a J-variable assignment v such that $p(t_1, \ldots, t_n) \mid v = p(d_1, \ldots, d_n)$. We often loosely refer to J-ground instances of atoms and of literals as *J-ground atoms* and *J-ground literals* respectively or even as *ground atoms* and *ground literals* respectively if J is understood. In accordance with [22, Definition 1], we write $\mathrm{ground}_J(P)$ for the set of all J-(ground) instances of clauses, or J-ground clauses, or simply ground clauses, in P; the latter term being used, of course, when again J is understood. Thus, typically, if $A' \leftarrow L_1, \ldots, L_n$ is a clause in P, then $A' \mid v \leftarrow L_1 \mid v, \ldots, L_n \mid v$ is an element of $\mathrm{ground}_J(P)$, where v is a J-variable assignment such that $A = A' \mid v$ is a J-instance of A' and $L_i \mid v$ is a J-instance of L_i for $i = 1, \ldots, n$. All elements of $\mathrm{ground}_J(P)$ are obtained thus from some clause and some J-variable assignment.

Example 1. As an example of a normal logic program, we give the following program from [3] for computing the transitive closure of a graph.

$$
\begin{aligned}
r(X, Y, E, V) &\leftarrow m([X, Y], E) \\
r(X, Z, E, V) &\leftarrow m([X, Y], E), \neg m(Y, V), r(Y, Z, E, [Y|V]) \\
m(X, [X|T]) &\leftarrow \\
m(X, [Y|T]) &\leftarrow m(X, T)
\end{aligned}
$$

$$e(a) \leftarrow \qquad \text{for all } a \in N$$

Here, N denotes a finite set containing the nodes appearing in the graph as elements. In the program, uppercase letters denote variable symbols, lowercase letters constant symbols, and lists are written using square brackets as usual under Prolog. One evaluates a goal $\leftarrow r(x, y, e, [x])$ where x and y are nodes and e is a graph specified by a list of pairs denoting its edges. The goal is supposed to succeed when x and y can be connected by a path in the graph. The predicate m implements membership of a list. The last argument of the predicate r acts as an accumulator which collects the list of nodes which have already been visited in an attempt to reach y from x. The transitive closure program has been studied in detail in [3, 12].

The set of all two-valued interpretations based on J for a given normal program P will be denoted by $I_{P,J}$. Elements of $I_{P,J}$ are called *J-interpretations* and are called *J-models of P* if they are also models of P. The set $I_{P,J}$ is a complete lattice with respect to the ordering $\subseteq$ defined by $I \subseteq K$ if and only if $I \models A$ implies $K \models A$ for every $A \in B_{P,J}$. In order to simplify notation, we note that $I_{P,J}$ can be identified with the power set $2^{B_{P,J}}$ and the ordering $\subseteq$ is then indeed set-inclusion. For $I \in I_{P,J}$, we set ${}^cI = B_{P,J} \setminus I$. With this convention and following [22, Section 2], in classical two-valued logic we write $I \models p(d_1, \ldots, d_n)$ (respectively $I \models \neg p(d_1, \ldots, d_n)$) if $p(d_1, \ldots, d_n) \in I$ (respectively $p(d_1, \ldots, d_n) \notin I$). By abusing the meaning of conjunction, and its notation, in the obvious way (see [22, Section 2]), it is now meaningful to write $I \models L_1 \mid v, \ldots, L_n \mid v$, where $L_1 \mid v, \ldots, L_n \mid v$ denotes a "conjunction" $L_1 \mid v \wedge \ldots \wedge L_n \mid v$ of J-instances of literals.

The immediate consequence operator $T_{P,J}$ for a given program P is defined as usual as a mapping on $I_{P,J}$ as follows (where $\mathtt{body}$ denotes a conjunction of J-instances of literals):

$$T_{P,J}(I) = \{A \in B_{P,J} \mid \text{ there exists } A \leftarrow \mathtt{body} \text{ in ground}_J(P) \text{ with } I \models \mathtt{body}\}.$$

Finally, recall from [2] that a two-valued J-interpretation M is a supported J-model of P if and only if M (together with Clark's Equality Theory) is a J-model of the Clark-completion of P if and only if $T_{P,J}(M) = M$.

2 Three-Valued Semantics

A *three-valued J-interpretation* of a program P is a pair (T, F) of disjoint sets $T, F \subseteq B_{P,J}$. Given such a J-interpretation $I = (T, F)$, a J-ground atom A is true (t) in I if $A \in T$, false (f) in I if $A \in F$, and undefined (u) otherwise; $\neg A$ is true in I iff A is false in I, $\neg A$ is false in I iff A is true in I and $\neg A$ is undefined in I iff A is undefined in I.

Given $I = (T, F)$, we denote T by I^+ and F by I^-. Thus, $I = (I^+, I^-)$. If $I^+ \cup I^- = B_{P,J}$, we call I a *total* three-valued J-interpretation of the program P. Total three-valued interpretations can be identified with elements of $I_{P,J}$.

Given a program P, the set $I_{P,J,3}$ of all three-valued J-interpretations of P forms a complete partial order (in fact, complete semi-lattice) with the ordering $\leq$ defined by

$$I \leq K \qquad \text{if and only if} \qquad I^+ \subseteq K^+ \text{ and } I^- \subseteq K^-$$

with least element $(\emptyset, \emptyset)$ which we will denote by $\perp$. Notice that total three-valued J-interpretations are maximal elements in this ordering.

In our present context, it will be sufficient to give truth tables for conjunction and disjunction, and we will make use of three different three-valued logics which we are now going to define. It should be noted here that the truth tables for disjunction are the same in all three logics and that disjunction is commutative.

The first logic, which we will denote by $\mathcal{L}_1$, evaluates conjunction as in Fitting's Kripke-Kleene semantics [11] (in fact, as in Kleene's strong three-valued logic, see [13]). Fitting's work built on [20] and was subsequently studied in the literature by Kunen in [17], Apt and Pedreschi in [3], and Naish in [21]. Disjunction will be evaluated differently though, as indicated by the truth table in Table 1.

Table 1. Truth tables for the logics $\mathcal{L}_1$, $\mathcal{L}_2$, and $\mathcal{L}_3$

		Logic $\mathcal{L}_1$		Logic $\mathcal{L}_2$		Logic $\mathcal{L}_3$	
p	q	$p \wedge q$	$p \vee q$	$p \wedge q$	$p \vee q$	$p \wedge q$	$p \vee q$
t	t	t	t	t	t	t	t
t	u	u	u	u	u	u	u
t	f	f	t	f	t	f	t
u	t	u	u	u	u	u	u
u	u	u	u	u	u	u	u
u	f	f	u	u	u	u	u
f	t	f	t	f	t	f	t
f	u	f	u	f	u	u	u
f	f	f	f	f	f	f	f
Operator		$\Phi_{P,1}$		$\Phi_{P,2}$		$\Phi_{P,3}$	

The second three-valued logic, $\mathcal{L}_2$, will be used for studying acceptable programs and is non-commutative under conjunction. It will be sufficient to evaluate u $\wedge$ f to u instead of f and leaving the truth table for $\mathcal{L}_1$ otherwise unchanged. This way of defining conjunction was employed in [4] and [6], see also the discussion of LISP in [13]. The truth table is again given in Table 1.

The third logic, $\mathcal{L}_3$, will be used for studying locally hierarchical and acyclic programs. For this purpose, we use a commutative version of $\mathcal{L}_2$ where we evaluate f $\wedge$ u to u instead of f, see the discussion in [13] of Kleene's weak three-valued logic in relation to Pascal. The truth table is shown in Table 1.

Let P be a normal logic program, and let $\mathcal{L}_i$ denote one of the three-valued logics above, where $i = 1, 2$ or 3. Corresponding to each of these logics we define an operator $F_{P,J}$ on $I_{P,J,3}$ as follows. For $I \in I_{P,J,3}$, let $F_{P,J}(I) = (T, F)$ where T denotes the set

$$\{A \in B_{P,J} \mid \text{there is } A \leftarrow \texttt{body} \in \text{ground}_J(P) \text{ s.t. } \texttt{body} \text{ is true}_i \text{ in } I\},$$

and F denotes the set

$$\{A \in B_{P,J} \mid \text{for every } A \leftarrow \texttt{body} \in \text{ground}_J(P), \texttt{body} \text{ is false}_i \text{ in } I\}.$$

Of course, true$_i$ and false$_i$ here denote truth respectively falsehood in the logic $\mathcal{L}_i$. Notice that if A is not the head of any clause in P, then A is false in $F_{P,J}(I)$ for any I.

It is clear that $F_{P,J}$ is monotonic in all three cases. We set $F_{P,J}\uparrow 0 = \bot$,

$$F_{P,J}\uparrow\alpha = F_{P,J}(F_{P,J}\uparrow(\alpha - 1)) \text{ for } \alpha \text{ a successor ordinal, and}$$
$$F_{P,J}\uparrow\alpha = \bigcup_{\beta<\alpha} F_{P,J}\uparrow\beta \text{ for } \alpha \text{ a limit ordinal.}$$

Since $F_{P,J}$ is monotonic, it has a least fixed point which is equal to $F_{P,J}\uparrow\alpha$ for some ordinal α called the *closure ordinal* of P (for the chosen logic $\mathcal{L}_i$).

Throughout the sequel, we will denote $F_{P,J}$ by $\Phi_{P,1}$, $\Phi_{P,2}$ or $\Phi_{P,3}$ if the chosen logic is correspondingly $\mathcal{L}_1$, $\mathcal{L}_2$ or $\mathcal{L}_3$. The appropriate symbol is also included in Table 1 for ease of reference. Note that the behaviour of each of these operators depends only on the evaluation of conjunction. In fact, $\Phi_{P,1}$ is the very same operator as used in [11].

Proposition 1. *Let P be a normal logic program and let $I, I', I'' \in I_{P,J,3}$ be such that $I \leq I' \leq I''$. Then we have*

$$\Phi_{P,3}(I) \leq \Phi_{P,2}(I') \leq \Phi_{P,1}(I'').$$

Proof. The following observations are clear from the given truth tables, and indeed suffice. If a body of a clause is true (false) in $\mathcal{L}_3$, then it is true (false) in $\mathcal{L}_2$. If it is true (false) in $\mathcal{L}_2$, then it is true (false) in $\mathcal{L}_1$.

The following result is taken from [16] generalizing a result in [3].

Proposition 2. *Let P be a normal logic program and let $I = (I^+, {}^cI^+)$ be a total three-valued J-interpretation for P. Then I is a fixed point of $\Phi_{P,1}$ if and only if I^+ is a fixed point of $T_{P,J}$. Furthermore, if $\Phi_{P,1}$ has exactly one fixed point M and M is total, then M^+ is the unique fixed point of $T_{P,J}$.*

Proposition 3. *Let P be a normal logic program, let $F_{P,J}$ denote $\Phi_{P,i}$, for $i = 1, 2, 3$, and assume that $M = F_{P,J}\uparrow\alpha$ is total, where α is the corresponding closure ordinal of P. Then M^+ is the unique two-valued supported J-model of P.*

Proof. By totality of M and the previous results we obtain M^+ as a fixed point of $T_{P,J}$. Since M is the least fixed point of $F_{P,J}$ and is maximal in $I_{P,J,3}$, it is the unique fixed point of $F_{P,J}$ which finishes the proof.

Given a J-ground atom A which occurs as the head of an element $A \leftarrow C$ of $\text{ground}_J(P)$, we form the *J-pseudo clause*, or simply *pseudo clause*, $A \leftarrow \vee_i C_i$ whose *body* $\vee_i C_i$ is the (possibly infinite) disjunction of the bodies C_i of all clauses in $\text{ground}_J(P)$ whose head is A; we call A the *head* of the pseudo clause $A \leftarrow \vee_i C_i$. The set of all such pseudo clauses will be denoted by P^*. It will be convenient to assign "truth" values to $\vee_i C_i$, relative to the logics $\mathcal{L}_i$ by in fact assigning truth values to arbitrary disjunctions of literals and then employing the same sort of abuse for "disjunctions" of J-ground literals which was established earlier for conjunction. This is done as follows: $\vee_i C_i$ will be assigned value *true* (t) iff at least one C_i is true and none are undefined; it will be assigned value *undefined* (u) iff at least one C_i is undefined; it will be assigned value *false* (f) iff all the C_i are false. These definitions are the natural extension to possibly infinite disjunctions of the values given iteratively to finite disjunctions by the truth tables in Table 1.

Letting $F_{P,J}$ denote any one of the $\Phi_{P,i}$, for $i = 1, 2, 3$, we define an operator F_{P^*} on $I_{P,J,3}$ as follows. For $I \in I_{P,J,3}$, set $F_{P^*}(I) = (T, F)$, where T is the set of all J-ground atoms which occur as the head of a pseudo clause in P^* whose body is true in I, and F is the set of all J-ground atoms which occur as the head of a pseudo clause whose body is false in I. As before, $\Phi_{P^*,i}$ will denote F_{P^*} when the chosen logic is $\mathcal{L}_i$, $i = 1, 2, 3$. Note that F_{P^*} is again monotonic for any choice of underlying logic. Ordinal powers $F_{P^*}\uparrow\alpha$ are defined as for $F_{P,J}$.

Example 2. We give an example illustrating the program transformation P^*. Let P be the (propositional) program

$$\begin{aligned} &a \leftarrow b \\ &a \leftarrow c \\ &b \leftarrow \\ &c \leftarrow c \end{aligned}$$

then P^* is

$$\begin{aligned} &a \leftarrow b \vee c \\ &b \leftarrow \\ &c \leftarrow c \end{aligned}$$

Let I be the three-valued interpretation $(\{b\}, \emptyset)$. Then $\Phi_{P,1}(I) = (\{a, b\}, \emptyset)$, which is also the least fixed point of $\Phi_{P,1}$. However, since c is undefined in I, we have $\Phi_{P^*,1}(I) = (\{b\}, \emptyset)$, which is the least fixed point of $\Phi_{P^*,1}$. The difference between $\Phi_{P,1}$ and $\Phi_{P^*,1}$ results from the way in which disjunction is defined, see the following proposition, Proposition 4. In fact, in this context it is worth noting an observation made by one of the referees of this paper, as follows. In classical two-valued logic, the programs $(a \leftarrow b) \wedge (a \leftarrow c)$ and $a \leftarrow (b \vee c)$ are equivalent simply because of the distributive laws and De Morgan's law that $\neg b \wedge \neg c$ and $\neg(b \vee c)$ are equivalent. In the Logics $\mathcal{L}_i$, $i = 1, 2, 3$, $\neg b \wedge \neg c$ and $\neg(b \vee c)$ are not equivalent as can easily be verified by, for example, taking b to be true and c to be undefined. In fact, the rule $a \leftarrow (b \vee c)$ with disjunctive body is weaker (leaves more undefined) than the two separate rules $a \leftarrow b$ and $a \leftarrow c$.

Proposition 4. *Let P be a normal logic program and let $I, I', I'' \in I_{P,J,3}$ be such that $I \leq I' \leq I''$. Then we have*

$$\Phi_{P^*,3}(I) \leq \Phi_{P^*,2}(I') \leq \Phi_{P^*,1}(I''),$$

and for F denoting any of the Φ_i, for $i = 1, 2, 3$, we have

$$F_{P^*}(I) \leq F_{P,J}(I) \qquad \text{and} \qquad F_{P^*}(I)^- = F_{P,J}(I)^-.$$

Proof. The proof is along the same lines as the proof of Proposition 1 noting that in a disjunction $\vee_i C_i$ which is true, no C_i is undefined.

3 Unique Supported Model Classes

3.1 Acceptable Programs

Acceptable programs were introduced in [3] and were shown to be strongly related to left-terminating programs. Given a normal logic program P, a *level mapping* for P is a mapping l from J-ground atoms to an ordinal α. We always assume that l is extended to J-ground literals by setting $l(\neg A) = l(A)$ for every J-ground atom A. A level mapping which maps into ω will be called an ω-level mapping. Following [3], we define a subset P^- of P as follows.

Definition 1. *Let P be a normal logic program and let p, q be predicate symbols occurring in P.*
(i) We say that p refers to *q if there is a clause in P with p in its head and q in its body.*
(ii) We say that p depends on *q if (p, q) is in the reflexive, transitive closure of the relation* refers to.
(iii) The set of predicate symbols in P which occur in a negative literal in the body of a clause in P is denoted by Neg_P.
(iv) The set of predicate symbols in P on which the predicate symbols in Neg_P depend is denoted by Neg_P^*. For convenience, we will denote this set simply by N.
(v) We define P^- to be the set of clauses in P which contain a predicate symbol from N in the head.

The following definition is the generalization to an arbitrary preinterpretation J of the definition of acceptability given in [3] for the Herbrand preinterpretation.

Definition 2. *Let P be a normal logic program, let l be an ω-level mapping for P, and let I be a (two-valued) J-model for P whose restriction to the predicate symbols in N is a supported J-model of P^-. Then P is called J-*acceptable with respect to l and I *if, for every clause* $A \leftarrow L_1, \ldots, L_n$ *in* $ground_J(P)$*, the following implication holds for all* $i \in \{1, \ldots, n\}$*:*

$$\text{if } I \models \bigwedge_{j=1}^{i-1} L_j \text{ then } l(A) > l(L_i).$$

A program is called J-acceptable with respect to l *if* l *is a level mapping and there exists a* J*-model* I *such that the program is* J*-acceptable with respect to* l *and* I*. A program is called* J-acceptable, *or just* acceptable if J is understood, *if it is* J*-acceptable with respect to some level mapping and some* J*-model.*

Example 3. The transitive closure program given in Example 1 is Herbrand-acceptable; for details of the model and level mapping required, see [3].

We are able to characterize J-acceptable programs by means of the operator $\Phi_{P^*,2}$, and we do this next. We will need the following proposition from [16].

Proposition 5. *Suppose that* P *is* J*-acceptable with respect to a level mapping* l*. Then* $M_{P,J} = \Phi_{P,1}\uparrow\omega$ *is total,* $M^+_{P,J}$ *is the unique supported* J*-model of* P *and* P *is* J*-acceptable with respect to* l *and* $M^+_{P,J}$*.*

Lemma 1. *Let* P *be* J*-acceptable. Then* $M = \Phi_{P^*,2}\uparrow\omega$ *is total. Furthermore,* $M = \Phi_{P,2}\uparrow\omega$*, and* M^+ *is the unique supported* J*-model of* P*.*

Proof. Let l be a level mapping with respect to which P is J-acceptable. By Proposition 5, P is J-acceptable with respect to l and $M^+_{P,J}$. Assume that there is a J-ground atom A which is undefined in M. Without loss of generality we can assume that $l(A)$ is minimal. Then by definition of $\mathcal{L}_2$, there is precisely one pseudo clause in P^* of the form $A \leftarrow \vee_i C_i$ in which at least one of the C_i, say C_1, is undefined. Thus, there must occur a left-most J-ground body literal B in C_1 which is undefined in M, and this ground literal is to the left in C_1 of the first ground literal which is false in M. Hence, all ground literals occurring to the left of B must be true in M. Since $M \leq M_{P,J}$ by Proposition 4, all these ground literals must also be true in $M^+_{P,J}$. By acceptability of P we therefore conclude that $l(B) < l(A)$, contradicting the minimality of $l(A)$. By Proposition 4, the second statement holds. The last statement follows from Proposition 3.

Definition 3. *Let* P *be* J*-acceptable. Define its* canonical level mapping *as follows:* $l_P(A)$ *is the lowest ordinal* α *such that* A *is not undefined in* $\Phi_{P^*,2}\uparrow(\alpha+1)$*.*

Proposition 6. *Let* P *be* J*-acceptable. Then* l_P *is an* ω*-level mapping and* P *is* J*-acceptable with respect to* l_P *and* $M_{P,J}$*. Furthermore, if* l *is another level mapping with respect to which* P *is* J*-acceptable, then* $l_P(A) \leq l(A)$ *for all* $A \in B_{P,J}$*. In particular,* l_P *is exactly the canonical level mapping defined in [16].*

Proof. By the previous lemma, l_P is indeed an ω-level mapping.

Let A be the head of a J-ground clause $\mathcal{C}$ in P with $l_P(A) = n$. Then the body $\vee_i C_i$ of the corresponding pseudo clause in P^* is either true or false (i.e. is not undefined) in $\mathcal{N} = \Phi_{P^*,2} \uparrow n$. If $\vee_i C_i$ is true, each C_i evaluates to true or false in $\mathcal{N}$. If C_i evaluates to true in $\mathcal{N}$ (and at least one must), then all J-ground literals in C_i are true in $\mathcal{N}$, and therefore have level less than or equal to $n-1$. If C_i evaluates to false in $\mathcal{N}$, then there must be a ground literal in C_i which is false in $\mathcal{N}$ such that all ground literals occurring to the left of it are true in $\mathcal{N}$. Moreover all these ground literals are not undefined in $\mathcal{N}$ and hence have level less than or equal to $n-1$. A similar argument applies if $\vee_i C_i$ is false in $\mathcal{N}$. Since $\mathcal{N} \le \mathcal{M}_{\mathcal{P},\mathcal{J}}$, it is now clear that the clause $\mathcal{C}$ satisfies the condition of acceptability given in Definition 2 with respect to l_P and $M_{P,J}$.

Now let l be another level mapping with respect to which P is J-acceptable. By Proposition 5, P is J-acceptable with respect to l and $M_{P,J}$. Let $A \in B_{P,J}$ with $l(A) = n$. We show by induction on n that $l(A) \ge l_P(A)$. If $n = 0$, then A appears only as the head of unit clauses, and therefore $l_P(A) = 0$. Now let $n > 0$. Then in every clause with head A, the left prefix of the corresponding body, up to and including the first ground literal which is false in $M_{P,J}$, contains only ground literals L with $l(L) < n$. By the induction hypothesis, $l_P(L) < n$ for all these ground literals L and, consequently, $l_P(A) \le l(A)$ by definition of l_P.

The last statement follows from [16], where it is shown that the given minimality property characterizes l_P.

We are now in a position to characterize J-acceptable programs.

Theorem 1. *Let P be a normal logic program. Then P is J-acceptable if and only if $M = \Phi_{P^*,2}\uparrow\omega$ is total.*

Proof. By Lemma 1 it remains to show that totality of M implies acceptability. Define the ω-level mapping l_P for P as in Definition 3. Since M is total, l_P is indeed an ω-level mapping for P. We will show that P is J-acceptable with respect to l_P and M.

Arguing as in the proof of the previous proposition, let A be the head of a J-ground clause $\mathcal{C}$ in P with $l_P(A) = n$. Then the corresponding body C evaluates to true or false in $\mathcal{N} = \Phi_{P^*,2}\uparrow n$. If it evaluates to true in $\mathcal{N}$, then all J-ground literals in C are true in $\mathcal{N}$, and therefore have level less than or equal to $n-1$. If it evaluates to false in $\mathcal{N}$, then there must be a ground literal in C which is false in $\mathcal{N}$ such that all ground literals occurring to the left of it are true in $\mathcal{N}$. Again, all these ground literals are not undefined in $\mathcal{N}$ and hence have level less than or equal to $n-1$. Since $\mathcal{N} \le \mathcal{M}$, the clause $\mathcal{C}$ satisfies the condition of acceptability given in Definition 2.

In [19], it was shown that the class of programs which terminate under Chan's constructive negation ([10]) coincides with the class of programs which are acceptable with respect to a model based on a preinterpretation whose domain is the Herbrand universe and contains infinitely many constant and function symbols. We therefore obtain the following result.

Theorem 2. *A normal logic program P terminates under Chan's constructive negation if and only if $\Phi_{P^*,2}\uparrow\omega$ is total, where $\Phi_{P^*,2}$ is computed with respect to a preinterpretation whose domain is the Herbrand universe and contains infinitely many constant and function symbols.*

3.2 Locally Hierarchical Programs

Locally hierarchical programs were introduced in [8], for the special case of the Herbrand base, as a natural generalization of acyclic programs. They were further studied in [9] and in [23] (and also called *strictly level-decreasing* there). Here, we consider them over an arbitrary preinterpretation J and our definition and subsequent results are therefore completely general.

Definition 4. *A normal logic program P is called* locally hierarchical *if there exists a level mapping $l : B_{P,J} \to \alpha$, where α is some countable ordinal, such that for every clause $A \leftarrow L_1, \ldots, L_n$ in $\mathrm{ground}_J(P)$ we have $l(A) > l(L_i)$ for all i. If, further, $\alpha = \omega$, we call P* acyclic.

We will now give a new characterization of these programs along the lines of Theorem 1, using the operator $\Phi_{P^*,3}$.

Lemma 2. *Let P be locally hierarchical with respect to the level mapping l and let $A \in B_{P,J}$ be such that $l(A) = \alpha$. Then A is true or false in $\Phi_{P^*,3}\uparrow(\alpha+1)$. In particular, there exists an ordinal α_P such that $\Phi_{P^*,3}\uparrow\alpha_P$ is total.*

Proof. The proof is by transfinite induction on α. The base case follows directly from the fact that if $\alpha = 0$, then A appears as head of unit clauses only. Now let $\alpha = \beta + 1$ be a successor ordinal. Then all J-ground literals appearing in bodies of clauses with head A have level less than or equal to β. By the induction hypothesis, they are all not undefined in $\Phi_{P^*,3}\uparrow(\beta+1)$ and therefore A is either true or false in $\Phi_{P^*,3}\uparrow(\alpha+1)$. If α is a limit ordinal, then all ground literals occurring in bodies of clauses with head A have level strictly less than α. Hence, by the induction hypothesis and since α is a limit ordinal, all these ground body literals are not undefined in $\Phi_{P^*,3}\uparrow\alpha$, and therefore A is true or false in $\Phi_{P^*,3}\uparrow(\alpha+1)$.

Corollary 1. *Let P be a locally hierarchical program with level mapping $l : B_{P,J} \to \alpha$ and let $M = \Phi_{P,1}\uparrow\alpha$. Then M is total and $M_{P,J} = M^+$ is the unique supported J-model of P.*

Proof. By Propositions 1 and 4, we have $\Phi_{P^*,3}\uparrow\beta \leq \Phi_{P,3}\uparrow\beta \leq \Phi_{P,1}\uparrow\beta$ for all ordinals β. Since $\Phi_{P^*,3}\uparrow\alpha$ is total by Lemma 2, the given statement holds using Proposition 3.

Definition 5. *Let P be locally hierarchical. Define the* canonical level mapping *l_P of P as a function $l_P : B_{P,J} \to \alpha_P$ where $l_P(A)$ is the least ordinal α such that A is true or false in $\Phi_{P^*,3}\uparrow(\alpha+1)$.*

Proposition 7. *Let P be locally hierarchical with respect to some level mapping l. Then l_P is a level mapping for P and, for all $A \in B_{P,J}$, we have $l_P(A) \leq l(A)$. Furthermore, the notion of canonical level mapping as defined here coincides with the same notion defined by different methods in [23].*

Proof. The mapping l_P is indeed a level mapping by Lemma 2. Let $A \in B_{P,J}$ with $l(A) = \alpha$. We show the given minimality statement by transfinite induction on α. If $\alpha = 0$, then A appears as the head of unit clauses only, and so $l_P(A) = 0$. If $\alpha = \beta + 1$ is a successor ordinal, then all J-ground literals L occurring in bodies of clauses with head A have level $l(L) \leq \beta$. By the induction hypothesis, we obtain $l_P(L) \leq \beta$ for all those ground literals, and so $l_P(A) \leq \alpha = l(A)$ by construction of l_P. If α is a limit ordinal, then all ground literals L occurring in bodies of clauses with head A have level $l(L) < \alpha$. Since $l_P(L) \leq l(L)$ and since α is a limit ordinal, we obtain that all these ground literals L are not undefined in $\Phi_{P^*,3}\uparrow\alpha$ and therefore $l_P(A) \leq \alpha = l(A)$ as desired.

The last statement follows since the minimality property just proved characterizes the canonical level mapping as was shown in [23].

Note that it is an easy corollary of the previous results that if a program P is acyclic, then $\Phi_{P^*,3}\uparrow\omega$ is total.

Theorem 3. *A normal logic program P is locally hierarchical if and only if $\Phi_{P^*,3}\uparrow\alpha$ is total for some ordinal α. It is acyclic if and only if $\Phi_{P^*,3}\uparrow\omega$ is total.*

Proof. Let P be a normal logic program such that $\Phi_{P^*,3}\uparrow\alpha$ is total for some α. We define a mapping $l : B_{P,J} \to \alpha$ by analogy with the definition of the canonical level mapping for locally hierarchical programs. From the definion of $\mathcal{L}_3$ it is now obvious that P is indeed locally hierarchical with canonical level mapping l. The reverse was shown in the previous proposition. The statement for acyclic programs now follows immediately as well.

3.3 Φ^*-Accessible Programs

Our investigations of J-acceptable and locally hierarchical programs suggest we define a class of programs by the property that $\Phi_{P^*,1}\uparrow\alpha$ is total for some ordinal α. We will do this next and show also that this class is computationally adequate.

Definition 6. *A normal logic program P will be called a Φ^*-*accessible program *if $\Phi_{P^*,1}\uparrow\alpha$ is total for some ordinal α.*

Theorem 4. *Every Φ^*-accessible program has a unique supported J-model. Furthermore, the class of Φ^*-accessible programs contains all J-acceptable and all locally hierarchical programs.*

Proof. Immediate by Propositions 3 and 4.

Definition 7. *The* canonical level mapping *l^* for a given Φ^*-accessible program is defined as follows. For every $A \in B_{P,J}$, set $l^*(A) = \alpha$, where α is the minimal ordinal such that A is true or false in $\Phi_{P^*,1}\uparrow(\alpha + 1)$.*

The following is immediate by Proposition 4.

Proposition 8. *If P is J-acceptable or locally hierarchical with canonical level mapping l_P, then $l^*(A) \geq l_P(A)$ for all J-ground atoms A.*

Proposition 9. *Let P be Φ^*-accessible with unique supported J-model M. Let $\mathcal{C}$ be an arbitrary element of $\mathrm{ground}_J(P)$, let A be its head, and let $l^*(A) = \alpha$. Then the following property $(*)$ holds: Either the body of $\mathcal{C}$ is true in M, in which case every J-ground literal L in this body has level $l^*(L) < \alpha$, or there exists a ground body literal B in $\mathcal{C}$ which is false in M, and in this case $l^*(B) < \alpha$. Furthermore, if l is a level mapping for P which satisfies $(*)$, then $l^*(A) \leq l(A)$ for every $A \in B_{P,J}$.*

Proof. Since P is Φ^*-accessible, every body of every J-ground clause with head A is either true or false in $\Phi_{P^*,1} \uparrow \alpha$. In particular, the body of $\mathcal{C}$ is true or false in $\Phi_{P^*,1} \uparrow \alpha$. If it is true, then all J-ground literals L in the body are true in $\Phi_{P^*,1} \uparrow \alpha$ and so $l^*(L) < \alpha$ by definition of l^*. If the body is false, then there is a ground body literal B which is false in $\Phi_{P^*,1} \uparrow \alpha$, and again by definition of l^* we obtain $l^*(B) < l(A)$.

The minimality property of l^* is shown by transfinite induction along the same lines as in the proofs of the Propositions 6 and 7.

It was shown in [23] that the class of all locally hierarchical programs is computationally adequate in the sense that every partial recursive function can be computed with such a program if the use of safe cuts is allowed. For Φ^*-accessible programs, the cut need not be used, and we will show this next. The proof basically shows that given a partial recursive function, there is a definite program as given in [18] which computes that function. This program will turn out to be a Φ^*-accessible program.

Theorem 5. *Let f be a partial recursive function. Then there exists a definite Φ^*-accessible program which computes f.*

Proof. We will make use of the definite program P_f given in [18, Theorem 9.6], and we refer the reader to the proof of this theorem for details. It is easily seen that we have to consider the minimalization case only. In [18], the following program P_f was given as an implementation of a function f which is the result of applying the minimalization operator to a partial recursive function g, which is in turn implemented by a predicate p_g. We abbreviate $X_1, \ldots, X_n$ by X.

$$\begin{aligned} p_f(X,Y) &\leftarrow p_g(X,0,U), r(X,0,U,Y) \\ r(X,Y,0,Y) &\leftarrow \\ r(X,Y,s(V),Z) &\leftarrow p_g(X,s(Y),U), r(X,s(Y),U,Z) \end{aligned}$$

This program is not Φ^*-accessible. However, we can replace it with a program P_f' which has the same procedural behaviour and is Φ^*-accessible. In fact, we replace the definition of r by

$$\begin{aligned} r(X,Y,0,Y) &\leftarrow \\ r(X,Y,s(V),Z) &\leftarrow p_g(X,s(Y),U), r(X,s(Y),U,Z), lt(Y,Z), \end{aligned}$$

where the predicate lt is in turn defined as

$$lt(0, s(X)) \leftarrow$$
$$lt(s(X), s(Y)) \leftarrow lt(X, Y)$$

and is obviously Φ^*-accessible. By a straightforward analysis of the original program P_f, it is clear that the addition of $lt(y, z)$ in the second defining clause of r does not alter the behaviour of the program. Since lt and p_g are Φ^*-accessible, it is now easy to see that r is Φ^*-accessible, and so therefore is P'_f.

It is worth noting that negation is not needed here in order to obtain full computational power, so Theorem 5 strenghtens the result of [18] referred to in the proof of Theorem 5. By contrast, as already noted, definite locally hierarchical programs seem not to provide full computational power. Regardless of some known drawbacks in SLDNF-resolution, it is interesting to know that relative to it the class of all Φ^*-accessible programs has full computational power – neither the class of acyclic nor even the class of J-acceptable programs has this property.

4 Conclusions

The rather simple characterizations of the classes discussed in this paper are a contribution to exploring the "space" of all normal programs, a task which appears not yet to have been addressed very extensively. Both the class of locally hierarchical programs and the class of J-acceptable programs are natural generalizations of acyclic programs; the first can be understood as a generalization in semantical terms, and the second as a generalization expressing termination.

The results presented in this paper establish a common framework which highlights more clearly the differences and the similarities between these generalizations: each can be obtained uniquely by suitably defining conjunction in the underlying three-valued logic whilst retaining a fixed meaning for disjunction. Our approach then leads naturally to the definition of the class of all Φ^*-accessible programs, by choosing yet another definition of conjunction. This class is remarkable for two reasons: (i) each program in it has a unique supported J-model, and (ii) the class itself has full computational power under SLDNF-resolution whilst containing all J-acceptable and all locally hierarchical programs, but not all definite programs. However, a simple syntactical description of this class and how it relates to other better known classes is not yet known to us, nor is the complexity of deciding if a program is Φ^*-accessible.

Other classes of programs may well be susceptible to the sort of analysis presented here, and this also is ongoing research of the authors. As already noted in the Introduction, such an investigation carries forward the suggestion made in [13] that asymmetric semantics is worthy of further study.

Acknowledgements The authors wish to thank three anonymous referees for their comments which substantially helped to improve the style of this paper.
The first named author acknowledges financial support under grant SC/98/621 from Enterprise Ireland.

References

1. Apt, K.R., Bezem, M.: Acyclic Programs. In: Warren, D.H.D., Szeredi, P. (Eds.): Proceedings of the Seventh International Conference on Logic Programming. MIT Press, Cambridge MA, 1990, pp. 617–633
2. Apt, K.R., Blair, H.A., Walker, A.: Towards a Theory of Declarative Knowledge. In: Minker, J. (Ed.): Foundations of Deductive Databases and Logic Programming. Morgan Kaufmann Publishers Inc., Los Altos, 1988, pp. 89–148
3. Apt, K.R., Pedreschi, D.: Reasoning about Termination of Pure Prolog Programs. Information and Computation **106** (1993) 109–157
4. Andrews, J.H.: A Logical Semantics for Depth-first Prolog with Ground Negation. Theoretical Computer Science **184** (1–2) (1997) 105–143
5. Bidoit, N., Froidevaux, C.: Negation by default and unstratifiable logic programs. Theoretical Computer Science **78** (1991) 85–112
6. Barbuti, R., De Francesco, N, Mancarella, P, Santone, A.: Towards a Logical Semantics for Pure Prolog. Science of Computer Programming **32** (1–3) (1998) 145–176
7. Bezem, M: Characterizing Termination of Logic Programs with Level Mappings. In: Lusk, E.L., Overbeek R.A.(Eds.): Proceedings of the North American Conference on Logic Programming. MIT Press, Cambridge MA, 1989, pp. 69–80
8. Cavedon, L.: Continuity, Consistency, and Completeness Properties for Logic Programs. In: Levi, G., Martelli, M. (Eds.): Proceedings of the 6th International Conference on Logic Programming. MIT Press, Cambridge MA, 1989, pp. 571–584
9. Cavedon L.: Acyclic Logic Programs and the Completeness of SLDNF-Resolution. Theoretical Computer Science **86** (1991) 81–92
10. Chan, D.: Constructive Negation Based on the Completed Database. In: Proc. of the 5th Int. Conf. and Symp. on Logic Programming, 1988, pp. 111–125
11. Fitting, M.: A Kripke-Kleene Semantics for General Logic Programs. J. Logic Programming **2** (1985) 295-312
12. Fitting, M.: Metric Methods: Three Examples and a Theorem. J. Logic Programming **21** (3) (1994) 113–127
13. Fitting, M., Ben-Jacob, M.: Stratified, Weak Stratified, and Three-Valued Semantics. Fundamenta Informaticae **XIII** (1990) 19–33
14. Van Gelder, A., Ross, K.A., Schlipf, J.S.: The Well-Founded Semantics for General Logic Programs. Journal of the ACM **38** (3) (1991) 620–650
15. Gelfond, M., Lifschitz, V.: The Stable Model Semantics for Logic Programming. In: Kowalski, R.A., Bowen, K.A. (Eds.): Proceedings of the 5th International Conference and Symposium on Logic Programming, MIT Press, 1988, pp. 1070–1080
16. Hitzler, P., Seda, A.K.: Acceptable Programs Revisited. Preprint, Department of Mathematics, University College Cork, Cork, Ireland, 1999, pp. 1–15
17. Kunen, K.: Negation in Logic Programming. J. Logic Programming **4** (1987) 289–308
18. Lloyd, J.W.: Foundations of Logic Programming. Second Edition, Springer, Berlin, 1988
19. Marchiori, E.: On Termination of General Logic Programs with respect to Constructive Negation. J. Logic Programming **26** (1) (1996) 69–89
20. Mycroft, A.: Logic Programs and Many-valued Logic. In: Fontet, M., Mehlhorn, K. (Eds.): STACS 84, Symposium of Theoretical Aspects of Computer Science, Paris, France, 1984, Proceedings. Lecture Notes in Computer Science, Vol. 166, Springer, 1984, pp. 274–286

21. Naish, L.: A Three-Valued Semantics for Horn Clause Programs. Technical Report 98/4, University of Melbourne, pp. 1–11
22. Seda, A.K.: Topology and the Semantics of Logic Programs. Fundamenta Informaticae **24** (4) (1995) 359–386
23. Seda, A.K., Hitzler, P.: Strictly Level-decreasing Logic Programs. In: Butterfield, A., Flynn, S. (Eds.): Proceedings of the Second Irish Workshop on Formal Methods 1998 (IWFM'98), Electronic Workshops in Computing, British Computer Society, 1999, 1–18