Compactness properties for stable semantics of logic programs

V.W. Marek* J.B. Remmel†

December 3, 2006

Abstract

Logic programming with stable logic semantics (SLP) is a logical formalism that assigns to sets of clauses in the language admitting negations in the bodies a special kind of models, called stable models. This formalism does not have the compactness property. We show a number of conditions that entail a form of compactness for SLP.

1 Introduction

When logicians look at a knowledge representation formalism they ask a variety of questions. They looks at a syntax, and on the semantics. In semantical considerations, since Gödel and Hilbert, logicians recognize that not all models of the formalism are *intended*. Quite often semantics that the agent (user of the formalism) assigns to to the syntax require significant restrictions. For instance logicians will often limit the classes of intended models to ones that give intended meaning to one or more predicates. An examples of this approach is in studies of ω-models of theories interpreting the arithmetic of positive integers. The reasoning agent may limit models by looking only at models of some size of the universe (for instance denumerable models or finite models) or of special form (for instance transitive models od set theories, or models that are levels of cumulative hierarchy, V_α. The rise of Computer Science and of Artificial Intelligence introduced new classes of formalisms and new classes of their models. While previously, the mathematical practice and imagintion of logicians provided the basic intuitions in selecting formalisms and their semantics, the applications on Computer Science and Artificial Intelligence resulted in a variety of new formalisms with intuitions in the areas previously limited to informal, philosophical investigations only.

It should be observed that since Gödel [13] logicians were concerned with the nature of computation. Related isssues, such as constructivity of argument were also discussed by logicians, esp. intuitionists, before the advent of modern computers. The relationship of complexity of sets of natural numbers, and existence of algorithms for deciding some mathematical theories has a very long history, and was present in the minds of mathematicians long before the viable computing devices were introduced.

Practical computing, and availability of symbolic computation in particular revolutionized the approach to foundational issues. today we know that logic and computation are inextricably connected. It is not our imagination only. We use computers not only as tools for solving equations, or finding

*Department of Computer Science, University of Kentucky, Lexington, KY 40506–0046. Work partially supported by NSF grant IIS-0325063.

†Department of Mathematics, University of California at San Diego, La Jolla, CA 92903. Work partially supported by NSF grant .

values of functions but also as means to perform tasks that were reserved to human mental activities not long time ago.

As more and more solving tasks associated with human mental capabilities are delegated to computers, a variety of new formalisms for formaliztions of these tasks are introduced. Here we will be interested in one such formalism, so called *logic programming with stable semantics* which we will abbreviate SLP. This formalism, although using the name logic *programming* is grounded in the work of logicians on so-called *Horn formulas*. These formulas, introduced by A. Horn in late 1940ies, has been extensively studied for a variety of reasons and in various contexts. In the 1960ies, S.C. Kleene and R. Smullyan, showed that partial recursive functions can be represented in Horn logic. Then R. Kowalski introduced algorithms for testing the presence of an atom in the least model of Horn theory. These ideas were transformed into a practical programming language, called PROLOG by A. Colmerauer. This language one of the family of *declarative* programming languages and many of its implementations resulted in practical tools and are used in a variety of applications. The basic construct (in a somewhat idealized PROLOG) is a clause (also known as a rule), an expression of the form $p \leftarrow l_1, \ldots, l_k$ where p is an atom, and $l_1, \ldots, l_k$ are literals (atoms or negated atoms). This is a step beyond Horn logic proper, where $l_1, \ldots, l_k$ need to be atoms[1]. From the very beginning, the implementations of PROLOG admitted admitted negated atoms among $l_1, \ldots, l_k$ and attempted to assign to the clauses a meaning that may informally be stated like this: "*if* $l_1, \ldots, l_k$ *have been established then we establish* p". This works well in Horn case, resulting in the *intended* semantics of the least model. But when we admit negated literals among $l_1, \ldots, l_k$ then the interpretation of rules is not clear. Among a large number of proposals, the one that, essentially, gained concensus as a correct one, is the *stable semantics* due to M. Gelfond and V. Lifschitz [12]. Stable semantics is, in effect, an interpretation of rules as *default rules* studied by R. Reiter. The consensus on stable semantics has been reached in the mid 1990ies. Today, stable sematics and its extensions to so-called answer sets is a basis for so-called *Answer Set Programming* (ASP) with several practical implementations.

When logicians look at a formalism such as SLP, they ask a variety of questions pertaining both to the expressive power of the formalism, and to logical properties of its semantics. The question of expressibility has been studied early and it turned out that the problem of existence of stable models of *finite* propositional logic programs (i.e. finite collections of propositional clauses) is an NP-complete problem. For infinite programs (and in particular for finite programs admitting function symbols -such programs are interpreted as their *ground versions*) the SLP turned out to be overexpressive [22]. That is very complex sets of integers, more complex than anything considered computable, are expressed by such models (see also remarks below). This resulted in very significant limitations for ASP.

One question that the logicians will ask when dealing with a formalism is its compactness. This question has been asked in case of SLP by M. Gelfond. Motivated by this question we quickly found counterexamples that will be presented in the next Section. But a natural way to look at negative answers is to ask if general properties that are not true, can be approximated by some other properties. In the case of our problem, the issue was this: *compactness for* SLP *does not hold in general; are there some classes of programs where compactness is guaranteed*? In fact, one such class has been found by Bonatti [5]. Those are programs which split into two parts, lower and upper in such a way that the upper part reduces to a Horn program on every stable model of the lower part. Here we will discuss three different syntactically motivated classes of programs with the compactness property. We introduce a proof-theoretic techniques for handling stable models. This technique uses *proof schemes*, an extension of the concept of proof to non-monotonic context. The main difference betwee the proof schemes and proofs as studied in proof theory is that proof schemes have guards

[1] We did not discuss here the issue of unification and its role in PROLOG. We limit ourselveds to the propositional case, although we admit infinite programs.

(we call them *supports*) that allow the proof to be executed. The support is a set of atoms that need to be *absent* in the context to execute the proof scheme. We can consider inclusion-minimal supports, and distinguish programs which we call finite-support programs which have only finite number of such minimal supports. One of our results is that such programs are characterized by the continuity of so-called Gelfond-Lifschitz operator, and that these programs do have compactness property. The second class of programs is based on existence of schemes of a certain form. This class of programs, called locally-finite programs has been introduced by Cenzer and Remmel in [6]. We show that the programs in this class also have compactness property. The third class considered in this paper is based on Scott's notion of consistency property. Again we find that this is syntactic property (different from the previous two) that entail compactness. We discuss the ramifications in the Conclusions section.

2 Motivating examples

The Compactness Theorem for propositional logic says that if Θ is a collection of sentences and every finite subset of Θ has a model, then Θ has a model. In this paper, we study the analogue of compactness for the stable logic semantics of propositional logic programs. At first glance, there are some obvious differences between stable models of propositional logic programs and models of sets of sentences in a propositional logic. For example, if T is a set of sentences in a propositional logic and $S \subseteq T$, then it is certainly the case that every model of T is a model of S. Thus a set of propositional sentences T has the property that if T has a model, then every subset of T has a model. This is certainly not true for propositional logic programs. For example, consider the following logic example.

Example 2.1. Let P consists of the following two clauses:

$C_1 = a \leftarrow \neg a, \neg b$ and
$C_2 = b \leftarrow$

Then it is easy to see that $\{b\}$ is stable model of P. However the subprogram P' consisting of just clause C_1 does not have a stable model. That is b is not in any stable model of P' since there is no clause in P' whose head is b. Thus the only possible stable models of P' would be $M_1 = \emptyset$ and $M_2 = \{a\}$. But it is easy to see that both M_1 and M_2 are not stable models of P'. $\triangle$

Thus the best that we could hope for as an analogue of compactness for propositional logic programs would be the following principle which we will call *Comp*:

(*Comp*) *If for any finite program $P' \subseteq P$, there exist a finite program P'' such that $P' \subseteq P'' \subseteq P$ such that P'' has a stable model, then P has a stable model.*

Our next example, will show that (Comp) is not true for propositional logic programs.

Example 2.2. Let P be an infinite propositional program consisting of the following clauses.

1. $a \leftarrow \neg a, \neg b$
2. $b \leftarrow \neg c_i$, for all $i \geq 0$
3. $c_i \leftarrow$ for all $i \geq 0$.

Note that P has no stable model. That is, if M is a stable model of P, then $c_i \in M$ for all i by the clauses in the group (3). Thus $b \notin M$ since the only way to derive b is from one of the clauses in the group (2) and these are all blocked for M. Now consider a. We cannot have $a \in M$ since the only way to derive a is via the clause (1) but it would be blocked if $a \in M$. Thus $a \notin M$. However if $a \notin M$ and $b \notin M$, then we must have $a \in M$ by clause (1). This is a contradiction so that P has no stable model.

Now we fix $n \geq 0$ and consider a finite subprogram P_n of P. consisting of the following clauses.

1. $a \leftarrow \neg a, \neg b$
2. $b \leftarrow \neg c_i$, for all $0 \leq i \leq n+1$
3. $c_i \leftarrow$, for all $0 \leq i \leq n$.

It is easy to see that P_n has an exactly one stable model, namely $\{b, c_1, \ldots, c_n\}$. Moreover it is easy see that any finite subset F of P is contained in some P_n. Therefore the principle (*Comp*) fails for P. △

Since the analogue of compactness fails for propositional logic programs, it is natural to ask if there are certain conditions (*Cond*) such that whenever a propositional logic program statisfies (*), then (*Comp*) holds. In turns out that there are several conditions that have appeared in the literature that ensure that (*Comp*) holds. The main goal of this paper is survey those conditions.

3 Stable Models and Proof Schemes

Before we can state some conditions which ensure property (*Comp*), we need to formally introduce stable models and the concept of proof schemes.

For an introductory treatment of logic programs, see [1]. Here is a brief self-contained account of their stable models [12]. Let us assume as given a fixed first order language $\mathcal{L}$ based on predicate letters, constants, and function symbols. The Herbrand base of the language is defined as the set $B_{\mathcal{L}}$ of all ground atoms of the language $\mathcal{L}$. A literal is an atom or its negation, a ground literal is an a ground atom or its negation. A logic program P is a set of "program clauses", that is, an expression of the form:

$$p \leftarrow l_1, \ldots, l_k \tag{1}$$

where p is an atom, and $l_1, \ldots, l_k$ is a list of literals.

Then p is called the conclusion of the clause, the list $l_1, \ldots, l_k$ is called the body of the clause. Ground clauses are clauses without variables. Horn clauses are clauses with no negated literals, that is, with atomic formulas only in the body. We will denote by $Horn(P)$ the part of the program P consisting of its Horn clauses. Horn clause programs are programs P consisting of Horn clauses. That is for Horn programs P, $P = Horn(P)$. Each such program has a least model in the Herbrand base determined as the least fixed point of a continuous operator T_P representing 1-step Horn clause logic deduction ([17]).

Informally, the *knowledge* of a logic program P is the set of clauses in P with no negated literals in the bodies, that is, the Horn clauses. The set of beliefs of a logic program P is the set of clauses with negated literals occurring in the bodies of clauses of P. This use of language is sufficiently suggestive to guide the reader to translations of many nonmonotone theories in other reasoning systems into

equivalent logic programs so that extensions as models for the non-monotonic theory correspond to stable models as models for the logic program.

A ground instance of a clause is a clause obtained by substituting ground terms (terms without variables) for all variables of the clause. The set of all ground instances of the program P is called $ground(P)$.

Let M be any subset of the Herbrand base. A ground clause is said to be M-*applicable* if the atoms whose negations are literals in the body are not members of M. Such clause is then *reduced* by eliminating remaining negative literals. This *monotonization* $GL(P, M)$ of P with respect to M is the propositional Horn clause program consisting of reducts of M-applicable clauses of $ground(P)$ (see Gelfond-Lifschitz [12]). Then M is called a *stable model* for P if M is the least model of the Horn clause program $GL(M, P)$. We denote this least model as $F_P(M)$. It is easy to see that a stable model for P is a minimal model of P ([12]). We denote by $Stab(P)$ the set of all stable models of P. There may be no, one, or many stable models of P.

We should note that the syntactical condition of stratification of Apt, Blair, and Walker [3] singles out programs with a well-behaved, unique stable model, but there is no reason to think that in belief revision one could move from stratified program to stratified program; but how one might do this is an interesting and challenging question.

What kind of proof theory is appropriate for logic programs? The key idea for our proofs is that of a proof scheme with conclusion an atom p. Proof schemes are intended to reflect exactly how p is a finitary but non-monotonic consequence of P.

Proof schemes represent a form of resolution proofs tailored to fit non-monotonic proofs. Two items distinguis them from the resolution proof trees. first, the derivation process is represented in a linear fashion, as sequences. Second, the negative information, controlling applicability of proof schemes is represented explicitely.

A proof scheme uses, as in Horn logic, the positive information present in the positive literals of bodies of clauses of P, but proof schemes also have to respect the negative information present in the negative literals of bodies of clauses. With this motivation, here is the definition. A *proof scheme* for p with respect to P is a sequence of triples $< \langle p_l, C_l, S_l \rangle >_{1 \leq l \leq n}$, with n a natural number, such that the following conditions all hold.

1. Each p_l is in $B_{\mathcal{L}}$. Each C_l is in $ground(P)$. Each S_l is a finite subset of $B_{\mathcal{L}}$.
2. p_n is p.
3. The S_l, C_l satisfy the following conditions. For all $1 \leq l \leq n$, one of **(a)**, **(b)**, **(c)** below holds.
   - **(a)** C_l is $p_l \leftarrow$, and S_l is S_{l-1},
   - **(b)** C_l is $p_l \leftarrow \neg s_1, \ldots, \neg s_r$ and S_l is $S_{l-1} \cup \{s_1, \ldots, s_r\}$, or
   - **(c)** C_l is $p_l \leftarrow p_{m_1}, \ldots, p_{m_k}, \neg s_1, \ldots, \neg s_r$, $m_1 < l$,...,$m_k < l$, and S_l is $S_{l-1} \cup \{s_1, \ldots, s_r\}$.

(We put $S_0 = \emptyset$).

The atoms p_i occur in a proof schem in the order they are derived. Because an atom may be the head of several rules, we also list the specific rule C_i that is used to derive p_i. The sets S_i control the applicability of proof scheme. The idea here is that, unlike usual proofs, the proof schemes carry within themself the information on its applicability with respect to potential stable models. To have the scheme applicable with respect to M, the support of that scheme (S_n) must be disjoint from M.

Let us suppose that $\varphi =< \langle p_l, C_l, S_l \rangle >_{1 \leq l \leq n}$ is a proof scheme. Then $conc(\varphi)$ denotes atom p_n and is called the *conclusion* of φ. Also, $supp(\varphi)$ is the set S_n and is called the *support* of φ.

Condition (3) tells us how to construct the S_l inductively, from the S_{l-1} and the C_l. The set S_n consists of the negative information of the proof scheme.

A proof scheme may not need all its lines to prove its conclusion. It may be possible to omit some clauses and still have a proof scheme with the same conclusion. If we omit as many clauses as possible, retaining the conclusion but still maintaining a proof scheme, this is a *minimal proof scheme* with that conclusion. It may be possible to do this with many distinct results, but obviously there are only a finite number of ways altogether to trim a proof scheme to a minimal proof scheme with the same conclusion, since no new clauses are ever introduced. Of course, a given atom may be the conclusion of no, one, finitely many, or infinitely many different minimal proof schemes. These differences are clearly computationally significant if one is searching for a justification of a conclusion. The apparatus needed to discuss this was introduced in [18].

Formally, we preorder proof schemes φ, ψ by $\varphi \prec \psi$ if

1. φ, ψ have same conclusion,
2. Every clause in φ is also a clause of ψ.

The relation $\prec$ is reflexive, transitive, and well-founded. Minimal elements of $\prec$ are called minimal proof schemes.

Here are some propositions from [18, 19].

Proposition 3.1. *Let P be a program and $M \subseteq B_{\mathcal{L}}$. Let p be an atom. Then p is in $F_P(M)$ if and only if there exists a proof scheme with conclusion p whose support is disjoint from M.*

If Z is a set of atoms we let $\neg Z$ be the conjunction of all the negations of atoms of Z. Now we fix program P and atom p for the discussion. Associate with the atom p a (possibly infinitary) Boolean equation E_p

$$p \leftrightarrow (\neg Z_1 \vee \neg Z_2 \vee \ldots), \tag{2}$$

where the $Z_1, Z_2 \ldots$ is a (possibly infinite) list of supports of all minimal proof schemes with conclusion p with respect to P. In fact, for our purposes it is enough to list only the inclusion-minimal supports. This is called a *defining equation* for p with respect to P. If there are infinitely many distinct minimal supports for proof schemes with conclusion p, this will be an infinitary equation. We make two other conventions about the defining equation of p. Namely

1. If p is not the conclusion of any proof scheme with respect to P, then the defining equation for p is $p \leftrightarrow \bot$, which is equivalent to $\neg p$. Hence in this case, $\neg p$ must hold in every stable model of P.
2. If p has a proof scheme with empty support, that is, a proof scheme which uses only Horn clauses, then the defining equation for p is equivalent to $\top$. In this case, p belongs to all stable models of P.

The set Eq_P of all equations E_p obtained as p ranges over the Herbrand base is called a defining system of equations for program P.

Example 3.1. Let P be a program:

$p(0) \leftarrow \neg q(X)$
$nat(0) \leftarrow$
$nat(s(X)) \leftarrow nat(X)$.

Then for each n, $< \langle p(0), p(0) \leftarrow \neg q(s^n(0)), \{q(s^n(0)\}\rangle >$ is a minimal proof scheme with conclusion $p(0)$. Thus atom $p(0)$ has an infinite number of minimal proof schemes with respect to program P. $\triangle$

The defining equations characterize stable models of logic programs.

Proposition 3.2. *Let P be a logic program with defining system of equations Eq_P. Let M be a subset of the Herbrand universe $B_{\mathcal{L}}$. Then M is a stable model for P if and only if $M \cup \{\neg q\colon q \in B_{\mathcal{L}} \setminus M\}$ is a solution of the system Eq_P.*

Here is another characterization of stable models, this time via proof schemes.

Proposition 3.3. *Let P be a program. Also, suppose that M is a subset of the Herbrand universe $B_{\mathcal{L}}$. Then M is a stable model of P if and only if, for every $p \in B_{\mathcal{L}}$, it is true that p is in M if and only if there exists a proof scheme φ with conclusion p such that the support of φ is disjoint from M.*

Given a logic program P, Dung and Kanchanasut in [10] introduce a construction of a purely negative propositional program P' (that is a program consisting of clauses of the form $p \leftarrow \neg q_1, \ldots, \neg q_n$, $n \geq 0$, where $p, q_1, \ldots q_n$ are ground atoms) with the property that P and P' have the same stable models. It is very easy to construct P' out of the set of proof schemes (minimal proof schemes are sufficient for that purpose). Namely, let, for a given $p \in B_{\mathcal{L}}$, $Z_1^p, Z_2^p, \ldots$ list the supports of all minimal proof schemes for p in P. Then the set of purely negative clauses

$$\{p \leftarrow \neg Z_i^p : p \in B_{\mathcal{L}}, i \in n\}$$

is a purely negative program P' with the desired property that $Stab(P) = Stab(P')$.

4 FSP logic programs and their continuity properties

Our first condition that ensures a program P has property (*Comp*) is the notion of *finitary support programs* introduced by Marek, Nerode, and Remmel [22]. That is, Marek, Nerode, and Remmel examined logic programs P such that every defining equation for every atom p is finite. We will use show in the next section that FSP indeed the property *Comp*. In this section we show continuity property of an operator associated with FSP logic programs.

The FSP property is equivalent to requiring that every atom has only a finite number of inclusion-minimal supports of minimal proof schemes. Such a program may have the property that there is an atom which is the conclusion of infinitely many different minimal proof schemes, but these schemes have only finitely many supports altogether among them.

Example 4.1. Let P be the program:

$p(0) \leftarrow q(X)$
$q(X) \leftarrow \neg r(0)$
$nat(0) \leftarrow$
$nat(s(X)) \leftarrow nat(X)$.

Then the atom $p(0)$ is the conclusion of infinitely many proof schemes:

$$< \langle q(s^n(0)), q(s^n(0)) \leftarrow \neg r(0), \{r(0)\}\rangle, \langle p(0), p(0) \leftarrow q(s^n(0)), \{r(0)\}\rangle >$$

as n ranges over ω. But the single minimal support of all these proof schemes is $\{r(0)\}$. That is, whenever $r(0)$ is not in M, then $p(0)$ will be in $F_P(M)$. $\triangle$

The above program motivates the following definition.

Definition 4.1. *We say that a* finitary support *program (*FSP *program, for short) is a logic program such that for every atom p, there is a finite set of finite sets S, which are exactly the inclusion-minimal supports of all those minimal proof schemes with conclusion p.*

We now study the FSP property. It turns out that this property is equivalent to the continuity property for a suitably defined operator. This is precisely the same operator whose square (that is two-fold application) determines the *monotonic* operator whose least and largest fixpoints determine the well-founded model of the program ([26]).

We associate an operator with each logic program as follows.

Definition 4.2. *Let P be a program. The operator $F_P : \mathcal{P}(B_\mathcal{L}) \to \mathcal{P}(B_\mathcal{L})$ is defined as follows: If $S \subseteq B_\mathcal{L}$ then $F_P(S)$ is the set of all atoms in $B_\mathcal{L}$ for which there exists a proof scheme p such that $supp(p) \cap S = \emptyset$. Thus F_P assigns to S the set $F_P(S)$.*

Proposition 4.3. *The operator F_P is anti–monotonic, that is, if $S_1 \subseteq S_2$, then $F_P(S_2) \subseteq F_P(S_1)$.*

Proposition 4.4. *The operator F_P is lower half-continuous; that is, if $\langle S_n\rangle_{n\in\omega}$ is a monotone decreasing sequence of subsets of $B_\mathcal{L}$ then $\bigcup_{n\in\omega} F_P(S_n) = F_P(\bigcap_{n\in\omega} S_n)$.*

Proposition 4.5. *Let P be a logic program. Then following conditions are equivalent:*
(a) P is an FSP logic program.
(b) F_P is an upper half-continuous operator; that is, whenever $\langle S_n\rangle_{n\in\omega}$ is a monotone increasing sequence of subsets of $B_\mathcal{L}$, we have

$$\bigcap_{n\in\omega} F_P(S_n) = F_P(\bigcup_{n\in\omega} S_n)$$

5 Coding Stable Models into Trees

The key result to prove that at FSP programs have property (*Comp*) is the proof of a result of Marek, Remmel, and Nerode [19, 22] to the effect that for any recursive program P, there is recursive tree $T_P \subseteq \omega^\omega$ such that there is an effective one-one degree preserving correspondence between the set of stable models of P, $Stab(P)$, and the set of infinite paths throught T_P, $Path(T_P)$.

In this section, we shall give the neccessary recursion theoretic background to make precise and to prove the result of Marek, Nerode, and Remmel that given any recursive logic program P, there is a recursive tree T such that there is an effective 1-1 degree-preserving map between the set of stable models of P and the set of paths through T. Then we shall show that the fact that condition (*Comp*) holds for FSP programs follows from their proof.

5.1 Recursive programs

When we discuss finite programs then we can easily read off a recursive representation of the Herbrand base. The reason is that the alphabet of such a program, that is, the set of predicate symbols

and function symbols that appear in the program, is finite. The situation changes when P is an infinite predicate logic program representable with a recursive set of Gödel numbers. When we read off the enumeration of the alphabet of the program from an enumeration of the program itself, there is no guarantee that the alphabet of P is recursive. In particular the Herbrand base of the program is recursively enumerable but may not necessarily be recursive.

For the purposes of this paper, we define a program P to be recursive if not only the set of its Gödel numbers is recursive, but also the resulting representation of the Herbrand base is recursive.

5.2 Recursively FSP programs

A *recursively* FSP program is an FSP recursive program such that we can uniformly compute the finite family of supports of proof schemes with conclusion p from p. The meaning of this is obvious, but we need a technical notation for the proofs. Start by listing the whole Herbrand base of the program, $B_{\mathcal{L}}$ as a countable sequence in one of the usual effective ways. This assigns an integer (Gödel number) to each element of the base, its place in this sequence. This encodes finite subsets of the base as finite sets of natural numbers, all that is left is to code each finite set of natural numbers as a single natural number, its *canonical index*. To the finite set $\{x_1, \ldots, x_k\}$ we assign as its canonical index $can(\{x_1, \ldots, x_k\}) = 2^{x_1} + \ldots + 2^{x_k}$. We also set $can(\emptyset) = 0$. If program P is FSP, and the list, in order of magnitude, of Gödel numbers of all minimal support of schemes with conclusion p is

$$Z_1^p, \ldots, Z_{l_r}^p,$$

then we define a function $su^P \colon B_{\mathcal{L}} \to \omega$ as below.

$$p \mapsto can(\{can(Z_1^p), \ldots, can(Z_{l_r}^p)\})$$

We call a logic program P a *recursively* FSP program if it is FSP and the function su^P is recursive.

5.3 Tools from Recursion Theory

Let $\omega = \{0, 1, 2, \ldots\}$ denote the set of natural numbers and let $<,> \colon \omega \times \omega \to \omega - \{0\}$ be some fixed one-to-one and onto recursive pairing function such that the projection functions π_1 and π_2 defined by $\pi_1(< x, y >) = x$ and $\pi_2(< x, y >) = y$ are also recursive. We extend our pairing function to code n-tuples for $n > 2$ by the usual inductive definition, that is $< x_1, \ldots, x_n >=< x_1, < x_2, \ldots, x_n >>$ for $n \geq 3$. We let $\omega^{<\omega}$ denote the set of all finite sequences from ω and $2^{<\omega}$ denote the set of all finite sequences of 0's and 1's. Given $\alpha = (\alpha_1, \ldots, \alpha_n)$ and $\beta = (\beta_1, \ldots, \beta_k)$ in $\omega^{<\omega}$, we write $\alpha \sqsubseteq \beta$ if α is initial segment of β, that is, if $n \leq k$ and $\alpha_i = \beta_i$ for $i \leq n$. For the rest of this paper, we identify a finite sequence $\alpha = (\alpha_1, \ldots, \alpha_n)$ with its code $c(\alpha) =< n, < \alpha_1, \ldots, \alpha_n >>$ in ω. We let 0 be the code of the empty sequence $\emptyset$. Thus, when we say a set $S \subseteq \omega^{<\omega}$ is recursive, recursively enumerable, etc., we mean the set $\{c(\alpha) \colon \alpha \in S\}$ is recursive, recursively enumerable, etc. A *tree* T is a nonempty subset of $\omega^{<\omega}$ such that T is closed under initial segments. A function $f \colon \omega \to \omega$ is an infinite *path* through T if for all n, $(f(0), \ldots, f(n)) \in T$. We let $[T]$ denote the set of all infinite paths through T. A set A of functions is a Π_1^0-class if there is a recursive predicate R such that $A = \{f \colon \omega \to \omega : \forall_n(R(< n, < f(0), \ldots, f(n-1) >>)\}$. A Π_1^0-class A is *recursively bounded* if there is a recursive function $g \colon \omega \to \omega$ such that $\forall_{f \in A} \forall_n (f(n) \leq g(n))$. It is not difficult to see that if A is a Π_1^0-class, then $A = [T]$ for some recursive tree $T \subseteq \omega^{<\omega}$. We say that a tree $T \subseteq \omega^{<\omega}$ is *highly recursive* if T is a recursive, finitely branching tree such that there is a recursive procedure which given $\alpha = (\alpha_1, \ldots, \alpha_n)$ in T produces a canonical index of the set of immediate successors of α in T, that is, produces a canonical index of $\{\beta = (\alpha_1, \ldots, \alpha_n, k) \colon \beta \in T\}$. If A is a recursively bounded

Π^0_1-class, then $A = [T]$ for some highly recursive tree $T \subseteq \omega^{<\omega}$, see [15]. We let A' denote the jump of the set A and $\mathbf{0'}$ denote the jump of the empty set. Thus $\mathbf{0'}$ is the degree of any complete r.e. set. We say that a tree $T \subseteq \omega^{<\omega}$ is *highly recursive in* $\mathbf{0'}$ if T is a finitely branching tree such that T is recursive in $\mathbf{0'}$ and there is an effective procedure which given an $\mathbf{0'}$-oracle and an $\alpha = (\alpha_1, \ldots, \alpha_n)$ in T produces a canonical index of the set of immediate successors of α in T, that is, produces a canonical index of $\{\beta =< \alpha_1, \ldots, \alpha_n, k) \colon \beta \in T\}$.

We say that there is an effective one-to-one degree preserving correspondence between the set of stable models of a recursive program P, $Stab(P)$, and the set of infinite paths $[T]$ through a recursive tree T if there are indices e_1 and e_2 of oracle Turing machines such that
(i) $\forall_{f \in [T]} \{e_1\}^{gr(f)} = M_f \in Stab(P)$,
(ii) $\forall_{M \in Stab(P)} \{e_2\}^M = f_M \in [T]$, and
(iii) $\forall_{f \in [T]} \forall_{M \in Stab(P)} (\{e_1\}^{gr(f)} = M$ if and only if $\{e_2\}^M = f)$.
Here $\{e\}^B$ denotes the function computed by the e^{th} oracle machine with oracle B. We write $\{e\}^B = A$ for a set A if $\{e\}^B$ is a characteristic function of A. If f is a function $f \colon \omega \to \omega$, then $gr(f) = \{< x, f(x) > \colon x \in \omega\}$. Condition (i) says that the infinite paths of the tree T, uniformly produce stable models via an algorithm with index e_1. Condition (ii) says that stable models of P uniformly produce branches of the tree T via an algorithm with index e_2. A is *Turing reducible* to B, written $A \leq_T B$, if $\{e\}^A = B$ for some e. A is *Turing equivalent* to B, written $A \equiv_T B$, if both $A \leq_T B$ and $B \leq_T A$. Thus condition (iii) asserts that our correspondence is one-to-one and if $\{e_1\}^{gr(f)} = M_f$, then f is Turing equivalent to M_f. Finally, given sets A and B, we let $A \oplus B = \{2x : x \in A\} \bigcup \{2x + 1 : x \in B\}$.

5.4 Representing programs by trees

Theorem 5.1. *We suppose that the first order language $\mathcal{L}$ has infinitely many ground atoms.*

1. *Then for any recursive program P in $\mathcal{L}$, there exists a recursive tree $T \subseteq \omega^{<\omega}$ and an effective one-to-one degree preserving correspondence between the set of all stable models of P, $Stab(P)$ and $[T]$, the set of all infinite paths through T.*
2. *If, in addition to the hypothesis of (1), program P is FSP, then the tree T is finite splitting.*
3. *If, in addition to the hypothesis of (2), program P is recursively FSP, then the tree T is a highly recursive tree.*

Theorem 5.1 allows us to prove that the class of denumerable *FSP* has the property)*Comp*).

Theorem 5.2. *Let us suppose that P is a countable FSP program. Then P satisifies property (Comp). That is, if for any finite program $P' \subseteq P$, there exist a finite program P'' such that $P' \subseteq P'' \subseteq P$ such that P'' has a stable model, then P has a stable model.*

6 Locally Determined Logic Programs

In this section, we shall describe another condition on propositional logic programs that will ensure property (*Comp*) that has appeared in the literature. Namely, we shall introduce the notion of locally determined logic programs due to Cenzer and Remmel [6]. For the rest of this paper, we shall only consider countable logic programs P. Thus whenever we say that P is a logic program, we shall always assume that P is countable.

The informal notion of a locally determined logic program P is one in which the existence of a proof scheme for an atom a_i (or the lack of existence thereof) can be determined by examining only clauses or proof schemes involving some initial segment of the Herbrand base of P. More formally, we fix some countable logic program P and some listing $a_0, a_1, \ldots$ of the atoms of Herbrand base of P without repetitions. (We shall make the convention that if P is a recursive logic program, then there is some recursive function $h : \omega \to \omega$ such that $h(i) = a_i$.) Then given a proof scheme or a clause ψ, we write $max(\psi)$ for the $max(\{i : a_i \text{ occurs in } \psi\})$. We shall write P_n for the set of all clauses $C \in P$ such that $max(C) \leq n$ and let $A_n = \{a_0, \ldots, a_n\}$.

Definition 6.1. *We shall say that n is a level of P if for all $S \subseteq \{a_0, \ldots, a_n\}$ and all $i \leq n$, whenever there exists a proof scheme ϕ such that $cln(\phi) = a_i$ and $supp(\phi) \cap S = \emptyset$, then there exists a proof scheme ψ such that $cln(\psi) = a_i$, $supp(\psi) \cap S = \emptyset$ and $max(\psi) \leq n$. Note that by definition, the Herbrand base H_{P_n} of P_n is contained in A_n. We let $lev(P) = \{n : n \text{ is a level of } P\}$.*

The following result has essentially been proven in [9, 16]

Theorem 6.2. *Suppose that n is a level of P and E is a stable model of P. Then $E_n = E \cap \{a_0, \ldots, a_n\}$ is a stable model of P_n.*

Definition 6.3. *We shall say that a logic program P is* **locally determined** *if P is countable and there are infinitely many n such that n is a level of P.*

Example 6.1. Let P be the infinite logic program with the following set of clauses.

1. $a_{2i} \leftarrow \neg a_{2i+1}$, for all $i \in \omega$
2. $a_{2i+1} \leftarrow \neg a_{2i}$, for all $i \in \omega$.

Thus the Herbrand base of P is $\{a_0, a_1, \ldots\}$. It is easy to see that S is stable model of P if and only if $|S \cap \{a_{2i}, a_{2i+1}\}| = 1$ for all $i \in \omega$. In fact, one can easily prove that $lev(P) = \{2i + 1 : i \in \omega\}$. Moreover, it is clear that P is locally finite. $\triangle$

Example 6.2. Let Q be the logic program with the following set of clauses for all $i \in \omega$.

1. $a_{3i} \leftarrow \neg a_{3i+1}, \neg a_{3i-2}, \ldots \neg a_1, \neg a_{3i+2}, \neg a_{3i-1}, \ldots \neg a_2$
2. $a_{3i+1} \leftarrow \neg a_{3i}, \neg a_{3i-3}, \ldots \neg a_0, \neg a_{3i+2}, \neg a_{3i-1}, \ldots \neg a_2$
3. $a_{3i+2} \leftarrow \neg a_{3i}, \neg a_{3i-3}, \ldots \neg a_0, \neg a_{3i+1}, \neg a_{3i-2}, \ldots \neg a_1$
4. $a_{3i} \leftarrow a_{3i+3}$
5. $a_{3i+1} \leftarrow a_{3i+4}$
6. $a_{3i+2} \leftarrow a_{3i+5}$

The Herbrand base of Q is $\{a_0, a_1, \ldots\}$. It is easy to see that P has exactly 3 stable models, namely, $S_0 = \{a_{3i} : i \in \omega\}$, $S_1 = \{a_{3i+1} : i \in \omega\}$ and $S_2 = \{a_{3i+2} : i \in \omega\}$. In this case, Q is not locally finite since for any $i > 0$, the following set of clauses can be used to construct a minimal proof scheme of a_0 with support equal to

$$\{a_{3i+1}, a_{3i-2}, \ldots, a_1, a_{3i+2}, a_{3i-1}, \ldots, a_2\}.$$

$a_{3i} \leftarrow \neg a_{3i+1}, \neg a_{3i-2}, \ldots \neg a_1, \neg a_{3i+2}, \neg a_{3i-1}, \ldots \neg a_2$

$a_{3i-3} \leftarrow a_{3i}$

$a_{3i-6} \leftarrow a_{3i-3}$

$\vdots$

$a_0 \leftarrow a_3$.

However we claim that $lev(P) = \{3i + 2 : i \in \omega\}$. That is, let us fix $n \geq 0$ and let us suppose that $T \subseteq \{a_i : i \leq 3n + 2\}$ and let us suppose that ψ is a proof scheme with conclusion a_r where $supp(\psi) \cap T = \emptyset$ and $r \leq 3n + 2$. We shall consider three cases.
Case 1. $T = \emptyset$.
In this case it is easy to see that every element of $\{a_i : i \leq 3n+2\}$ which is the conclusion of a proof scheme of P_{3n+2} of length one using one of the clauses (1), (2), and (3).
Case 2. There exist a_{3i+s} and a_{3j+t} in T where $s \neq t$ and $s, t \in \{0, 1, 2\}$.
In this case all clauses of the form (1), (2) or (3) where the head of the clause is some a_k with $k > 3n + 2$ cannot be part of a proof scheme ψ such that $supp(\psi) \cap T = \emptyset$. Thus the only clauses of the form (1), (2), or (3) that can be part of ψ are clauses from P_{3n+2}. However, in that case, the only clauses of the form (4), (5), and (6) that can be part of ψ must also be in P_{3n+2} because there is no way that we can derive an element a_k with $k \geq 3n + 2$ that is in the body of a clause of the form (4), (5), and (6) if we can only use clauses in ψ of type (1), (2), and (3) in ψ from P_{3n+2}. Here we are using the fact that ψ is a minimal proof scheme. It follows that ψ must be a proof scheme for P_{3n+2}.
Case 3. Conditions of Case 1 or Case 2 do not hold.
In this case, T must be contained in one of the stable models S_0, S_1, or S_2. We shall assume that $T \subseteq S_0$ since the other two cases are similar. Since $T \cap S_0 \neq \emptyset$, there can be no clause of type (2) and (3) in ψ which in not in P_{3n+2}. Now we suppose that a clause of type (3) occurs in ψ with head a_{3j} where $j > n$. Then this clause combined with clauses of type (4) in ψ can be used to derive elements of the form a_{3i} with $i \leq n$. But for all $i \leq n$, we can clearly immediately derive a_{3i} form the clause

$$a_{3i} \leftarrow \neg a_{3i+1}, \neg a_{3i-2}, \ldots, \neg a_1, \neg a_{3i+2}, \neg a_{3i-1}, \ldots \neg a_2 \tag{3}$$

which lies in P_{3n+2} and whose constraints do not intersect T. It follows that we can construct an minimal proof scheme ψ' in P_{3n+2} with the same conclusion as ψ such that $supp(\psi') \cap T = \emptyset$.

It follows that $3n + 2$ is level of P for all n. Moreover it is clear from the clauses of type (1) and (2) that $3n$ and $3n + 1$ are not levels of P so that $lev(P) = \{3n + 2 : n \in \omega\}$ as claimed. $\triangle$

Example 6.3. In this example, we give a program which is very similar to example 6.2, but is not locally determined. Let R be the logic program with the following set of clauses. The parameter i ranges over ω in all six groups of clauses.

1. $a_{3i} \leftarrow \neg a_{3i+1}, \neg a_{3i+2}$
2. $a_{3i+1} \leftarrow \neg a_{3i}, \neg a_{3i+2}$
3. $a_{3i+2} \leftarrow \neg a_{3i}, \neg a_{3i+1}$
4. $a_{3i} \leftarrow a_{3i+3}$
5. $a_{3i+1} \leftarrow a_{3i+4}$
6. $a_{3i+2} \leftarrow a_{3i+5}$

Just as in example 6.2, the Herbrand base of R is $\{a_0, a_1, \ldots\}$ and it easy see that P has exactly 3 stable models, namely, $S_0 = \{a_{3i} : i \in \omega\}$, $S_1 = \{a_{3i+1} : i \in \omega\}$ and $S_2 = \{a_{3i+2} : i \in \omega\}$. In this case, R has no levels. Again it is easy to see that the clauses of the form (1) and (2) ensure that $3n$ and $3n + 1$ are not levels of P. However in this case, $3n + 2$ is also not a level of P. That is consider $T = \{a_{3n}, a_{3n+1}\}$. It is easy to see that there is no minimal proof scheme ψ of P_{3n+2} such that $supp(\psi) \cap T = \emptyset$ and the conclusion of ψ is a_{3n+2}. However the following is a minimal proof ψ' of P with conclusion a_{3n+2} such that $supp(\psi') \cap T = \emptyset$.

$$\langle\langle a_{3n+5}, a_{3n+5} \leftarrow \neg a_{3n+3}, \neg a_{3n+4}, \{a_{3n+3}, a_{3n+4}\}\rangle\langle a_{3n+2}, a_{3n+2} \leftarrow a_{3n+5}\rangle, \{a_{3n+3}, a_{3n+4}\}\rangle\rangle.$$

△

Example 6.4. Suppose that we are given a set $L = \{l_0 < l_1 < \ldots\}$. Then we can construct a program P such that $H_P = \{a_0, a_1, \ldots\}$ and $lev(P) = L$ as follows. Let $l_{-1} = -1$. Then for each $n \geq 0$, we add the clause

$$a_{l_n} \leftarrow$$

to P if $l_n - l_{n-1} = 1$. Otherwise we add the following clauses to P

$$a_{l_{n-1}+k} \leftarrow \neg a_{l_{n-1}+1}, \ldots, \neg a_{l_{n-1}+k-1}, \neg a_{l_{n-1}+k+1}, \ldots, \neg a_{l_{n-1}+(l_n - l_{n-1})}$$

for all $k = 1, \ldots, l_n - l_{n-1}$. △

Definition 6.4. *Suppose that P is a recursive logic program. Then we say that P is* **effectively locally determined** *if P is locally determined and there is a recursive function f such that for all i, $f(i) \geq i$ and $f(i)$ is a level of P.*

In [22], Marek, Nerode, and Remmel showed that the problem of finding a stable model of a locally finite recursive logic program can be reduced to finding an infinite path through a finitely branching recursive tree and the problem of finding a stable model of a highly recursive logic program can be reduced to finding an infinite path through a highly recursive tree. A locally determined logic program is not always locally finite since it is possible that a given atom has infinitely many proof schemes which involves arbitrarily large atoms as in Example 6.2 above. Vice versa, it is possible to give examples of locally finite logic programs which is not locally determined. Nevertheless, we shall see that we get similar results to those of Marek, Nerode, and Remmel for locally determined and effectively locally determined recursive logic programs.

Theorem 6.5. *Let P be a recursive logic program.*

1. *If P is locally determined, then there is a recursive finitely branching tree T and a one-to-one degree preserving correspondence between the set of stable models $\mathcal{S}(P)$ of P and $[T]$ and*
2. *If P is effectively locally determined, then there is a highly recursive finitely branching tree T and a one-to-one degree preserving correspondence between the set set of stable models $\mathcal{S}(P)$ of P and $[T]$.*

A careful inspection of the tree T of Theorem **??** allows us to establish the following fact.

Corollary 6.6. *Suppose that P is a countable locally determined logic program such that there are infinitely many n such that P_n has a stable model E_n. Then P has a stable model.*

We also have the following.

Corollary 6.7. *Supose that P is infinite locally determined logic program, then P satisfies property (Comp).*

One can immediately apply a number of known results from the theory of recursively bounded Π^0_1 classes to derive corresponding results about the set of stable models of an effectively locally determined recursive logic program.

Corollary 6.8. *Suppose that P is an effectively locally determined recursive logic program which has at least one stable model. Then*

1. *P has a stable model whose Turing jump is recursive in $\mathbf{0}'$.*
2. *If P has no recursive stable model, then P has $2^{\aleph_0}$ stable models.*
3. *If P has only finitely many stable models, then each of these stable models is recursive.*
4. *There is a stable model E of P in an r.e. degree.*
5. *There exist stable models E_1 and E_2 of P such that any function, recursive in both E_1 and E_2, is recursive.*
6. *If P has no recursive stable model, then there is a nonzero r.e. degree $\mathbf{a}$ such that P has no stable model recursive in $\mathbf{a}$.*

A similar corollary holds for locally determined recursive logic programs where the statements in Corollary 6.8 are replaced by versions which are relativized to a $\mathbf{0}'$ oracle.

7 FC-normal logic programs

In this section with the notion of FC-normal logic programs as defined by Marek, Nerode, and Remmel [23]. These programs always have a stable model. In fact, they have an even stronger property that property (*Comp*). Namely, an FC-normal program P has the property that if P' is a finite subprogram of P which has a stable model E, then P has a stable model S which extends E.

Recall that $Horn(P)$ is the set of Horn clauses of a logic program P. We let $T_{Horn(P)}$ denote the one-step provability operator associated with $Horn(P)$, see [1]. That is, if $Q \subseteq H_P$, then $T_{Horn(P)}(Q)$ equals

$$\{p \in H_P : (\exists C = p \leftarrow q_1, \ldots q_m \in Horn(P))\ (q_1, \ldots, q_m \in Q)\}.$$

We call a family of subsets of H_P, Con, a **consistency property** over P if it satisfies the following conditions:

1. $\emptyset \in Con$.
2. If $A \subseteq B$ and $B \in Con$, then $A \in Con$.
3. Con is closed under directed unions.
4. If $A \in Con$ then $A \cup T_{Horn(P)}(A) \in Con$.

Conditions (1)-(3) are Scott's conditions for information systems. Condition (4) connects "consistent" sets of atoms to the Horn part of the program; if A is consistent then adding atoms provable from A preserves "consistency". The following fact is easy to prove.

Proposition 7.1. *If Con is a consistency property with respect to P and $A \in Con$, then $T_{Horn(P)} \Uparrow \omega(A) \in Con$.*

Here, for a Horn program Q, $T_Q \Uparrow \omega(A)$ is the cumulative fixpoint of T_Q over A. Proposition 7.1 says that our condition (4) in the definition of consistency property implies that the cumulative closure of a "consistent" set of atoms under $T_{H(P)}$ is still "consistent".

Given a consistency property, we define the concept of an **FC-normal** program with respect to that property. Here FC stands for "Forward Chaining".

Definition 7.2. (a) *Let P be a program, let Con be a consistency property with respect to P. Call P* **FC-normal with respect to** *Con if for every clause $C = p \leftarrow q_1, \ldots, q_n, \neg r_1, \ldots, \neg r_m$ such that $C \in ground(P) - ground(Horn(P))$ and every consistent fixpoint A of $T_{Horn(P)}$ such that $q_1, \ldots, q_n \in A$ and $p, r_1, \ldots, r_m \notin A$ we have*
(1) *$A \cup \{p\} \in Con$ and*
(2) *$A \cup \{p, r_i\} \notin Con$ for all $1 \leq i \leq m$.*
(b) *P is called* **FC-normal** *if there exists a consistency property Con such that P is FC-normal with respect to Con.*

Example 7.1. Let the Herbrand base consist of atoms a, b, c, d, e, f. Let the consistency property be defined by the following condition:
$A \notin Con$ if and only if either $\{c, d\} \subseteq A$ or $\{e, f\} \subseteq A$.

Now let us consider the following program.

(1) $a \leftarrow$
(2) $b \leftarrow c$
(3) $c \leftarrow b$
(4) $c \leftarrow a, \neg d$
(5) $e \leftarrow c, \neg f$

It is not difficult to check that this program is FC-normal with respect to the consistency property described above. Moreover, one can easily check that P possesses a unique stable model $M = \{a, b, c, e\}$.
If we add to this program the clause $f \leftarrow c, \neg e$, the resulting program is still FC-normal but now there are two stable models, $M_1 = \{a, b, c, e\}$ and $M_2 = \{a, b, c, f\}$. △

Marek, Nerode, and Remmel [23] showed that FC-normal normal programs have many of the properties that are possessed by normal default theories.

Theorem 7.3. *If P is an FC-normal program, then P possesses a stable model.*

Theorem 7.4. *If P is an FC-normal program with respect to the consistency property Con and $I \in Con$, then P possesses a stable model I' such that $I \subseteq I'$.*

Marek, Nerode, and Remmel proved Theorem 7.3 and 7.4 via a generally forward chaining algorithm which can be applied to FC-normal programs of any cardinality. Since in our case, we are dealing with only recursive and hence countable programs, we shall give only the countable version of their forward chaining construction. That is, let us suppose we fix some well-ordering $\prec$ of $ground(P) - ground(H(P))$ of order type ω. Thus, the well-ordering $\prec$ determines some listing of the clauses of $ground(P) - ground(H(P))$, $\{c_n : n \in \omega\}$. Their forward chaining construction then defines an increasing sequence of sets $\{T_n^{\prec}\}_{n \in \omega}$ in stages.

The countable forward chaining construction of $T^{\prec} = \bigcup_{n \in \omega} T_n^{\prec}$.

Stage 0. Let $T_0^{\prec} = T_{Horn(P)} \Uparrow \omega(\emptyset)$.

Stage $n+1$. Let $\ell(n+1)$ be the least $s \in \omega$ such that
$c_s = \varphi \leftarrow \alpha_1, \ldots, \alpha_k, \neg\beta_1, \ldots, \neg\beta_m$ where $\alpha_1, \ldots, \alpha_k \in T_n^{\prec}$ and
$\beta_1, \ldots, \beta_m, \varphi \notin T_n^{\prec}$. If there is no such $\ell(n+1)$, let $T_{n+1}^{\prec} = T_n^{\prec}$. Otherwise let

$$T_{n+1}^{\prec} = T_{Horn(P)} \Uparrow \omega(T_n^{\prec} \cup \{p_{\ell(n+1)}\})$$

where $p_{\ell(n+1)}$ is the head of $c_{\ell(n+1)}$.

Example 7.2. If we consider the final extended program of Example 7.1, it is easy to check that any ordering $\prec_1$ in which the clause $C_1 = e \leftarrow c, \neg f$ precedes the clause $C_2 = f \leftarrow c, \neg e$ will have $T^{\prec_1} = M_1$ while any ordering $\prec_2$ in which C_2 precedes C_1 will have $T^{\prec_2} = M_2$. $\triangle$

This given, Marek, Nerode, and Remmel proved the following results.

Theorem 7.5. *If P is a countable FC-normal program and $\prec$ is* any *well-ordering of $ground(P) - ground(H(P))$ of order type ω, then :*
(1) *$T^{\prec}$ is a stable model of P where $T^{\prec}$ is constructed via the countable forward chaining algorithm.*
(2) *(completeness of the construction). Every stable model model of P is of the form $T^{\prec}$ for a suitably chosen ordering $\prec$ of $ground(P) - ground(H(P))$ of order type ω where $T^{\prec}$ is constructed via the countable forward chaining algorithm.*

Theorem 7.6. *If P is an FC-normal logic program with respect to Con, then every stable model M of P is in Con.*

Theorem 7.7. *Let P be an FC-normal logic program with respect to a consistency property Con. Then if E_1 and E_2 are two distinct stable models of P, then $E_1 \cup E_2 \notin Con$.*

Given a logic program P and a stable model M, we let $NG(M, P)$, the set of non-Horn generating clauses of P be equal to the set of all clauses $c = \varphi \leftarrow \alpha_1, \ldots, \alpha_k, \neg\beta_1, \ldots, \neg\beta_m$ in $ground(P)$ such that $\alpha_1, \ldots, \alpha_k \in M$ and $\beta_1, \ldots, \beta_m \notin M$.

FC-normal programs possess the following key "semi-monotonicity" property.

Theorem 7.8. *Let P_1, P_2 be two programs such that $P_1 \subseteq P_2$ but $H(P_1) = H(P_2)$. Let us assume, in addition, that both are FC-normal with respect to the same consistency property. Then for every stable model M_1 of P_1, there is a stable model M_2 of P_2 such that:*
(1) *$M_1 \subseteq M_2$ and*
(2) *$NG(M_1, P_1) \subseteq NG(M_2, P_2)$.*

As mentioned in the introduction, a recursive FC-normal logic program P is guaranteed to have at least one relatively well behaved stable model

Theorem 7.9. *Suppose that P is a recursive logic program and P is FC-normal. Then P has a stable model S such that S is r.e. in $\mathbf{0}'$ and hence $E \leq_T \mathbf{0}''$.*

We observe that this should be contrasted with the following result from [22]

Proposition 7.10. *There exists a recursive logic program P such that P possesses a stable model, but no stable model of P is hyperarithmetic.*

We note that Theorem 7.9 is in some sense the best possible. That is, results from [23] show that the following holds. Given sets $A, B \subseteq \omega$, let $A \oplus B = \{2x : x \in A\} \cup \{2x+1 : x \in B\}$.

Theorem 7.11. *Let A be any r.e. set and B be any set which is r.e. in A, i.e. $B = \{x : \phi_e^A(x)\downarrow\}$. Then there is a recursive FC-normal logic program P such P has a unique stable model S and $S \equiv_T A \oplus B$. In particular, if B is any set which is r.e. in $\mathbf{0}'$ and $B \geq_T \mathbf{0}'$, then there is an FC-normal recursive logic program P such that P has a unique stable model S and $S \equiv_T B$.*

However if either $Horn(P)$ or $P - Horn(P)$ is finite, then one can improve on Theorem 7.9. That is, the following was proved in [23].

Theorem 7.12. *Let P be a FC-normal recursive logic program such that $P - Horn(P)$ is finite, then every stable model of $\mathcal{S}$ is r.e..*

We say that a recursive logic program P is **monotonically decidable** if for any finite set $F \subseteq \mathcal{H}(P)$, $T_{Horn(P)} \Uparrow \omega(F)$ is recursive and there is a uniform effective procedure to go from a canonical index of a finite set F to a recursive index of the $T_{Horn(P)} \Uparrow \omega(F)$, i.e. if there is a recursive function f such that for all k, $\phi_{f(k)}$ is the characteristic function of $T_{Horn(P)} \Uparrow \omega(D_k)$. It is easy to see that if $Horn(P)$ is finite, then the recursive program P is automatically monotonically decidable.

Theorem 7.13. *Let P be a recursive logic program such that P is FC-normal and monotonically decidable, then P has an stable model which is r.e.*

We end this section by giving complexity results for finite FC-normal logic program where the forward chaining algorithm runs in polynomial time.

For complexity considerations, we shall assume that the elements of H_P are coded by strings over some finite alphabet Σ. Thus every $a \in H_P$ will have some length which we denote by $||a||$. Next, for a clause

$$r = c \leftarrow a_1, \ldots, a_n, \neg b_1, \ldots, \neg b_m,$$

we define $||r|| = (\sum_{i \leq n} ||a_i||) + (\sum_{i \leq m} ||b_j||) + ||c||$. Finally, for a set Q of clauses, we define

$$||Q|| = \sum_{r \in Q} ||r||.$$

Theorem 7.14. *Suppose P is a finite FC-normal logic program and $\prec$ is some well-ordering of $P - Horn(P)$. Then $E^{\prec}$ as constructed via our forward chaining algorithm can be computed in time $O(||Horn(P)|| \cdot ||P - Horn(P)|| + ||P - Horn(P)||^2)$.*

We note that none of the theorems above make any explicit assumptions that the underlying consistency property of a recursive FC-normal logic P is in any way effective. Indeed none of the above results require that the underlying consistency property has any effective properties.

Finally Marek, Nerode, and Remmel [23] proved the following result about recursive FC-normal logic programs.

Theorem 7.15. *Let T be a recursive tree in $2^{<\omega}$ such that $[T] \neq \emptyset$. Then there is a FC-normal recursive logic program P such that there is an effective one-to-one degree preserving correspondence between $[T]$ and $\mathcal{S}(P)$.*

Reiter ([24]) proved that there is a recursive normal default theory with no recursive extension. Theorem 7.15, which was originally proved for nonmonotonic rule systems of which logic programs and default theories are special cases, contains Reiter's result as special case. In addition, it gives much finer information even for recursive normal default theories since the set of degrees of paths through highly recursive trees have been extensively studied. For example, our correspondence allows us to transfer results about the possible degrees of paths through highly recursive trees to results about the degrees of stable models of recursive FC-normal logic programs. A number of results of this kind has been proved in [23].

8 Conclusions

In this paper we established three classes $\mathcal{C}$ of logic programs such that $\mathcal{C}$ have compactness property. Two of these classes are based on enforcing tighter restriction on proof theoretic characterizations of stable models. The third one is based on the form of consistency property of Scott applied to logic programs. It is likely that there are other compact classes of programs and, in fact, one such class has been identified in [5]. The classes we found are quite rich, and it is likely that additional strong restrictions would have to be imposed on compact classes to make them a viable tool for applications. But what would be such applications? As is clear from the results stated in this paper, unlimited negation allows the programmer to write programs that have a unique stable model, but that model is so complex that it cannot be effectively queried. Even the stratified negation does not prevent stable models that are too complex for querying [2]. The logic programming community answered these challenges by limiting programs to DATALOG$^\neg$ programs (which essentially means eliminating untabled functions). In such setting the Herbrand base $B_{\mathcal{L}}$ is finite and so practical systms based on this mechanism can be built [25, 4]. Yet, the current implementations of PROLOG mostly work and allow for use of negation. Thus there is hope that the quest for some reasonable and *practical* semantics of logic programs with negation is not entirely futile. Our paper is a small step in this direction.

References

[1] K. Apt. Logic programming, In: J. van Leeuven, ed, Handbook of Theoretical Computer Science, pages 493–574, MIT Press, Cambridge, MA", 1990.

[2] K. R. Apt, H. A. Blair, Arithmetical Classification of Perfect Models of Stratified Programs, Fundamenta Informaticae 13 (1990) 1-17.

[3] K. Apt, H. Blair, and A. Walker. Towards a Theory of Declarative Knowledge. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 89–142, Los Altos, CA, 1987. Morgan Kaufmann.

[4] Y. Babovich and V. Lifschitz. *Cmodels*, 2002. `http://www.cs.utexas.edu/users/tag/cmodels.html`.

[5] P.A. Bonatti. Resolution for Skeptical Stable Model Semantics. *Journal of Automated Reasoning* 27:391–421, 2001.

[6] D. Cenzer and J. B. Remmel, Index Sets for Π^0_1-classes, Annals of Pure and Applied Logic 93 (1998), 3-61.

[7] D. Cenzer and J. B. Remmel, Π^0_1-classes in Mathematics, in: *Handbook of Recursive Mathematics: Volume 2*, eds. Yu. L. Ershov, S.S. Goncharov, A. Nerode, and J.B. Remmel, Studies in Logic and the Foundations of Mathematics, vol. 139, Elsevier, 1998, pp. 623-822.

[8] D. Cenzer, J. B. Remmel, and A. K. C. S. Vanderbilt, Locally Determined Logic Programs, in: *Proceedings of the 6th international Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR99)*, Springer-Verlag, 1999, pp. 34-49.

[9] P. Cholewiński, Stratified default theories, in *Proceedings of CSL'94*, Lecture Notes in Computer Science, vol. 933, Springer-Verlag, 1995.

[10] P.M. Dung and K. Kanchanasut. A Fixpoint Approach to Declarative Semantics of Logic Programs, In: E.L. Lusk and R.A. Overbeek eds, *Logic Programming, Proceedings of North American Conference*, pages 604–625, 1989

[11] A. Ferry, A topological characterization of the stable and minimal model classes of propositional logic programs, Ann. Math. Artificial Intelligence 15 (1995), 325-355.

[12] M. Gelfond and V. Lifschitz. The Stable Semantics for Logic Programs. In *Proceedings of the 5th International Symposium on Logic Programming*, pages 1070–1080, Cambridge, MA., 1988. MIT Press.

[13] K. Gödel. Über formal unentscheibare Sätze der Principia Mathemaica und verwandter Systeme I. *Monatshefte Math. Phys.* 38:173–198, 1931.

[14] C.G. Jockusch and R.I. Soare. π_1^0 Classes and Degrees of Theories. *Transactions of American Mathematical Society*, 173:33–56, 1972.

[15] V. Lifschitz and H. Turner, Splitting a logic program, in: *Proceedings of the Eleventh International Conference on Logic Programming*, ed. P. Van Hentenryck, 1994, pp. 23–37.

[16] J. Lloyd, *Foundations of Logic Programming*, Springer-Verlag, 1989.

[17] W. Marek, A. Nerode, and J.B. Remmel. Nonmonotonic Rule Systems I. *Annals of Mathematics and Artificial Intelligence*, 1:241–273, 1990.

[18] W. Marek, A. Nerode, and J.B. Remmel. Nonmonotonic Rule Systems II. *Annals of Mathematics and Artificial Intelligence*, 5:229-264, 1992.

[19] W. Marek, A. Nerode, and J.B. Remmel. A Context for Belief Revision: Normal Logic Programs (Extended Abstract) *Proceedings, Workshop on Defeasible Reasoning and Constraint Solving*, International Logic Programming Symposium, San Diego, CA., 1991.

[20] W. Marek, A. Nerode, and J.B. Remmel. How Complicated is the Set of Stable Models of a Logic Program? *Annals of Pure and Applied Logic*, 56:119-136, 1992.

[21] W. Marek, A. Nerode, and J. B. Remmel, The stable models of predicate logic programs. Journal of Logic Programming 21 (1994), 129-154.

[22] W. Marek, A. Nerode, and J. B. Remmel, Context for belief revision: Forward chaining-normal nonmonotonic rule systems, Annals of Pure and Applied Logic 67 (1994), 269-324.

[23] R. Reiter. A Logic for Default Reasoning. *Artificial Intelligence*, 13:81–132, 1980.

[24] P. Simons, I. Niemelä, and T. Soininen. Extending and implementing the stable model semantics. *Artificial Intelligence*, 138:181–234, 2002.

[25] A. Van Gelder, K.A. Ross, and J.S. Schlipf. Unfounded sets and well-founded semantics for general logic programs. *Journal of the ACM*, 38:587, 1991.