The Stable Models of a Predicate Logic Program

V. Wiktor Marek
Department of Computer Science
University of Kentucky
Lexington, KY 40506–0027, USA

Anil Nerode
Mathematical Sciences Institute
Cornell University
Ithaca, NY 14853

Jeffrey B. Remmel
Department of Mathematics
University of California at San Diego
La Jolla, CA 92903

1 Statement of problems and results

In this paper we investigate and solve the problem classifying the Turing complexity of stable models of finite and recursive predicate logic programs.

Gelfond-Lifschitz [7] introduced the concept of a stable model M of a Predicate Logic Program P. Here we show that, up to a recursive 1-1 coding, the set of all stable models of finite Predicate Logic Programs and the Π^0_1 classes (equivalently, the set of all infinite branches of recursive trees) coincide (Theorems 4.1 and 5.1). Typical consequences: 1) there are finite Predicate Logic Programs which have stable models, but which have no hyperarithmetic stable models; 2) for every recur-

sive ordinal α there is a finite Predicate Logic Program with a unique stable model of the same Turing degree as $\mathbf{0}^\alpha$ (Corollary 5.7). Another consequence of this result is that the problem of determining whether a finite Predicate Logic Program has a stable model is Σ^1_1-complete, i.e. the set of Gödel numbers of finite Predicate Logic Programs which have stable models is a Σ^1_1-complete set.

A *support* of a ground atom p is, roughly, a subset A of the Herbrand base such that whenever M is a stable model of the program P and $A \cap M = \emptyset$ then $p \in M$ (see below for a precise definition). Among supports of a ground atom p, there are always inclusion-minimal ones. Such minimal supports are finite. We call a program P **locally finite** if every atom has only finitely many minimal supports. Under our codings, locally finite Logic Programs correspond exactly to finitely splitting trees. Locally finite Logic Programs, for which there is an effective algorithm which applied to an atom p, produces a explicit list of all the minimal supports of p, correspond to recursively splitting recursive trees. We also show that local finiteness is a continuity property by associating with every Logic Program an operator on the Herbrand Base such that the program is locally finite iff the operator is continuous. It turns out that the classification of programs according to the number of supports of atoms provides additional information on the complexity of their stable models.

2 Motivation

Why are we interested in how hard it is to construct stable models M of Predicate Logic Programs P, and more generally in what the set of all stable models is like? Because stable models are good theoretical and computational candidates for knowledge representation of the set of beliefs, or point of view, of an agent holding to a theory in one of a variety of nonmonotone reasoning systems. These systems include:

- Reiter's extensions in Default Logic [23],
- Doyle's extensions in truth maintenance systems [6],
- Marek-Nerode-Remmel's theory of extensions in non-monotone rule systems [15, 16, 17, 18],
- Reinfrank et al. theory of nonmonotonic formal systems [22],

- Gelfond-Lifschitz stable models of logic programs [7].

McCarthy ([20]) suggested that non-monotonic reasoning could be formulated as a mathematical discipline. He introduced two notions of circumscription as a first try. The other systems above followed in his wake. The Marek-Nerode-Remmel formulation ([15]) was specifically designed to abstract all important common features in a logic-free formulation. It is a convenient half-way point for reformulation of non-monotonic theories as Logic Programs. Generally, the idea behind non-monotonic reasoning is that we should be allowed to deduce conclusions using a theory consisting of premises and rules of inference which can be a combination of

- knowledge, never later revised

and

-belief, held in the absence of contrary knowledge.

If we deduce using only knowledge, we are in the traditional domain of classical logic since Aristotle. If we deduce using beliefs as well, we can then deduce due to absence of knowledge as well as from its presence. We are then in the domain of non-monotonic reasoning. A warning to the untutored is that the notion of a unique least deductively closed set containing a theory, stemming from monotonic logics, is not appropriate for any of the non-monotone reasoning systems listed above. Rather, there are many minimal deductively closed sets for the theory, no one including another. If we pick one of these minimal deductively closed sets for the theory as our current "point of view" for decision making, and later new knowledge is obtained contradicting a belief of the theory, then we are impelled, for consistency's sake, to revise our theory by abandoning the offending belief. Also, we must abandon all conclusions inferred from beliefs contradicting facts, and we then must adopt a new theory and as a new "point of view", another minimal deductively closed set for that new theory. In contrast, in traditional monotone reasoning, once a premise is established, it and its consequences are never retracted or revised later. Characteristic of non-monotonic reasoning is retraction. Beliefs may be falsified by later facts and have to be abandoned, or at least replaced by new beliefs. What we are carrying out in other papers [18] is

- representing non-monotonic theories (premises and rules of inference) of current knowledge and belief as a Logic Program P,

- representing our current choice of a model, or deductively closed "point of view", as the choice of a stable model M of that Logic Program,

thus letting a pair (P, M) represent our current "state of mind". That is, when new facts contradict old beliefs, or newly preferred beliefs replace less preferred beliefs, we have a new Logic Program P' and need a new stable model M' of that theory to move to a new revised "state of mind" (P', M'). How to do this is our proposed calculus of belief revision using stable models ([17]).

To repeat, the Logic Programming machinery is a vehicle for natural representation of the syntax, deductive structure, and intended semantics of all the non-monotonic reasoning systems alluded to above. Implementations of Logic Programming can, in principle, be used as interpreters or compilers for these non-monotone reasoning systems. These implementations now vary widely, from those based on traditional Robinson's resolution to methods based on Jeroslow's "logic as mixed integer programming" paradigm ([3]). Equally important, informal and formal semantic reasoning about extensions of a default theory or a truth maintenance system can be carried out entirely using semantic reasoning about corresponding stable models of a corresponding Logic Program.

Where do the Logic Programs corresponding to nonmonotone theories come from? When stripped of logical and syntactic finery, many different non-monotonic logic systems have the same mathematical and computational structure, including their natural semantics. This is why algorithms for Logic Programs can also be used for default logic [23], truth maintenance systems [6, 5], circumscription [12, 13]. This can be dimly seen through the ad hoc translations of such systems into one another [8, 11, 19, 22]. But the diverse symbolisms are complicated and to a large extent irrelevant. The authors [15, 16] developed a common conceptual logic-free framework of non-monotone rule systems. The computational and mathematical equivalence of most of the subjects listed above is outlined there. Non-monotonic rule systems can be used as an easy intermediate stepping stone to reformulate theories and extensions of default logic and truth maintenance systems as Logic Programs and stable models.

To summarize, Logic Programming not only is an example of a non-monotonic reasoning system, but any interpreter or compiler for Logic Programs which computes stable models can also serve to compute extensions in the other non-monotonic reasoning systems listed above.

We remark that the non-monotonic rule system approach also revealed that finding stable models of logic programs and finding marriages for marriage problems and finding chain covers for partially ordered sets and many other combinatorial questions are essentially equivalent [15], allowing us to think about extensions using standard mathematics and algorithms for that standard mathematics.

2.1 Stable models

For an introductory treatment of Logic Programs, see [14]. Here is a brief self-contained account of their stable models [7]. Assume as given a fixed first order language 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 (atomic statements) of the language. A literal is an atomic formula or its negation, a ground literal is an atomic statement 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 atomic formula, 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. Horn clause programs are programs P consisting of Horn clauses. 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 ([14]).

Informally, the knowledge of a Logic Program is the set of clauses with no negated literals in the bodies, that is, the Horn clauses. The set of beliefs of a Logic Program is the set of clauses with negated literals occurring in the bodies. 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 [7]). 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 $N_{M,P}$. It is easy to see that a stable model for P is a minimal model of P ([7]). 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 [2] 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.

2.2 Proof schemes

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 non-monotonic consequence of P.

Of course, a proof scheme must use, 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 $p_l \leftarrow p_{m_1}, \ldots, p_{m_k}, \neg s_1, \ldots, \neg s_r$, $m_1 < l, \ldots, m_k < l$, and S_l is $S_{l-1} \cup \{s_1, \ldots, s_r\}$.

(We put $S_0 = \emptyset$).

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 [15].

Formally, 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 minimal proof schemes.

Here are some propositions from [15, 16].

Proposition 2.1 *Let P be a program and $M \subseteq B_{\mathcal{L}}$. Let p be an atom. Then p is in $N_{P,M}$ 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 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 2.1 *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.

Proposition 2.2 *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 $M \cup \{\neg q : q \in B_{\mathcal{L}} \setminus M\}$ is a solution of the system Eq_P.*

Here is a second characterization of stable models via proof schemes.

Proposition 2.3 *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.*

2.3 FSP Logic Programs

We now examine Logic Programs P such that every defining equation for every atom p is finite. This 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 2.2 *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.
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 $N_{P,M}$.

A *finitary support* program (*FSP* program) 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.

3 FSP and Continuity

In this section we 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 ([27]).

Associate an operator with each Logic Program as follows.

Definition 3.1 *Let P be a program. The operator $F_P : \mathcal{P}(B_\mathcal{L}) \rightarrow \mathcal{P}(B_\mathcal{L})$ 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 $N_{S,P}$.

Proposition 3.2 *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 3.3 *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 $\bigcup_{n\in\omega} F_P(S_n) = F_P(\bigcap_{n\in\omega} S_n)$.*

Proof: Suppose that $p \in B_{\mathcal{L}}$ is an atom. Then there exists a set $\mathcal{X}_p$ of finite subsets of $B_{\mathcal{L}}$ such that for every $S \subseteq B_{\mathcal{L}}$, $p \in F_P(S)$ if and only if there is a Y in $\mathcal{X}_p$ disjoint from S. Next, assume that $\langle S_n \rangle_{n\in\omega}$ is a descending sequence of subsets of the Herbrand base $B_{\mathcal{L}}$ and apply Proposition 3.2. We get the inclusion:

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

Conversely, assume that p is in $F_P(\bigcap_{n\in\omega} S_n)$. Then there exists a finite Y in $\mathcal{X}_p$,such that $\bigcap_{n\in\omega} S_n$ is disjoint from Y. This implies that there exists a finite n_0 such that $\bigcap_{n\leq n_0} S_n$ is disjoint from Y. Since the sequence $\langle S_n \rangle_{n\in\omega}$ is monotone decreasing, S_{n_0} is disjoint from Y. Thus p is in $F_P(S_{n_0})$, and therefore in $\bigcup_{n\in\omega} F_P(S_n)$. □

Proposition 3.4 *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

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

Proof: Assume (a), namely assume that P is FSP. We must prove that

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

The inclusion $\supseteq$ follows immediately from the anti–monotonicity of F_P (Proposition 3.2).

Now let us assume that p is in $\bigcap F(S_n)$. It follows that for every n in ω, there exists a Y in $\mathcal{X}_P$ such that S_n is disjoint from Y. Since $\mathcal{X}_p$ is

finite, there must be a Y in $\mathcal{X}_P$ such that for infinitely many n, S_n is disjoint from Y. Since the sequence $\langle S_n \rangle_{n\in\omega}$ is monotone increasing, *all* S_n must omit Y. Thus $(\bigcup_{n\in\omega} S_n) \cap Y = \emptyset$. So the implication (a)⇒(b) holds.

Now assume (b), namely that F_P is upper half-continuous. We must show that each $\mathcal{X}_p$ is finite. Assume otherwise, for a contradiction. So suppose that p is such that $\mathcal{X}_p$ is infinite. Since we can assume that $\mathcal{X}_p$ contains only inclusion-minimal supports, we face to the following situation: There is an infinite set of finite sets, $\mathcal{X}$ such that the elements of $\mathcal{X}$ are pairwise inclusion-incompatible and $\mathcal{X}$ is a family of sets such that for every S, $p \in F_P(S)$ if and only if for some $Y \in \mathcal{X}$, $Y \cap S = \emptyset$.

Order the countably infinite set $\mathcal{X}$ with order type ω as $\mathcal{X} = \langle Y_0, Y_1, \ldots \rangle$. We now construct two sequences, one $\langle a_n \rangle_{n\in\omega}$ of elements of $B_{\mathcal{L}}$, the other $\langle K_n \rangle_{n\in\omega}$ a sequence of subsets of ω.

Define $K_0 = \omega$, let a_0 be the first element of Y_0 (we assume that $B_{\mathcal{L}}$ is ordered of ordertype ω as well) such that $\{j : a \notin Y_j\}$ is infinite.
We claim that a_0 is well-defined. Otherwise, for every $a \in Y_0$ there is a natural number n_a such that for all $m > n_a$, $a \in Y_m$. Then, since Y_0 is finite, there is an $n \in \omega$ such that for all $m > n$, $Y_0 \subseteq Y_m$ (we just need take $n = max_{a\in Y_0} n_a$). We have a contradiction because distinct elements of $\mathcal{X}$ are incomparable with respect to inclusion. Hence a_0 exists and we may set $K_1 = \{n \in K_0 : a_0 \notin Y_n\}$.
Similarly, suppose that a_l and K_l are already defined, and K_l is infinite, and $K_l = \{n : Y_n \cap \{a_0, \ldots, a_l\} = \emptyset\}$. Let n_l be the least n in K_l. Then we may select a_{l+1} as the first element a of Y_{n_l} such that $\{j \in K_l : a \notin Y_j\}$ is infinite. As above we can prove there is such an a_{l+1}. We then set $K_{l+1} = \{j : \{a_0, \ldots, a_{l+1}\} \cap Y_j = \emptyset\}$. Notice that our construction ensures that K_{l+1} is infinite. Now define X_n as $\{a_0, \ldots a_n\}$, then, by construction, for some j (in fact for infinitely many j), X_n is disjoint from Y_j. Thus p is in $F_P(X_n)$. Hence

$$p \in \bigcap_{n\in\omega} F_P(X_n)$$

On the other hand, setting $Z = \bigcup_{n\in\omega} X_n$ we have, by construction, $Z \cap Y_j \neq \emptyset$, for all $j \in \omega$. Since $\mathcal{X} = \langle Y_0, Y_1, \ldots \rangle$ and none of the Y_j's is omitted by Z, $p \notin F_P(Z)$. Hence $F_P(\bigcup_{n\in\omega} X_n) \subset \bigcap_{n\in\omega} F_P(X_n)$. □

4 Coding Stable Models into Trees

In this section, we shall give the neccessary recursion theoretic background to make precise and to prove our claim 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.

4.1 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.

4.2 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 define a function $su^P: B_{\mathcal{L}} \to \omega$ 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.

4.3 Tools from Recursion Theory

Let $\omega = \{0, 1, 2, \ldots\}$ denote the set of natural numbers and let $< , >: \omega \times \omega \to \omega - \{0\}$ 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 $\{c(\alpha): \alpha \in S\}$ 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 $f: \omega \to \omega$ 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 $A = \{f: \omega \to \omega : \forall_n(R(< n, < f(0), \ldots, f(n-1) >>)\}$. A Π_1^0-class A is *recursively bounded* if there is a recursive function $g: \omega \to \omega$ 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 $\{\beta = (\alpha_1, \ldots, \alpha_n, k): \beta \in T\}$. If A is a recursively bounded Π_1^0-class, then $A = [T]$ for some highly recursive tree $T \subseteq \omega^{<\omega}$, see [10]. 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 $\{\beta =< \alpha_1, \ldots, \alpha_n, k) : \beta \in T\}$.

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 $f : \omega \to \omega$, then $gr(f) = \{< x, f(x) > : x \in \omega\}$. 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 $A \oplus B = \{2x : x \in A\} \bigcup \{2x + 1 : x \in B\}$.

4.4 Representing programs by trees

Theorem 4.1 *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.*

Proof: Enumerate effectively and without repetitions the Herbrand base $B_{\mathcal{L}}$ of the language $\mathcal{L}$ of the program P. Use this to identify $B_{\mathcal{L}}$ with ω, having in mind that when we talk about a subset $A \subseteq \omega$, then such a set A determines a unique subset of $B_{\mathcal{L}}$. Next, notice that since P is a recursive program, so is $ground(P)$. Also, given $p \in B_{\mathcal{L}}$ the set of minimal proof schemes with conclusion p is recursive. The set of supports of such schemes is *not* necessarily recursive, although it is recursively enumerable. We will have to be careful about this point to avoid errors in the proof.

We shall encode a stable model M of P by a path $\pi_M = (\pi_0, \pi_1, \ldots)$ through the complete ω-branching tree $\omega^{<\omega}$ as follows.

First, for all $i \geq 0$, $\pi_{2i} = \chi_M(i)$. That is, at the stage $2i$ we encode the information if the atom encoded by i belongs to the model M.

Next, if $\pi_{2i} = 0$ then $\pi_{2i+1} = 0$. But if $\pi_{2i} = 1$, that is $i \in M$, then we put π_{2i+1} equal to that $q_M(i)$ such that $q_M(i)$ is the least code for a minimal proof scheme for i for which the support of φ is disjoint from M. That is, we select a minimal proof scheme φ for i ,or to be precise for the atom encoded by i, such that φ has the smallest possible code of any proof scheme ϱ such that $supp(\varrho) \cap M = \emptyset$. If M is a stable model then (Proposition 2.3) for every $i \in M$, at least one such proof scheme exists.

Clearly $M \leq_T \pi_M$. For it is enough to look at the values of π_M at even places to read off M. Now given an M-oracle, it should be clear that for each $i \in M$, we can use an M-oracle to find $q_M(i)$ effectively. This means that $\pi_M \leq_T M$. Thus the correspondence $M \mapsto \pi_M$ is an effective degree-preserving correspondence. It is trivially $1:1$.

Now we have to construct a recursive tree $T \subseteq \omega^\omega$ such that $[T] = \{\pi_M : M \in stab(P)\}$.

Let N_k be the set of all codes of minimal proof schemes φ such that all the atoms appearing in all the rules used in φ are smaller than k. Obviously N_k is finite. We can also find the canonical index for N_k, uniformly in k.

We have to say which finite sequences belong to our tree T. To this end, given a sequence $\sigma = (\sigma(0), \ldots, \sigma(k)) \in \omega^{<\omega}$ set $I_\sigma = \{i : 2i \leq k \wedge \sigma(2i) = 1\}$ and $O_\sigma = \{i : 2i \leq k \wedge \sigma(2i) = 0\}$. Now we define T by putting σ into T if and only if the following five conditions are met:
(a) $\forall_i (2i+1 \leq k \wedge \sigma(2i) = 0 \Rightarrow \sigma(2i+1) = 0)$.
(b) $\forall_i (2i+1 \leq k \wedge \sigma(2i) = 1 \Rightarrow \sigma(2i+1) = q$
where q is a code for a minimal proof scheme φ such that $conc(\varphi) = i$

and $supp(\varphi) \cap I_\sigma = \emptyset$).
(c) $\forall_i(2i+1 \leq k \wedge \sigma(2i) = 1 \Rightarrow$ there is no code $c \in N_{\lfloor k/2 \rfloor}$ of a minimal proof scheme ψ such that $conc(\psi) = i$, $supp(\psi) \subseteq O_\sigma$ and $c < \sigma(2i+1))$.
(here $\lfloor \cdot \rfloor$ is the so-called number-theoretic "floor" function).
(d) $\forall_i(2i \leq k \wedge \sigma(2i) = 0 \Rightarrow$ there is no code $c \in N_{\lfloor k/2 \rfloor}$ of a minimal proof scheme θ such that $conc(\theta) = i$ and $supp(\psi) \subseteq O_\sigma)$
(e) If $k = 2i+1$, then there is no number $j < \sigma(k)$ such that:
(e_1) j is a code for a minimal proof scheme with conclusion i
(e_2) The proof scheme ψ coded by j has the same support as the proof scheme φ coded by $\sigma(k)$.

Condition (a) corresponds to the first condition we imposed on π_M. Conditions (b), (c), (d), take care of the second condition. It should be clear that the conditions (c) and (d) do not eliminate all "false" finite sequences. There may be sequences that are in T and are not initial sequences of a π_M. Those sequences will be cut off because their extensions will be eliminated. Condition (e) needs an additional explanation: there may be many minimal proof schemes with the conclusion i and same support. Condition (e) ensures that only one code, in fact the least one, of such a proof scheme is encoded among the successors of a given σ of even length $2i$. The net effect of this condition is that, if i has the property that there are only finitely many supports of minimal proof schemes with the conclusion i, then every node on level $2i$ will have only finitely many successors. Moreover it is clear that all the proof schemes used in our encoding of M by π_M are put into T.

It is immediate that if $\sigma \in T$ and $\tau \sqsubseteq \sigma$, then $\tau \in T$. Moreover it is clear from the definition that T is a recursive subset of $\omega^{<\omega}$. Thus T is a recursive tree.

Also, it is easy to see that our definitions ensure that, for a stable model M of P, the sequence π_M is a branch through T, that is, $\pi_M \in [T]$.

We shall show now that every infinite branch through T is of the form π_M for a suitably chosen stable model M. To this end assume that $\beta = (\beta(0), \beta(1), \ldots)$ is an infinite branch through T. There is only one candidate for M, namely $M_\beta = \{i : \beta(2i) = 1\}$.

Two items have to be checked:
(I) M_β is a stable model of P.
(II) $\pi_{(M_\beta)} = \beta$.

First we prove (I).
If M_β is not a stable model of P then, according to Proposition 2.3 one of two cases must hold:
(i) there is $i \in M_\beta \setminus N_{M_\beta,P}$ or
(ii) there is $j \in N_{M_\beta,P} \setminus M_\beta$.

If (i) holds, then consider such an i and the term $\beta(2i+1)$. For $(\beta(0), \ldots, \beta(2i+1)) = \beta^{(2i+1)}$ to be in T it must be the case that $\beta(2i+1)$ is a code of a minimal proof scheme φ such that $conc(\varphi) = i$ and $supp(\varphi) \cap I_{\beta^{(2i+1)}} = \emptyset$. But since $i \notin N_{M_\beta,P}$, there must be some n belonging to $M_\beta \cap supp(\varphi)$. Choose such an n. Then $\beta^{(2n)} \notin T$ because $supp(\varphi) \cap T_{\beta^{(2n)}} \neq \emptyset$ (Thus if there is a proof scheme that seems to work, but should not, then we close the branch a bit later).

If (ii) holds, then consider such a j. For some n there is a proof scheme ψ such $conc(\psi) = j$, $supp(\psi) \subseteq O_{\beta^{(n)}}$. But then $\beta^{(n)}$ does not satisfy the condition (d) of our definition of the tree.

Thus both (i) and (ii) have been excluded, and we have established that if $\beta \in [T]$, then M_β is a stable model of P. This is property (I) above.

Finally, we claim (II) that is: if $\beta \in [T]$ then $\beta = \pi_{(M_\beta)}$.

If $\beta \neq \pi_{(M_\beta)}$ then for some $i \in M_\beta$ there is a code c of a minimal proof scheme φ such that $conc(\varphi) = i$, $supp(\varphi) \subseteq \omega \setminus M_\beta$ and $c < \beta(2i+1)$. Then there is an n large enough so that $supp(\varphi) \subseteq O_{\beta^{(n)}}$ and hence $\beta^{(n)}$ does not satisfy the condition (c) for $\beta^{(n)}$ to be in T. Hence, if $\beta \neq \pi_{(M_\beta)}$, then $\beta^{(n)} \notin T$ for some n and so $\beta \notin [T]$. This completes the proof of (II) and of (1) as well.

Now we look at the properties (2) and (3) from our theorem.

Notice that the tree T constructed in the proof of (1) has the property that, whenever $\beta^{(n)} \in T$, then two things happen:
(†) For every i such that $2i \leq n$, $\beta(2i) \in \{0, 1\}$.
(‡) For every i such that $2i + 1 \leq n$, $\beta(2i+1)$ is either 0 or it is a code of a minimal proof scheme φ such that $conc(\varphi) = i$ and no $j < \beta(2i+1)$ has same conclusion and same support. Therefore if P has a finite number of supports of minimal proof schemes for each i, then T is finitely branching. This proves (2).

Finally, if P has the additional property that there is a recursive function whose value at i encode all the supports of minimal proof schemes for i, then the tree T is recursively bounded. Indeed, at all even levels the values of the bounding function can be read off from the previous

levels. At the odd levels we proceed as follows: Once we know the code for all codes of supports of schemes for i, we search the proof schemes until we find a proof scheme for i with the given support. Then we make sure that there is no proof scheme with smaller code and same conclusion and support. We do this for all supports for i. So if we have a recursive function encoding a standard code of the set of all codes of supports of minimal proof schemes, then from the value of this function we can decode all the values that we put at odd levels. As the procedure is effective in the function encoding the set of supports, there is a recursive bound on the values of the successors. This proves (3). □

5 Coding Trees into Stable Models of Finite Logic Programs

In this section, we shall give the converse of the Theorem 4.1, namely that given any recursive tree T, there exists a finite Predicate Logic Program P such that there is an effective 1:1 degree preserving correspondence between $[T]$ and $Stab(P)$. We also give two refinements.

5.1 Representation of trees by programs

Theorem 5.1 *Let C be any Π^0_1-class. Then*

1. *There is a finite program, P, and an effective one-to-one degree preserving correspondence between the elements of C and the set of all stable models of P, $Stab(P)$.*
2. *If in addition C is of the form $[T]$ for a finitely splitting T, then P can be chosen FSP.*
3. *If in addition T is a highly recursive tree, then P can be chosen recursively FSP.*

Proof: Let C be a non-empty Π^0_1 class. Then there is a recursive tree $T \subseteq \omega^{<\omega}$ such that $C = [T]$.

Consequently, we need to prove that, given a recursive tree T, there is a finite program P such that there is a 1-1, degree-preserving correspondence between $[T]$ and $Stab(P)$.

Our assumption is that T is a recursive tree. Recall that the code of the empty sequence $\emptyset$ is 0 and the code $c(\alpha)$ of a finite sequence $\alpha = (a_1, \ldots, a_n)$ is $c(\alpha) =< n, < a_1, \ldots, a_n >>$ where $<,>$ is the recursive pairing function defined in section 4.3. The usual way of representing recursive relations by Horn Clause programs gives the following results.

I. There exists a finite Horn program P_0 such that for a predicate $tree(\cdot)$ of the language of P_0, the atom $tree(n)$ belongs to the least Herbrand model of P_0 if and only if n is a code for a finite sequence σ and $\sigma \in T$ (n is an abbreviation of term $s^n(0)$),

II. There is a finite Horn program P_1 such that for a predicate $seq(\cdot)$ of the language of P_1, the atom $seq(n)$ belongs to the least Herbrand model of P_1 if and only if n is the code of a finite seqence α.

III. There is a Horn program P_2 which correctly computes several notions for manipulating predicates and functions on sequences.

Here is a short list of predicates that are properly defined within P_2 and which compute several recursive relations:
(a) *samelength*$(\cdot,\cdot)$. This succeeds if and only if both arguments are the codes of sequences of the same length.
(b) *diff*$(\cdot,\cdot)$. This succeeds if and only if the arguments are codes of sequences which are different.
(c) *shorter*$(\cdot,\cdot)$. This succeeds if and only both arguments are codes of sequences and the first sequence is shorter than the second sequence.
(d) *length*$(\cdot,\cdot)$. This succeeds when the first argument is a code of a sequence and the second argument is the length of that sequence.
(e) *notincluded*$(\cdot,\cdot)$. This succeeds if and only if both arguments are codes of sequences and the first sequence is not the initial segment of the second sequence.

Now, the program P^- is the union of programs $P_0 \cup P_1 \cup P_2$. This program P^- is a Horn program. The predicate $tree(\cdot)$ computes in the least model of P^- precisely all the codes of sequences that are in T. Clearly P^- is a finite program. We denote its language by $\mathcal{L}^-$. M^- is the least Herbrand model of P^-.

After we add the clauses (1)-(7) below, the resulting program will be also a finite program. Those additional clauses will not contain any of predicates of the language $\mathcal{L}^-$ *in the head*, but they will appear in the body of clauses (1) to (7). Therefore, whatever stable model of the extended program we consider, its trace on the set of ground atoms of $\mathcal{L}^-$ will be M^-. In particular, the meaning of the predicates listed above will always be the same.

We are ready now to write the additional clauses which, together with the program P^-, will form the desired program P.

First of all, we select three new unary predicates:
(i) *inbranch*$(\cdot)$, having as intended interpretation the set of codes of sequences forming a branch through T. This branch corresponds to the stable model of P.
(ii) *notinbranch*$(\cdot)$, having as intended interpretation the set of all codes of sequences which are in T but are not on the branch that the model describes.
(iii) *control*$(\cdot)$, which is used to make sure that the branch is infinite.

Here are the final 7 clauses of our program.

(1) $inbranch(X) \longleftarrow tree(X),\ \neg notinbranch(X)$
(2) $notinbranch(X) \longleftarrow tree(X),\ \neg inbranch(X)$
(3) $inbranch(0)$ /* Recall 0 is the code of the empty sequence */
(4) $notinbranch(X) \longleftarrow tree(X),\ inbranch(Y),$
$tree(Y), samelength(X,Y), diff(X,Y)$
(5) $notinbranch(X) \longleftarrow tree(X),\ tree(Y),\ inbranch(Y),\ shorter(Y,X),$
$notincluded(Y,X)$
(6) $control(X) \longleftarrow inbranch(Y),\ length(Y,X)$
(7) $control(X) \longleftarrow \neg control(X)$.

Clearly, $P = P^- \cup \{(1), \ldots, (7)\}$ is a finite program. Its language is denoted by $\mathcal{L}$.

We shall prove that:
(I) If T is a finitely splitting recursive tree, then every element of $B_{\mathcal{L}}$ has only finitely many supports of minimal proof schemes. Thus, if T is finitely splitting, then P is FSP.
(II) There is a one-to-one degree preserving correspondence between the stable models of P and the infinite branches through P.
(III) If T is recursively bounded, then P is recursively FSP.

First we prove (I). Let us look at the structure of our program. As we wrote it, there are two connected but separate parts. First, we have the part P^-. It is a Horn program and therefore the support of the minimal proof schemes for each atom of its language is empty. This is because there is no negation in the body of clauses of P^-. Hence no negative information is collected into its support. Thus, all atoms in the language of $\mathcal{L}^-$ either have no derivation at all, or have only derivations with empty support. When we add clauses (1)-(7), we note that no atom of $\mathcal{L}^-$ is in the head of any of these new clauses. This

means that no grounded instance of such a clause can be present in a minimal proof scheme with conclusion any atom of $\mathcal{L}^-$. This means that minimal proof schemes with conclusion an atom p of $\mathcal{L}^-$ (*with respect to* P) can involve only clauses from P^-, and so must have empty support. Thus, ground atoms of $\mathcal{L}^-$ have only finitely many supports of minimal proof schemes: zero, if they are not in the least model of P^-, or one, if they are in the least model of P^-. Morover we can ensure that the least model of P^- is recursive so that given any ground atom of $\mathcal{L}^-$, we can effectively decide if it has a proof scheme.

Now we shall look at the atoms appearing in the heads of clauses (1)-(7),

These are atoms of the following three forms:
(a) *inbranch*(t)
(b) *notinbranch*(t)
(c) *control*(t)

The ground terms of our language are of form n, where $n \in \omega$, that is, of the form $s^n(0)$. In the case of atoms of the form *inbranch*(t) and *notinbranch*(t) the only ground terms which possess a proof scheme must be those for which t is a code of a sequence of natural numbers belonging to T. The reason for this is that the clauses having those predicates in the head have, in the body, predicates from $\mathcal{L}^-$ which fail if t is not a sequence. The only exception is clause (3) which is, itself, a fact of the form *inbranch*(0) and 0 is the code of the empty seqence which is in every tree T by definition. This eliminates from our consideration ground atoms of the form *inbranch*(t) or *notinbranch*(t) with $t \notin T$. Similarly, the only ground atoms of the form *control*(t) which possess a proof scheme are atoms of the form *control*(n), where n is a natural number.

Thus we are left with this cases:
(A) *inbranch*$(c(\sigma))$, where $\sigma \in T$.
(B) *control*(m), where m is a natural number.
(C) *notinbranch*$(c(\sigma))$, where $\sigma \in T$.

Case (A). Atoms of the form *inbranch*$(c(\sigma))$, $\sigma \in T$.
There are only two clauses C with *inbranch* in the head. Those are (1) and (3). Clause (3) is an atom. This implies that a minimal proof scheme which derives *inbranch*(0) and uses (3) must be of the form $< \langle \textit{inbranch}(0), (3), \emptyset \rangle >$. The remaining clause (1) has this feature: its body contains positive atoms from $\mathcal{L}^-$. Atoms appearing there negatively are *notinbranch*. In case the last clause employed in the

minimal proof scheme for $inbranch(c(\sigma))$ is (1), then this proof scheme φ must be a concatenation of a proof scheme for $tree(c(\sigma))$ and the sequence $< \langle inbranch(c(\sigma)), (1), \{notinbranch(c(\sigma))\}\rangle >$. The reason why we have only a one element support is that the support of the proof scheme of $tree(\sigma)$ is empty.

Thus, in case (A) there were either one or two supports of minimal proof schemes.

Case (B). Atoms of the form $control(k)$. There are only two clauses with the atom *control* in the head. These are clauses (6) and (7). Clause(6) has in the body two atoms: *length* - which has always at most one support of a minimal proof scheme (namely the empty set), and $inbranch(\cdot)$. Since we are proving (I) and so our assumption is that the tree T is finitely splitting, there are only finitely many sequences σ in T of length k. Now, let us look at a proof scheme for $control(k)$ whose last rule is of type (6). Such a scheme must be composed of the scheme for $inbranch(c(\sigma))$ and a scheme for $length(c(\sigma), k)$. The existence of this second scheme implies that the length of σ is k. There are only finitely many supports of minimal proof schemes for each $inbranch(c(\sigma))$ (here we use (A)). All such supports are supports for $control(k)$. Hence $control(k)$ has only finitely many supports of its minimal proof schemes which end in clause (6). If a minimal proof scheme φ ends in clause (7), then φ is $< \langle control(k), (7), \{control(k)\}\rangle >$. Thus there are only finitely many supports for minimal proofs schemes with conclusion equal to $control(k)$ for any k.

Case (C) Atoms of the form $notinbranch(c(\sigma))$. Here we have to take into account clauses (2), (4), and (5). When the last clause used in the minimal proof scheme is (2), we reason as in case (A), the subcase of clause (1). When that last clause is (4) the support is inherited from an atom of the form $inbranch(c(\tau))$ where τ is also in T, $\tau \neq \sigma$, and the length of τ is the same as that of σ. There are only finitely many such τ's, and each of them has only finitely many supports for $inbranch(c(\tau))$. Therefore here also we have finitely many supports. The case of clause (5) is similar, there are only finitely many sequences τ in T of shorter length, and the supports of proof schemes for $inbranch(c(\tau))$ are supports for $notinbranch(c(\sigma))$. Thus this case also creates only finitely many supports. Thus there are only finitely many supports of minimal proofs schemes with conclusion $notinbranch(c(\sigma))$.

Thus, we have proved that if T is finitely branching then every ground atom possesses only finitely many supports of minimal proof schemes.

Now we prove (II).
We establish a "normal form" for the stable models of P. Each such model must contain M^-, the least model of P^-. In fact, the restriction of a stable model of P to $B_{\mathcal{L}^-}$ is M^-.

Given any $\beta \in \omega^\omega$, that is, $\beta = (\beta^{(1)}, \beta^{(2)}, \ldots)$ is any infinite sequence of natural numbers, we assign to M the following set of ground atoms: $M_\beta = M^- \cup \{control(n) : n \in \omega\} \cup \{inbranch(c(\beta^{(n)}) : n \in \omega\} \cup \{notinbranch(c(\sigma)) : \sigma \in T \setminus \{\beta^{(n)} : n \in \omega\}$
We prove that the stable models of P are exactly all M_β for $\beta \in [T]$. This will use Proposition 2.3.

First, assume that M is a stable model of P. We know that the atoms of $\mathcal{L}^-$ in M constitute M^-. What is the disposition of the remaining atoms of $\mathcal{L}$? We claim that all the atoms of the form $control(n)$ are in M. Suppose to the contrary that, for some k, atom $control(k)$ is not in M, then clause (7) would ensure that $control(k)$ is in M which is a contradiction.

Because for every k, $control(k) \in M$, it follows that for every $k \geq 0$, M contains an atom of the form $inbranch(c(\sigma))$ with $length(\sigma) = k$. That is, we can not use clause (7) to derive $control(k)$. Hence our analysis of minimal proof schemes φ such that $conc(\varphi) = control(k)$ shows that φ must include a minimal proof scheme for some atom of the form $inbranch(c(\sigma))$ with $\sigma \in T$ since otherwise, $control(k)$ has no derivation, contradicting the stability of M (see Proposition 2.3).

Next, we claim that for each $\sigma \in T$, precisely one of $inbranch(c(\sigma))$, $notinbranch(c(\sigma))$ is in M. Indeed clauses (1) and (2) ensure that at least one of these two atoms is in M. But, if both $inbranch(c(\sigma))$, $notinbranch(c(\sigma))$ are in M, since only clauses (1) and (3) have $inbranch$ in the head, we see that for $\sigma \neq \emptyset$, only (1) can be used. But the use of (1) is blocked by presence of $notinbranch(c(\sigma))$ in M. The case $\sigma = \emptyset$ is similar. Thus we have established that exactly one of $inbranch(c(\sigma))$, $notinbranch(\sigma)$ is in M.

Next, we will use the fact that M satisfies the clause (4) to show that for every k, there is at most one atom in M of the form $inbranch(c(\sigma))$ having $length(\sigma) = k$. But we know that for some σ of length k, there is an atom of the form $inbranch(c(\sigma))$ in M. So we can conclude that there is exactly one such σ.

Using clause (5) we establish the compatibility of every pair σ, σ' such that $inbranch(c(\sigma))$, $inbranch(c(\sigma'))$ belong to M. Taking into account the fact that M contains atoms of the form $inbranch(c(\sigma))$ with σ of

arbitrary length, we conclude that there is exactly one infinite sequence β such that $inbranch(c(\beta^{(n)}))$ belongs to M for all $n \in \omega$. By (4) all the atoms of the form $notinbranch(c(\tau))$ for $\tau \in T \setminus \{\beta^{(n)} : n \in \omega\}$ are in M. Since no other atom is in M, we conclude that $M = M_\beta$. Note that we have proved that if $[T]$ is empty, then P has no stable model.

To complete the argument for (II), we have to prove that $\beta \in [T]$ implies that M_β is a stable model of P. The presence of clauses (1) and (2) in P implies that $\{inbranch(c(\beta^{(n)})) : n \in \omega\} \cup \{notinbranch(c(\sigma)) : \sigma \in T \setminus \{\beta^{(n)} : n \in \omega\}\} \subseteq N_{M_\beta,P}$. Then clause (6) can be used to show that for all n, $control(n)$ also belongs to $N_{M_\beta,P}$. But $M^- \subseteq N_{M_\beta,P}$. We finally conclude that $M_\beta \subseteq N_{M_\beta,P}$.

We can prove that $N_{M_\beta,P} \subseteq M_\beta$ by a straightforward induction on the length of proof scheme. The case when a proof scheme has as conclusion atom $p \in \mathcal{L}^-$ is easy. It reduces to induction on the number of iterations of the operator T_{P^-}). Also, induction on the number of applications of clauses (1)-(7) is straightforward.

Given $\beta \in [T]$, the construction of M_β is easy. First, construct M^-; this does not depend on β and moreover M^- is a recursive set. Then add all the atoms $inbranch(c(\beta^{(n)}))$, and then all the atoms $notinbranch(c(\sigma))$ for all $\sigma \in T \setminus \{\beta^{(n)} : n \in \omega\}$. T is recursive, so the resulting set of atoms is certainly recursive in β. Thus M^β is recursive in β. On the other hand β can be effectively recovered from M_β easily, because our procedures were uniform, once T was fixed. We conclude that there is an effective, one-to-one, degree-preserving correspondence between $[T]$ and $Stab(P)$. This establishes (II).

To verify (3) we have to trace back our construction of supports of proof schemes for every atom. Whether or not T is recursively bounded, there is just one support for the atoms in M^-, the empty set. We can encode it. The atoms of $\mathcal{L}^-$ which are not in M^- have no supports. Since M^- is recursive, we can effectively find the set of supprots of minimal proof schemes for any atom in $\mathcal{L}^-$.

We only need to take care of atoms of the form $inbranch(c(\sigma))$, $control(n)$, and $notinbranch(c(\sigma))$. The first of these atoms, according to our analysis has either one or two supports. Whether or not T is finitely branching, we can effectively find these supports. The supports of $control(k)$ are either $\{control(k)\}$ or are inherited from supports of $inbranch(c(\sigma))$. If the tree T is recursively bounded then, uniformly in k, we can effectively enumerate all sequences of length k that are in T. From that enumeration, we can uniformly find a code for all the supports for all

minimal proof schemes for *control*(k). Finally, using a similar argument we can encode uniformly all supports of minimal proof schemes for atoms of the form *notinbranch*$(c(\sigma))$. Thus, the program P is recursively FSP. This completes the proof of (III), and of the theorem as well. □

A classical result, first explicit in [26] and [1] but known a long time earlier in equational form, is that every r.e. relation can be computed by a suitably chosen predicate over the least model of a finite Horn program. An elegant method of proof due to Shepherdson (see [25] for references) uses the representation of recursive functions by means of finite register machines. When such machines are represented by Horn programs in the natural way, we get programs in which every atom can be proved in only finitely many ways (See also [21]). Thus we can conclude:

Proposition 5.2 *Let $r(\cdot,\cdot)$ be a recursive relation. Then there is a finite program P_r computing $r(\cdot,\cdot)$ such that every atom in the least model M_r of P_r has only finitely many minimal proof schemes.*

We can combine Proposition 5.2 with the proof of Theorem 5.1, to strengthen parts (2) and (3) of that theorem. In part (2), we can require that P has only finitely many proof schemes for every atom. In part (3), we can require that there is a recursive bound on such proof schemes.

Let us think of the expressive power of a Logic Program as being crudely characterized by the kind of associated Π^0_1-class or the kind of tree which corresponds to its set of stable models in our constructions. Then, comparing the two versions of Theorem 5.1, we see that the more stringent requirement (beyond having only finitely many supports) of having only finitely many proof schemes, does not change the expressive power of Logic Programs. That is, if we can write a Logic Program P such that its stable models are "nicely" represented by the paths through a finitely splitting T, then we can write another Logic Program P' with that property plus the additional property that P' has only finitely many proof schemes for every atom.

5.2 Positive results for programs

When we compare Theorems 4.1 and 5.1, we see for Theorem 4.1 that not only for finite, but also for recursive Logic Programs P, the class

of all stable models $Stab(P)$ can be encoded by a Π^0_1 class. In turn, in Theorem 5.1 we encode Π^0_1 classes as the set of all stable models of a *finite* Logic Program. This implies that from the point of view of Turing reducibility it makes no difference if we write finite or infinite (but recursive) Logic Programs. Thus we have:

Corollary 5.3 *The expressive power of the stable semantics for finite Logic Programs and for recursive Logic Programs is the same, in the sense of 1-1 Turing degree preserving transformations. That is, for every recursive program predicate P there exists a finite predicate program P' such that there is an effective 1-1 Turing degree-preserving transformation from $Stab(P')$ onto $Stab(P)$.*

The degrees of elements of Π^1_0-classes have been extensively studied in recursion theory. The combined results of the theorems 4.1 and 5.1 is that we can immediately transfer results about degrees of elements of Π^1_0-classes to results about the degrees of stable models of finite Predicate Logic Programs. Below we shall state a sample of such results.

Corollary 5.4 (Positive results for recursive Logic Programs) *Suppose P is a recursive Logic Program with a stable model. Then*

1. *P has a stable model which is recursive in a complete Σ^1_1 set.*
2. *If P has denumerably many stable models, then each stable model of P is hyperarithmetic. Otherwise P has $2^{\aleph_0}$ stable models.*

If a program P is recursively FPS, then the tree T constructed in the proof of Theorem 4.1 is recursively bounded and so the class $[T]$ is highly recursive. Recursion theory again provides us with information on the Turing degrees of elements of such classes.

Corollary 5.5 (Positive results for recursively FPS Programs) *Suppose that P is a recursively FPS Logic Program with a stable model. Then*

1. *P has a stable model whose Turing jump is recursive in $\mathbf{0}'$.*
2. *If P has only finitely many stable models, then each of these stable models is recursive.*
3. *There is a stable model M of P in an r.e. degree.*

4. *There exist stable models M_1 and M_2 of P such that any function, recursive in both M_1 and M_2, is recursive.*

5. *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 $\mathbf{a}$.*

The next set of corollaries follow because a recursive finitely branching tree is automatically highly recursive in $\mathbf{0}'$.

Corollary 5.6 (Positive results for FPS Programs) *For any recursively FPS Logic Program P that possesses a stable model:*

1. *There is a stable model M of P whose Turing jump is recursive in $\mathbf{0}''$, the Turing jump of $\mathbf{0}'$.*

2. *If P has only finitely many stable models, then each of these stable models is recursive in $\mathbf{0}'$.*

3. *There is a stable model M which is in some r.e. degree in $\mathbf{0}'$.*

4. *There are stable models M_1 and M_2 such that any function, recursive in both M_1 and M_2, is recursive in $\mathbf{0}'$.*

5. *If P has no stable model which is recursive in $\mathbf{0}'$, then there is a nonzero degree $a >_T \mathbf{0}'$ such that a is r.e. $\mathbf{0}'$ and such that P has no stable model recursive in a.*

5.3 Negative results for programs

Every finite Logic Program is certainly recursive, so positive results such as those stated above in Corollaries 5.4, 5.5 and 5.6 for recursive Logic Programs certainly also hold for finite Logic Programs. In contrast, we get stronger negative results by constructing *finite* Logic Programs which do the same tasks we previously proved could be done with *recursive* Logic Programs, see [18]. Moreover our reduction of the Π^0_1 classes to classes $Stab(P)$ for a suitably constructed finite program P not only allows us to estimate the Turing complexity of stable models, but also provides us with finite programs with "pathological" behavior. This is interesting because it means that trying to prove that all finite programs have better behavior than this is fruitless, and to get better behavior we have to look for additional hypotheses.

Corollary 5.7 (Negative results for finite Logic Programs)

1. *There exists a finite Logic Program P such that P has a stable model but P has no stable model which is hyperarithmetic.*

2. *For any recursive ordinal α, there is a finite Logic Program P such that P has a unique stable model M and $M \equiv_T 0^{(\alpha)}$.*

Using well-known recursion-theoretic facts about recursively bounded Π^0_1 classes we get:

Corollary 5.8 (Negative results for recursively FSP Programs)

1. *There exists a finite Logic Program P_1 which is recursively FSP such that P_1 has no recursive stable model (although P_1 possesses $2^{\aleph_0}$ stable models).*

2. *There exists a finite recursively FSP Logic Program P_2 such that P_2 possesses $2^{\aleph_0}$ stable models and any two stable models $M_1 \neq M_2$ of P_2 are Turing incomparable.*

3. *If $\mathbf{a}$ is a Turing degree and $\mathbf{0} <_T \mathbf{a} <_T \mathbf{0}'$, then there exists a finite recursively FSP Logic Program P_3 such that P_3 has $2^{\aleph_0}$ stable models, a stable model of degree $\mathbf{a}$ but P_3 has no recursive stable model.*

4. *There exists a finite recursively FSP Logic Program P_4 such that if $\mathbf{a}$ is the degree of any stable model of P_4 and $\mathbf{b}$ is a r.e. degree with $\mathbf{a} <_T \mathbf{b}$, then $\mathbf{b} \equiv_T \mathbf{0}'$.*

5. *If $\mathbf{c}$ is any r.e. degree, then there exists a finite recursively FSP Logic Program P_5 such that the set of r.e. degrees which contain stable models of P_5 equals the sets of r.e. degrees $\geq_T \mathbf{c}$.*

6. *There exists a finite recursively FSP Logic Program P_6 such that if W is stable model for P_6 where $W <_T \mathbf{0}'$, then there exists a nonrecursive r.e. set A such $A <_T W$.*

We can relativize all the results in Corollary 5.5 to an $\mathbf{0}'$ oracle for FSP finite Logic Programs. This is due to the following result of Jockusch, Lewis, and Remmel.

Theorem 5.9 ([9]) *For any tree T which is highly recursive in $\mathbf{0}'$, there is a recursive finitely branching tree $S \subseteq \omega^{<\omega}$ with an effective one-to-one degree preserving correspondence between $[T]$ and $[S]$.*

Encoding highly recursive in $\mathbf{0}'$ trees by binary trees gives us now results on FSP finite Logic Programs.

Corollary 5.10 (Negative results for finite FSP Logic Programs)

1. *There exists a finite FSP Logic Program P_1 such that P_1 has no stable model which is recursive in $\mathbf{0}'$, although P possesses $2^{\aleph_0}$ stable models.*

2. *There exists a finite FSP Logic Program P_2 such that P_2 possesses $2^{\aleph_0}$ stable models and any two stable models $M_1 \neq M_2$ of P_2 have the property that $M_1 \bigoplus \mathbf{0}' \not\equiv_T M_2 \bigoplus \mathbf{0}'$.*

3. *If $\mathbf{a}$ is a Turing degree and $\mathbf{0}' <_T \mathbf{a} <_T \mathbf{0}''$, then there exists a finite FPS Logic Program P_3 such that P_3 has $2^{\aleph_0}$ stable models, a stable model of degree $\mathbf{a}$ but P_3 has no stable model which is recursive in $\mathbf{0}'$.*

4. *There exists a finite FPS Logic Program P_4 such that P has $2^{\aleph_0}$ stable models, and if $\mathbf{a}$ is the degree of any stable model of P_4 and $\mathbf{b}$ is a degree which is r.e. in $\mathbf{0}'$ with $\mathbf{a} <_T \mathbf{b}$, then $\mathbf{b} \equiv_T \mathbf{0}''$.*

5. *If $\mathbf{c} \geq_T \mathbf{0}'$ is any degree which is r.e. in $\mathbf{0}'$, then there exists a finite FSP Logic Program P_5 such that the set of degrees which are r.e. in $\mathbf{0}'$ and which contain stable models of P_5 equals the sets of degrees $\geq_T \mathbf{c}$ which are r.e. in $\mathbf{0}'$.*

6. *There exists a finite FSP Logic Program P_6 such that if W is stable model for P_6 where $\mathbf{0}' \leq_T W <_T \mathbf{0}''$, then there exists a set A such $\mathbf{0}' <_T A <_T W$ and A is r.e. in $\mathbf{0}'$.*

Corollary 5.6, combined with the fact that the perfect model, if it exists, is the unique stable model of the Logic Program ([7]), gives the following:

Corollary 5.11 *If P is a recursively FSP Logic Program, and P has a unique stable model then that unique stable model of P is recursive. Consequently, if P is a locally stratified, recursively FSP Logic Program, then its perfect model is recursive.*

This result is in contrast to the recent result of [4]. They show that arbitrarily complex hyperarithmetic sets can be encoded by perfect models

of a locally stratified finite program, so that every hyperarithmetic set is the projection of perfect model of such a program. Here, in contrast, the additional assumption of being recursively FSP reduces the complexity of such program to a recursive set!.

We end this paper with two more results which are a consequence of the proofs of Theorems 4.1 and 5.1

Corollary 5.12 *The problem of testing if a finite Predicate Logic Program possesses a stable model is Σ^1_1-complete, i.e., the set of Gödel numbers of finite Logic Programs which have a stable model is a Σ^1_1-complete set.*

Proof: Under a suitable Gödel numbering, we can view the set of ground atoms of any finite Logic Program P as a recursive subset of the natural numbers. It is then easy to see by directly writing out the definition in logical form that given a function $f : \omega \to \omega$, the predicate "the range of f is a stable model of P" is an arithmetic predicate. Moreover, given the index of an r.e. set e, i.e. let $W_e = \{x : \varphi_e(x) \text{ converges}\}$ where φ_e is the partial recursive function computed by the e-th Turing machine, it is also easy to check that the predicate "W_e is the set of codes of a finite predicate Logic Program" is also arithmetic predicate. It follows that $Stab = \{e : W_e$ is the set of codes of clauses of a finite Logic Program which has a stable model$\}$ is equal to the set of e such that $\exists f(R(e, f))$ where R is an arithmetic predicate. Hence the set $Stab$ is a Σ^1_1 set. Now it is well known that $Infpath = \{e : W_e$ is a recursive tree $\subseteq \omega^{<\omega}$ and $[W_e] \neq \emptyset\}$ is a Σ^1_1-complete set, see [24]. Thus to show that $Stab$ is a Σ^1_1-complete set, it is enough to show that $Infpath$ is 1:1 reducible to $Stab$, i.e. there exists a 1:1 recursive function g such that $e \in Infpath \leftrightarrow g(e) \in Stab$. The existence of the function g follows from a slight modification of the proof of Theorem 5.1. The idea is the following. Given an r.e. set W_e, we construct a new r.e. set W^*_e from W_e as follows. We enumerate W_e as $a_0, a_1, \ldots$. First we enumerate 0 which is the code of the empty sequence into W^*_e. Then we enumerate a_n into W^*_e if and only if

1. a_n is the code of some sequence $(\alpha_1, \ldots, \alpha_k)$,
2. the codes of all sequences $(\alpha_1, \ldots, \alpha_j)$ where $j \leq k$ appear in the enumeration of W_e, say $c((\alpha_1, \ldots, \alpha_j)) = a_{n_j}$ with n_j as small as posible,
3. for all $m \leq \max(\{n\} \cup \{n_j : j \leq k\})$, a_m is the code of some sequence $(\beta_1, \ldots, \beta_l)$ and for all $j \leq l$, the code of $(\beta_1, \ldots, \beta_j)$

appears in the enumeration of W_e.

It is easy to see that if W_e is tree, then $W_e = W_e^*$. If W_e is not a tree, then we claim that W_e^* is a finite tree. That is, either for some t, a_t is not the code of a sequence or a_t is the code of some sequence $(\beta_1, \ldots, \beta_m)$ but for some $j < m$, the code of $(\beta_1, \ldots, \beta_j)$ does not occur in W_e. In either case, (3) above ensures that no a_n with $n > t$ can be enumerated into W_e^* so that W_e^* will be finite. Finally, it is easy to check that our conditions ensure that if the code of $(\alpha_1, \ldots, \alpha_k)$ can be enumerated in W_e^*, then the codes of all sequences $(\alpha_1, \ldots, \alpha_j)$ where $j \leq n$ can also be enumerated in W_e^* so that W_e^* is a tree. Now there is a recursive function h such that $W_e^* = W_{h(e)}$. Then in the proof of Theorem 5.1, we can let $W_{h(e)}$ play the roll of T. Since the construction of the Program P_1 from $W_{h(e)}$ is uniform and works for r.e. as well as recursive sets, we can uniformly construct the r.e. index of a Program $P^{(e)}$ as in Theorem 5.1 such that there is an effecitve 1:1 correspondence between the stable models of $P^{(e)}$ and $[W_{h(e)}]$. Moreover, there is a recursive function g such that $W_g(e)$ is the set of codes of the clauses of $P^{(e)}$. It now follows that if $e \notin \mathit{Infpath}$, then either W_e is not a tree or W_e is a tree such that $[W_e] = \mathit{emptyset}$. In that case, $[W_e^*]$ will be empty and hence $P^{(e)}$ has no stable models. If $e \in \mathit{Infpath}$, then $P^{(e)}$ will have at least one stable model. Thus $e \in \mathit{Infpath} \leftrightarrow g(e) \in \mathit{Stab}$ as claimed so that Stab is a Σ_1^1-complete set.

We remark that the above proof works just as well if we use the set of canonical indices of finite Logic Programs rather than the set r.e. indices of finite Logic Programs. □

Corollary 5.13 *The problem of testing if a finite Predicate Logic Program possesses a a unique stable model is Σ_1^1-complete, i.e., the set of Gödel numbers of finite Logic Programs which have a unique stable model is Turing equivalent to a Σ_1^1-complete set.*

Proof: Let $\mathit{Stab}2^+ = \{e : W_e$ is the set of codes of clauses of a finite Logic Program which has at least 2 stable models$\}$ and $\mathit{Stab}1 = \{e : W_e$ is the set of codes of clauses of a finite Logic Program which has a unique stable model$\}$. Similarly let $\mathit{Infpath}2^+ = \{e : W_e$ is a tree with at least 2 infinite paths$\}$ and $\mathit{Infpath}1 = \{e : W_e$ is a tree with at least 2 infinite paths$\}$. Now the recursive function g of Corollary 5.12 shows that $\mathit{Infpath}1$ is 1:1 reducible to $\mathit{Stab}1$ and $\mathit{Infpath}2^+$ is 1:1 reducible to $\mathit{Stab}2^+$. Moreover it is easy to see that $\mathit{Infpath}2^+$ and $\mathit{Stab}2^+$ are Σ_1^1 sets. Clearly $\mathit{Infpath}1 = \mathit{Infpath} \setminus \mathit{Infpath}2^+$ and $\mathit{Stab}1 = \mathit{Stab} \setminus \mathit{Stab}2^+$. Thus both $\mathit{Infpath}1$ and $\mathit{Stab}1$ are Turing reducible to any Σ_1^1-complete

set, in particular they are both Turing reducible to *Infpath*. Now given any r.e. set W_e, let V_e consists of the codes of all strings of 0's plus the set of all y such that y is the code of a string $(\alpha_1 + 1, \alpha_2, \ldots, \alpha_n)$ where $(\alpha_1, \alpha_2, \ldots, \alpha_n)$ is in W_e^* (here W_e^* is defined as in the proof of Corollary 5.12). It follows that V_e is a tree with at least one infinite path, namely the path $< (0), (0,0), (0,0,0), \ldots >$, and that V_e has exactly one infinite path iff W_e^* has no infinite paths. Now there is a recursive function $k(x)$ such that $V_e = W_{k(e)}$ for all e. But then $e \notin$ *Infpath* $\leftrightarrow k(e) \in$ *Infpath*1. Hence the complement of *Infpath* is 1:1 reducible to *Infpath*1. Thus we have shown that *Infpath* $\leq_T$ *Infpath*1 $\leq_T$ *Stab*1 $\leq_T$ *Infpath*. Hence *Stab*1 is Turing equivalent to a Σ_1^1-complete set.

As was the case in Corollary 5.12, the above proof works just as well if we use the set of canonical indices of finite Logic Programs rather than the set r.e. indices of finite Logic Programs. □

Acknowledgements

This research was partially supported by NSF grants IRI 9012902, DMS-8902797, DMS-8702473 and ARO contract DAAG629-85-C-0018.

References

[1] H. Andreka and I. Nemeti I. The Generalized Completeness of Horn Predicate Logic as a Programming Language. *Acta Cybernetica*, 4:3–10, 1978.

[2] 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.

[3] C. Bell, A. Nerode, R. T. Ng, and V.S. Subrahmanian. Implementing Deductive Databases by Linear Programming. Cornell Mathematical Sciences Institute Technical Report 91-48, 1991.

[4] H. Blair, W. Marek, and J. Schlipf. Expressiveness of Locally Stratified Programs. Technical report, University of Syracuse, 1991.

[5] J. de Kleer. An Assumption-based TMS. *Artificial Intelligence*, 28:127–162, 1986.

[6] J. Doyle. A Truth Maintenance System. *Artificial Intelligence*, 12:231–272, 1979.

[7] 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.

[8] M. Gelfond and H. Przymusinska. On the Relationship Between Circumscription and Autoepistemic Logic. In *Proceedings of the ISMIS Symposium*, 1986.

[9] C.G. Jockusch, A. Lewis, and J. B. Remmel. π_1^0 Classes and Rado's Selection Principle. *Journal of Symbolic Logic*, 56:684–693, 1991.

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

[11] K. Konolige. On the Relation Between Default and Autoepistemic Logic. *Artificial Intelligence*, 35:343–382, 1988.

[12] V. Lifschitz. Computing Circumscription, *Proceedings IJCAI-1985*, Morgan Kaufmann.

[13] V. Lifschitz Pointwise Circumscription, *Proceedings AIII-1986*, pages 406-410, Morgan Kaufmann.

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

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

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

[17] 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. Also available as Cornell Mathematical Sciences Institute Technical Report 91-63.

[18] 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.

[19] W. Marek and V.S. Subrahmanian. The Relationship Between Logic Program Semantics and Non-monotonic Reasoning. In *Proceedings of the 6th International Conference on Logic Programming*, 1989.

[20] J. McCarthy. Circumscription - A Form of Non-Monotonic Reasoning. *Artificial Intelligence* 13:295–323, 1980.

[21] A. Nerode and R. Shore. *Logic for Applications*. Springer-Verlag, 1993.

[22] M. Reinfrank, O. Dressler, and G. Brewka. On the Relation Between Truth Maintenance and Non-monotonic Logics. In *Proceedings of IJCAI-89*, pages 1206–1212, San Mateo, CA., 1989. Morgan Kaufmann.

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

[24] H.J. Rogers. Theory of Recursive Functions and Effective Computability McGraw-Hill, 1967.

[25] J.C. Shepherdson. Unsolvable Problems for SLDNF-resolution. *Journal of Logic Programming*, 10:19–22, 1991.

[26] R.M. Smullyan. *First-order Logic*. Springer-Verlag, 1968.

[27] 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.