Fundamenta Informaticae XXXXV (2001) 237–252 237
IOS Press

Skeptical Reasoning in FC-Normal Logic Programs is Π^1_1-complete

Robert Saxon Milnikel
Department of Mathematics
Wellesley College
106 Central St.
Wellesley, MA 02481 USA

Abstract. FC-normal logic programs are a generalization by Marek, Nerode, and Remmel of Reiter's normal default theories. They have the property that, unlike most logic programs, they are guaranteed to have simple stable models. In this paper it is shown that the problem of skeptical reasoning in FC-normal programs is Π^1_1-complete, the same complexity as for logic programs without the restriction of FC-normality.

FC-normal programs are defined in such a way as to make testing a program for FC-normality very difficult in general. A large subclass of FC-normal programs, *locally FC-normal programs*, is defined, shown to be recursive, and shown to have skeptical consequence as expressive as the entire class of FC-normal programs.

1. Introduction

FC-normal logic programs were introduced by Marek, Nerode, and Remmel in [12] as a way to generalize the idea of normality in default logic in such a way that it could be brought to bear on logic programs. In general, the problem of discovering stable models of logic programs (and that of discovering extensions of default logics) is quite difficult. In [16], Reiter introduced *normal default logics*, which were guaranteed to have extensions, one for each ordering of the rules of the default theory (with possible redundancy). (*Normal defaults* are rules which have as their only restraint that adding the conclusion not lead to an inconsistency.) FC-normal logic programs share this property, in that there is a stable model of an FC-normal logic program P for each ordering of the clauses of $\mathrm{ground}(P)$ (the ground instances of clauses of P), again with possible redundancy. The "FC" comes from the Forward Chaining Construction of [12] which takes as input an ordering of clauses of $\mathrm{ground}(P)$ and yields a stable model by a deterministic monotone procedure.

In [12], it was shown that if an FC-normal logic program P is computable, then P must have a stable model computably enumerable in $\mathbf{0}'$, the Turing degree of the halting problem. (Each step of the forward

chaining construction requires closing under a Horn program, which can be done with a $\mathbf{0}'$ oracle.) It was also shown in [13] that this bound is tight, in that for any set c.e. in $\mathbf{0}'$ there is an FC-normal program which has that set as its unique stable model.

We noted that the general problem of finding stable models is quite difficult, and this can be made more precise. The set of codes of finite predicate logic programs which possess no stable model is, for example, Π^1_1-complete. (That is, it is at the same level of the computability hierarchy as the set of codes of finite-path countably-branching trees.) Armed with this result, it is easy to show that the set of ground atomic formulas common to all stable models of a given logic program is also, in general, Π^1_1-complete. (This set is known as the set of *skeptical consequences* of the program.)

Given a computable FC-normal logic program, we know that it has at least one stable model, but we will show in this paper that the problem of finding the set of skeptical consequences an FC-normal logic program is still Π^1_1-hard, even for a very restricted class of FC-normal programs. We work with this restricted class (locally FC-normal programs) because, in general, determining whether a given program is FC-normal turns out to be almost inconceivably incomputable. The set of locally FC-normal programs is, on the other hand, only Π^0_1.

We will begin with a brief exposition of logic programming. The following section of the paper formally introduces the forward chaining construction, FC-normal logic programs, and some restricted types of FC-normal programs. The last major section will consist largely of the main theorem and its proof, that there are locally FC-normal logic programs for which skeptical reasoning is as hard as it possibly could be.

2. Review of Logic Programming Ideas

We will assume that the reader has some familiarity with logic programming, and so will move through basic definitions and results quickly. There are many sources for a more thorough background in logic programming, including [1] and [8]. With the exception of a few theorems, almost all of the text of this section will be used to define terminology, so we will not set definitions apart from the general text in this section.

2.1. Normal Predicate Logic Programs and Stable Models

Let us begin by assuming a fixed language $\mathcal{L}$ consisting of predicate letters, function symbols, variables, and constants. A *ground term* of $\mathcal{L}$ is a term composed only of constants and function symbols. A *ground atomic statement* is an atomic formula in which each of the arguments is a ground term. We will refer to the collection of all ground atomic statements of $\mathcal{L}$ as the *Herbrand base* of $\mathcal{L}$, and denote it by $B_{\mathcal{L}}$.

A *Horn program clause* (or *Horn clause*) is an expression of the form

$$c \leftarrow a_1, \ldots, a_m$$

where $a_1, \ldots, a_m, c$ are all atomic formulas of $\mathcal{L}$. (If $m = 0$, we refer to the clause as an *axiom*.) The clause is called *ground* if $a_1, \ldots, a_m, c$ are all ground atomic statements. A *definite logic program* or *Horn program* is a collection of Horn clauses. One of the basic facts of logic programming is that each Horn program has a least model in the Herbrand base, obtained as the least fixed point of an operator representing one-step deduction. (The term *model* will be formally defined shortly, but in brief, a subset

of the collection of ground atomic statements of $\mathcal{L}$ is a model of the above clause if whenever it contains the a_i's, it also contains c. It is a model of the program if it is a model of every clause of the program.)

A *program clause* is an expression of the form

$$c \leftarrow a_1, \ldots, a_m, \textbf{not}\, b_1, \ldots, \textbf{not}\, b_n$$

where $a_1, \ldots, a_m, b_1, \ldots, b_n, c$ are all atomic formulas of $\mathcal{L}$. ($m, n \geq 0$.) As for Horn clauses, a clause will be called *ground* if $a_1, \ldots, a_m, b_1, \ldots, b_n, c$ are all ground atomic statements. We will refer to c as the *head* of the clause and to

$$a_1, \ldots, a_m, \textbf{not}\, b_1, \ldots, \textbf{not}\, b_n$$

as the *body* of the clause. The a_i's are the *premises* of the clause, and the b_j's are the *restraints* of the clause. A *logic program* is a collection of normal program clauses. (These are sometimes called *normal logic programs*, but we will not use this terminology so as to avoid confusion with FC-normal logic programs.)

The words *predicate* and *propositional* can be used to modify any of the above notions of clause and program, to distinguish between the normal case and the special case in which $\mathcal{L}$ consists only of 0-ary predicates, or *propositions*. If $\mathcal{L}$ is a propositional language, we can speak of *propositional Horn programs*, for example.

A ground instance of a (predicate) clause is a clause obtained by substituting ground terms for all variables in the clause. (Of course, this must be done uniformly within the clause.) The set of all ground instances of all clauses of a predicate program P is called the *grounding* of P, denoted $\mathrm{ground}(P)$. We are now in a position to define the term *model* (as it applies to Horn programs) more precisely. A subset M of the Herbrand base is called a *model* of Horn program P if for each clause $c \leftarrow a_1, \ldots, a_m$ of $\mathrm{ground}(P)$ we have $c \in M$ whenever $\{a_1, \ldots, a_m\} \subseteq M$.)

In general, we do not assume that $\mathcal{L}$, the underlying language of a program, is fixed or known. It is worth noting, then, that if one codes clauses of $\mathrm{ground}(P)$ as numbers, then we can not conclude from the computability of the set of codes of members of $\mathrm{ground}(P)$ that $\mathcal{L}$ or the Herbrand base of $\mathcal{L}$ is also computable (only that it is computably enumerable). However, when we refer to a *computable program* in this paper, we will mean both that the set of codes of members of $\mathrm{ground}(P)$ is computable and that the Herbrand base of $\mathcal{L}$ is computable.

While the interpretation of Horn programs is well-understood and not controversial, the best interpretation of logic programs continues to be the subject of debate. We will be concentrating exclusively on one such interpretation in this paper, the *stable model semantics* for logic programs, due to Gelfond and Lifschitz ([4]). This idea is closely related to the notion of stable expansion in autoepistemic logic ([15]) and to the notion of extension in both default logics ([16]) and nonmonotone rule systems ([9]).

The idea of a stable model of a normal program P is that we tentatively assume some subset M of the Herbrand base as context, and use that to determine which clauses of $\mathrm{ground}(P)$ are relevant in that context. (Relevant in that M does not violate any of the constraints of that clause.) We then eliminate irrelevant clauses and strip the restraints of the relevant ones, leaving us with a Horn program. If the unique minimal model of this Horn program happens to be M itself, our provisionally assumed context, then M has shown itself to be a stable model of P.

More formally, we will call a clause

$$c \leftarrow a_1, \ldots, a_m, \textbf{not}\, b_1, \ldots, \textbf{not}\, b_n$$

of $\mathrm{ground}(P)$ *M-applicable* if $\{b_1, \ldots, b_n\} \cap M = \emptyset$. The *reduct* of an applicable clause

$$c \leftarrow a_1, \ldots, a_m, \textbf{not}\, b_1, \ldots, \textbf{not}\, b_n$$

is

$$c \leftarrow a_1, \ldots, a_m.$$

The set of reducts of M-applicable clauses of $\mathrm{ground}(P)$ is called the *Gelfond-Lifschitz reduct of P with respect to M*, denoted $\mathrm{GL}(M, P)$. For any program P, $\mathrm{GL}(M, P)$ is a ground Horn program, and so has a least model. If M is the least model of $\mathrm{GL}(M, P)$, then we call M a *stable model* of P.

In general, a program can have no stable models, one stable model, or many stable models. Determining which of these is the case can be very difficult.

Theorem 2.1. (Marek, Nerode, and Remmel, [11])
The set of computable programs which have no stable models is Π^1_1-complete.

Another question which can be asked about the stable models of a program is: "Is ground atomic statement a a member of every stable model of P?" The set of ground atomic statements which are members of every stable model of P is called the set of *skeptical consequences* of P; and finding the skeptical consequences of a given program is also very difficult.

Theorem 2.2. (Marek, Nerode, and Remmel, [11])
The set of pairs $\langle a, P\rangle$ where P is a computable program and a is a skeptical consequence of P is Π^1_1-complete.

2.2. Logic Programming with Classical Negation

If we extend our language with the single unary connective $\neg$, allowing atomic formulas and negated atomic formulas (together referred to as *literals*), we can define *logic programming with classical negation*. (This idea is also due to Gelfond and Lifschitz ([5]).)

A *CN-clause* is an expression of the form

$$c \leftarrow a_1, \ldots, a_m, \textbf{not}\, b_1, \ldots, \textbf{not}\, b_n$$

where $a_1, \ldots, a_m, b_1, \ldots, b_n, c$ $(m, n \geq 0)$ are all literals of $\mathcal{L}$. A *CN-program* is a set of CN-clauses.

The analogues of Horn programs, **not**-free CN-programs, also have least models (in some sense of the word), but we will have to work a little harder to define them. Our base $Lit_{\mathcal{L}}$ will now contain ground literals, not only ground atomic statements. We will call a subset M of $Lit_{\mathcal{L}}$ *closed under a ground **not**-free CN-clause*

$$c \leftarrow a_1, \ldots, a_m$$

if either

- $M = Lit_{\mathcal{L}}$, or
- M is consistent (that is, M does not contain a and $\neg a$ for any atomic statement a) and whenever $\{a_1, \ldots, a_m\} \subseteq M$ we also have $c \in M$.

Of course, we will say that M is *closed under* ***not****-free CN-program* P if M is closed under each CN-clause of $\mathrm{ground}(P)$. Gelfond and Lifschitz showed that, just as there is a least Herbrand model for a Horn program, for every **not**-free CN-program P there is a least subset of $Lit_{\mathcal{L}}$ closed under P, called an *answer set* of P.

The definition of the Gelfond-Lifschitz reduct $\mathrm{GL}(M, P)$ can be extended to CN-programs in the obvious way, and we can define a notion very similar to that of stable model. A subset M of $Lit_{\mathcal{L}}$ is a *stable answer set* of a CN-program P if M is the least subset of $Lit_{\mathcal{L}}$ closed under $\mathrm{GL}(M, P)$. Skeptical consequence for CN-programs can be defined just as for logic programs, and the results of [11] cited above carry over to CN-programs.

3. FC-Normal Logic Programs

3.1. General Case

In [12], Marek, Nerode, and Remmel defined a class of logic programs designed to be analogous to normal default logics ([16]), sharing their most important properties. Because a process called the "forward chaining" procedure is guaranteed to lead to stable models of programs in the class defined by Marek, Nerode, and Remmel, they are called *FC-normal*. We will present a brief description of the forward chaining procedure, but most of the details are not particularly important to this paper, and the reader is referred to [12] and [14] for a more complete presentation.

One aspect of the process will be important in defining FC-normal logic programs, however; that being the notion of *monotonic closure*. Logic programs under stable model semantics behave nonmonotonically; that is, if we add axioms to a program, we may *lose* conclusions because the newly added or derived information may violate restraints of a previously applicable clause. In contrast, Horn programs are monotonic, in that adding axioms never causes us to have fewer conclusions than before. For this reason, we will separate a logic program P into its Horn and non-Horn portions. We will call the set of Horn clauses of $\mathrm{ground}(P)$ the *monotonic portion of* P, denoted $\mathrm{mon}(P)$; and by the *nonmonotonic portion of* P, we will mean $\mathrm{ground}(P) \setminus \mathrm{mon}(P)$, which will be denoted $\mathrm{nmon}(P)$.

Our principal use of the above distinctions will be to define monotonic closure, which will be deductive closure under $\mathrm{mon}(P)$. The subprogram $\mathrm{mon}(P)$ is Horn by definition, and as such has a least Herbrand model. But more generally if I is any subset of the Herbrand base of the language, Horn programs have least models containing I. (One can think of the least model of the Horn program augmented with axioms $a \leftarrow$ for each $a \in I$.) The *monotonic closure* with respect to a program P of a subset I of the Herbrand base will be the least model of $\mathrm{mon}(P)$ which contains I. We will denote the monotonic closure of I by $\mathrm{Cl}_{\mathrm{mon}}(I)$.

Similar notions can be defined for CN-programs. If P is a CN-program, $\mathrm{mon}(P)$ is the set of **not**-free CN-clauses of $\mathrm{ground}(P)$, and $\mathrm{Cl}_{\mathrm{mon}}(I)$ is the least subset of $Lit_{\mathcal{L}}$ containing I and closed under $\mathrm{mon}(P)$.

To define FC-normal logic programs, we will need one more preliminary definition, but let us first motivate that preliminary definition with a brief description of the forward chaining procedure for generating stable models of a countable FC-normal logic program P. (The definition of FC-normal will be roughly "possessing the properties which insure that the forward chaining procedure yields a stable model.")

The forward chaining procedure will depend on first choosing an ordering $\prec$ of $\mathrm{nmon}(P)$ of order type ω. We then have a listing of the clauses of $\mathrm{nmon}(P)$, $\{r_n | n \in \omega\}$. We will construct an increasing sequence of sets $\{M_n^{\prec}\}_{n\in\omega}$ in stages. This given, we will then define $M^{\prec} = \bigcup_{n\in\omega} M_n^{\prec}$. Once we have this construction defined, our goal will be to define a class of programs P for which $M^{\prec}$ is a stable model for any ordering $\prec$ of $\mathrm{nmon}(P)$.

Definition 3.1. *The Countable Normal Forward Chaining Construction of* $M^{\prec}$

- Stage 0: Let $M_0^{\prec} = \mathrm{Cl}_{\mathrm{mon}}(\emptyset)$.
- Stage $n+1$: Let $s \in \omega$ be the least number such that

$$r_s = c \leftarrow a_1, \ldots, a_m, \textbf{not}\, b_1, \ldots, \textbf{not}\, b_n$$

  where $a_1, \ldots, a_m \in M_n^{\prec}$ and $b_1, \ldots, b_n, c \notin M_n^{\prec}$. If there is no such s, let $M_{n+1}^{\prec} = M_n^{\prec}$. Otherwise, let $M_{n+1}^{\prec} = \mathrm{Cl}_{\mathrm{mon}}(M_n^{\prec} \cup \{c\})$.

The preliminary notion we need before defining FC-normality is a version of consistency. FC-normal logic programs will be ones which maintain consistency as the procedure above is followed. Stable models will be consistent sets, maximal in the sense that applying any additional rule from $\mathrm{nmon}(P)$ would violate consistency. This is an extremely oversimplified description, but it should help the reader understand some of the reasons for the following definitions of consistency property and FC-normal logic program. (Both definitions are drawn from [12].)

Definition 3.2. Let P be a logic program. We say that a subset *Con* $\subseteq \mathcal{P}(B_{\mathcal{L}})$ of the power set of the Herbrand base of $\mathcal{L}$ is a *consistency property* over P if

(1) $\emptyset \in$ *Con*,

(2) $\forall A, B \subseteq B_{\mathcal{L}}[(A \subseteq B \;\&\; B \in \textit{Con}) \Rightarrow A \in \textit{Con}]$,

(3) whenever $\mathcal{A} \subseteq$ *Con* has the property that $A, B \in \mathcal{A} \Rightarrow$ $\exists C \in \mathcal{A}[A \subseteq C \;\&\; B \subseteq C]$, then $\bigcup \mathcal{A} \in$ *Con*, and

(4) $\forall A \subseteq B_{\mathcal{L}}[A \in \textit{Con} \Rightarrow \mathrm{Cl}_{\mathrm{mon}}(A) \in \textit{Con}]$.

Conditions (1)–(3) have nothing specifically to do with the program P. They are Scott's conditions for information systems (see [18]), saying that (1) the empty set is consistent, (2) subsets of consistent sets are consistent, and (3) unions of directed families of consistent sets are consistent. The need for condition (4) is reasonably evident: we would not want to consider a set consistent if an inconsistency were derivable from that set.

After all these preliminaries, we are in a position to define the objects which are the subject of this paper, FC-normal logic programs.

Definition 3.3. Let P be a logic program and let *Con* be a consistency property over P.

1. A clause $c \leftarrow a_1, \ldots, a_m, \textbf{not}\, b_1, \ldots, \textbf{not}\, b_n \in \mathrm{nmon}(P)$ is *FC-normal with respect to Con* if $V \cup \{c\} \in$ *Con* and $V \cup \{c, b_i\} \notin$ *Con* for all $i \leq n$ whenever $V \subseteq B_{\mathcal{L}}$ is such that $V \in$ *Con*, $\mathrm{Cl}_{\mathrm{mon}}(V) = V, a_1, \ldots, a_m \in V$, and $c, b_1, \ldots, b_n \notin V$.

2. A logic program P is *FC-normal with respect to Con* if every $r \in \mathrm{nmon}(P)$ is FC-normal with respect to *Con*.

3. A logic program P is *FC-normal* if for some consistency property *Con* $\subseteq \mathcal{P}(B_{\mathcal{L}})$, P is FC-normal with respect to *Con*.

An example adapted from [14] illustrates the above definitions quite well.

Example 3.1. Let $B_{\mathcal{L}} = \{a, b, c, d, e, f\}$. Let *Con* be defined by the following condition: $A \notin$ *Con* if and only if either $\{c, d\} \subseteq A$ or $\{e, f\} \subseteq A$. Thus $\{a, b, c, e\}$, $\{a, b, c, f\}$, $\{a, b, d, e\}$, and $\{a, b, d, f\}$ are the maximal subsets of $\mathcal{P}(B_{\mathcal{L}})$ which are in *Con*.

Now consider the following program, P:

(1) $a \leftarrow$

(2) $b \leftarrow c$

(3) $c \leftarrow a, \textbf{not}\, d$

(4) $e \leftarrow c, \textbf{not}\, f$.

Clauses (1) and (2) form the Horn part of P and clauses (3) and (4) form the nonmonotonic part of P. Since no Horn clauses result in c, d, e, or f, we see that closing a consistent set monotonically will not result in an inconsistency, so *Con* is a consistency property over P. To check that clause (3) is FC-normal, note that if $d \notin A$, adding c to A will not make A inconsistent. A similar argument for clause (4) establishes that P is FC-normal with respect to *Con*. It is easy to check that P has unique stable model $\{a, b, c, e\}$.

If we add to P the clause $c \leftarrow b$, to get a program P', then *Con* is not a consistency property over P', since $\{b, d\} \in$ *Con*, but $\mathrm{Cl}_{\mathrm{mon}}(\{b, d\}) = \{a, b, c, d\} \notin$ *Con*.

If we add the clause $d \leftarrow e, \textbf{not}\, f$ to P to form a new program P'', *Con* will still be a consistency property over P'' because being a consistency property depends only on the Horn part of a program. However, P'' is not FC-normal with respect to *Con* because $d \leftarrow e, \textbf{not}\, f$ is not FC-normal with respect to *Con*. If $A = \{a, b, c, e\}$, we have $A \in$ *Con*, $e \in A$, $f \notin A$, and $d \notin A$, but $\mathrm{Cl}_{\mathrm{mon}}(A \cup \{d\}) = \{a, b, c, d, e\} \notin$ *Con*.

Finally, if we add to P the clause $f \leftarrow c, \textbf{not}\, e$ then the resulting program is FC-normal with respect to *Con*, but has two stable models, $\{a, b, c, e\}$ and $\{a, b, c, f\}$.

There are many results in [12], [13], and [14] which illustrate the interest and usefulness of FC-normal programs. Here are a few:

Theorem 3.4. (Marek, Nerode, and Remmel)
If P is a countable logic program FC-normal with respect to consistency property *Con*, then

1. If P is computable, then P has a stable model computably enumerable in $\mathbf{0}'$ (where $\mathbf{0}'$ is the Turing degree of the halting problem).
2. $M^{\prec}$ is a stable model of P if $M^{\prec}$ is constructed via the Countable Normal Forward Chaining procedure with respect to $\prec$, where $\prec$ is any ordering of $\mathrm{nmon}(P)$ of order type $\leq \omega$.
3. Every stable model of P is of the form $M^{\prec}$ for some ordering $\prec$ of $\mathrm{nmon}(P)$ of order type $\leq \omega$.
4. If $A \in$ *Con* then P has a stable model M with $A \subseteq M$.
5. If M_1 and M_2 are distinct stable models of P then $M_1 \cup M_2 \notin$ *Con*.

Theorem 3.5. (Marek, Nerode, and Remmel)
Let T be a computable subtree of $2^{<\omega}$ such that $[T] \neq \emptyset$. Then there is a computable FC-normal logic program P such that there is an effective one-to-one correspondence between $[T]$ and the set of stable models of P.

There are many results from [6] and [7] about the existence of paths through computable subtrees of $2^{<\omega}$ with various degree theoretic properties, all of which now can be applied directly to stable models of FC-normal programs, but we will not list them here. (These and many other results on paths through trees are collected in [3].)

3.2. Locally FC-Normal Programs

There is one major difficulty in working with FC-normal programs in general. In Definition 3.3 we used the phrase "if for some consistency property" in defining FC-normal programs. That translates into an existential statement over a *third*-order variable. To assert the existence of a consistency property is to say that there is a set of sets of elements of the Herbrand base with certain properties, which is — on its face — at least a Σ^2_1 sentence. It may be possible to reduce this extraordinary level of complexity, but simply to assert membership in a consistency property is already apparently Π^2_1 because of clause 3 of the definition of consistency property, asserting closure under unions of directed families. That the set of FC-normal logic programs could fall into any reasonably manageable complexity class seems unlikely. There is, however, a subclass of FC-normal programs which is quite reasonable and which captures the spirit of the normal default theories which inspired the definition of FC-normal logic programs.

Definition 3.6. A logic program P will be called *locally FC-normal* if it meets the following two criteria:

1. Whenever P contains a clause which has c as its head and which has b as a restraint, then any non-Horn clause which has b as its head has c as a restraint.
2. For no clause of P does the head also appear as a restraint.

The intent of the above characterization is to mimic normal default theories by designating pairs of elements as "pairwise incompatible". In default logic (which has at its core classical logic), the pairwise incompatible elements are always p and $\neg p$, but we now have the freedom to characterize any pair of elements of the Herbrand base as pairwise incompatible. There is also nothing to prevent us from making an element pairwise incompatible with several other elements. (The second condition is a technical one, intended to keep single elements from being considered "incompatible".)

Of course, we will run into difficulty if we can deduce two pairwise incompatible elements using only the Horn part of our program, since the elements deducible from the Horn part of the program are going to be part of any stable model. To guard against the possibility of this situation leading to an inconsistency, we will refrain from calling pairs of elements in $\mathrm{Cl}_{\mathrm{mon}}(\emptyset)$ incompatible.

We have yet to show that locally FC-normal logic programs are in fact FC-normal, but this is not hard to do. The only thing we need to be careful about is the fact that if the head of a Horn clause is an element incompatible with one we already have, we also need to consider the premises of that Horn clause incompatible with the already-present element.

Proposition 3.7. Locally FC-normal logic programs are FC-normal.

Proof:
Let P be a locally FC-normal logic program. Let us first define a set of sets of incompatible elements of the Herbrand base, *Incompat*. If

$$c \leftarrow a_1, \ldots, a_m, \textbf{not}\, b_0, \textbf{not}\, b_1, \ldots, \textbf{not}\, b_n$$

with $m, n \geq 0$ is a clause of P, then for each i, $0 \leq i \leq n$, such that $b_i \notin \mathrm{Cl}_{\mathrm{mon}}(\emptyset)$ and $c \notin \mathrm{Cl}_{\mathrm{mon}}(\emptyset)$ we will say $\{b_i, c\} \in$ *Incompat*. If

$$c_0 \leftarrow a_1, \ldots, a_m$$

with $m \geq 0$ is a clause of P and $\{c_0, c_1, \ldots, c_n\} \in$ *Incompat* ($n \geq 0$), then $\{a_1, \ldots, a_m, c_1, \ldots, c_n\} \in$ *Incompat*.

Let us note here that if we were including in *Incompat* pairs $\{b, c\} \subseteq \mathrm{Cl}_{\mathrm{mon}}(\emptyset)$, backtracking through Horn clauses adding incompatible sets would lead us to include the empty set in *Incompat*. However, since we do not consider pairs $\{b, c\}$ incompatible if either $b \in \mathrm{Cl}_{\mathrm{mon}}(\emptyset)$ or $c \in \mathrm{Cl}_{\mathrm{mon}}(\emptyset)$, we know that $\emptyset \notin$ *Incompat*. (In fact, every set of incompatible elements has cardinality at least 2, since we also exclude the possibility that $b = c$.)

Now it is quite simple to define a consistency property *Con* with respect to which P is FC-normal. We will say that $A \in$ *Con* if $\forall B \subseteq A[B \notin$ *Incompat*$]$.

Let us first confirm that *Con* is a consistency property over P. That we satisfy the first three clauses of the definition of a consistency property (that consistency properties be nonempty, closed under subsets, and closed under unions of directed families) is clear from the way *Con* was defined. A set is in *Con* unless it contains multiple members of certain finite sets, and this definition of *Con* will certainly ensure that *Con* is closed under subsets and unions of directed families. That $\emptyset \in$ *Con* has already been shown. That we satisfy the final clause of the definition, closure under monotonic closure, is guaranteed by our backtracking through possible monotonic proofs of incompatible elements when we defined *Incompat*.

Recall that a clause

$$c \leftarrow a_1, \ldots, a_m, \textbf{not}\, b_0, \textbf{not}\, b_1, \ldots, \textbf{not}\, b_n$$

with $m, n \geq 0$ of P is FC-normal with respect to *Con* if for any V meeting certain conditions we have that $V \cup \{c\} \in$ *Con* and $V \cup \{c, b_i\} \notin$ *Con* for all i, $0 \leq i \leq n$. Two of the conditions on V are that $V \in$ *Con* and $b_i \notin V$ for all i, $0 \leq i \leq n$. For a V meeting these conditions, this is enough to insure that $V \cup \{c\} \in$ *Con*, since we know that $\{c\} \notin$ *Incompat*. Further, if $c, b_i \notin \mathrm{Cl}_{\mathrm{mon}}(\emptyset)$, then $\{c, b_i\} \in$ *Incompat*, and so $V \cup \{c, b_i\} \notin$ *Con*, showing the clause to be FC-normal. On the other hand, among the conditions on V is that it be monotonically closed and that $c, b_i \notin V$, so if $c \in \mathrm{Cl}_{\mathrm{mon}}(\emptyset)$ or

$b_i \in \mathrm{Cl}_{\mathrm{mon}}(\emptyset)$, then either $c \in \mathrm{Cl}_{\mathrm{mon}}(V)$ or $b_i \in \mathrm{Cl}_{\mathrm{mon}}(V)$ for any V and no V can meet the conditions, making the clause vacuously FC-normal. Thus all nonmonotonic clauses of P are FC-normal with respect to *Con*, and so P itself is FC-normal with respect to *Con*. □

This tells us, for instance, that locally FC-normal logic programs always have an extension c.e. in $\mathbf{0}'$. Let us note here that we have already seen an example of a locally FC-normal logic program, Example 3.1. (There the pairwise incompatible sets were $\{c, d\}$ and $\{e, f\}$.) The only other fact about locally FC-normal logic programs that we will want to take note of is that they are much easier to detect than general FC-normal programs, in terms of their computability theoretic degree.

Proposition 3.8. The set of locally FC-normal computable logic programs is Π^0_1 over the set of computable logic programs.

Proof:
If P is a computable logic program, one must check all pairs of clauses from P to see that they do not violate the definition of local FC-normality. Given a pair of clauses, performing this check is effective. So one can write a Π^0_1 sentence to the effect of "P is locally FC-normal if for every pair of clauses in P, the pair does not violate the definition of local FC-normality." □

Now we have a class of FC-normal logic programs no more computability theoretically complex than normal default theories and which is still a generalization of normal default theories. (It is a generalization in the sense that using the standard embedding of default logic into logic programming, normal default theories are mapped to locally FC-normal logic programs.) It is also a rich enough class that the FC-normal program whose existence is guaranteed by Theorem 3.5 can be made to be locally FC-normal.

Proposition 3.9. Let T be a computable subtree of $2^{<\omega}$ such that $[T] \neq \emptyset$. Then there is a computable locally FC-normal logic program P such that there is an effective one-to-one correspondence between $[T]$ and the set of stable models of P.

Proof:
An inspection of the proof of Theorem 3.5 (found in [12]) shows that the program constructed there is in fact already locally FC-normal. □

3.3. Default-Normal CN-Programs

To conclude this section, let us define an even more specialized version of normality for logic programs with classical negation. Of course, one can define FC-normal and locally FC-normal CN-programs in a way almost identical to the way FC-normal logic programs are defined. (The only real change is that we insist that $\mathit{Lit}_{\mathcal{L}} \notin \mathit{Con}$ for consistency properties *Con*.) But with the expressiveness of classical negation, we can also define a particularly intuitive type of normality, one closely analogous to normality in default logics. To emphasize this relationship, we will call such CN-programs default-normal.

Definition 3.10. We will define both default-normal CN-clauses and default-normal CN-programs.

- A CN-clause which is not **not**-free is called *default-normal* if it is of the form

$$c \leftarrow a_1, \ldots, a_m, \textbf{not}\, \neg c$$

or

$$\neg c \leftarrow a_1, \ldots, a_m, \textbf{not}\, c$$

for atomic statement c and literals $a_1, \ldots, a_m$.

- A CN-program is called *default-normal* if every clause in $\mathrm{nmon}(P)$ is default-normal and in addition, $\mathrm{Cl}_{\mathrm{mon}}(\emptyset) \neq \mathit{Lit}_{\mathcal{L}}$.

We include the condition that $\mathrm{Cl}_{\mathrm{mon}}(\emptyset) \neq \mathit{Lit}_{\mathcal{L}}$ so that we can consider default-normal CN-programs as a special case of FC-normal CN-programs as defined in [12]. The only programs we lose to this condition are the programs whose only stable model is the entire set of literals because the monotonic portion of the program is inconsistent. And now if we consider the consistency property *Con* to be defined by $A \notin \mathit{Con}$ if and only if there is some ground atomic statement c such that both $c \in A$ and $\neg c \in A$, then default-normal coincides with FC-normal with respect to *Con*. Default-normality is also a special case of local FC-normality.

Our reason for defining default-normal CN-programs is that when one embeds a CN-program into default logic in the standard way, default-normal CN-programs map to normal default theories. This will be useful to us in the next section, since the FC-normal logic program constructed can be slightly modified to become a default-normal CN-program, and so we will be able to include a result about default logic as an almost effortless corollary. (For an example of a default-normal CN-program, see the example of an FC-normal logic program above (Example 3.1) and replace d by $\neg c$ and f by $\neg e$.)

4. Skeptical Reasoning for FC-Normal Programs

Having seen how useful and well-behaved FC-normal logic programs are in some contexts, we will now show that they are as badly behaved as possible when it comes to skeptical reasoning. (That is, they are as computability theoretically complex as logic programs in this regard.) The main theorem of this section is about coding countably branching trees into locally FC-normal logic programs. We will assume the reader is comfortable with basic terminology about finite sequences and trees. (See [3] for a good exposition.)

In the present context, by a *tree* τ we will mean a nonempty collection of finite sequences of numbers so that if sequence σ' is an initial segment of sequence $\sigma \in \tau$ ($\sigma' \subseteq \sigma$) then σ' is in τ as well. An *infinite path* through tree τ is a collection of sequences from τ closed under initial segments which contains exactly one sequence of length k for each natural number k. A tree τ will be referred to as *finite-path* if there are no infinite paths through τ. One very useful piece of notation will be $|\sigma|$ for the length of a finite sequence σ.

Theorem 4.1. Given a computable tree $\tau \subseteq \omega^{<\omega}$, there is a locally FC-normal finite predicate logic program P_τ and a proposition *pathIsFinite* in the language of P_τ such that *pathIsFinite* is a member of every stable model of P_τ if and only if τ is a finite-path tree.

Proof:
The idea of the proof is to use the nonmonotonic part of P_τ only to choose a path π through the full tree $\omega^{<\omega}$, which will serve as a potential path through τ. Every path π through $\omega^{<\omega}$ will correspond to a stable model of P_τ. If π is not an infinite path through τ, we will take note of this with the proposition *pathIsFinite*. If *pathIsFinite* appears in every stable model of P_τ, then there are no infinite paths through τ; and if *pathIsFinite* does not appear in the stable model of P_τ corresponding to path π, then π is an infinite path through τ.

If we choose some standard coding for finite sequences of integers, then the usual way of representing computable relations by Horn clauses yields the following results (for a computable tree τ):

- There exists a finite predicate Horn program P_τ^0 such that for a predicate *notInTree*$(\cdot)$ of the language of P_τ^0, the atom *notInTree*(n) belongs to the least Herbrand model of P_τ^0 if and only if n is a code for a finite sequence σ and $\sigma \notin \tau$. (n is an abbreviation of the term $s^n(0)$.)
- There is a finite predicate 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 sequence σ.
- There is a finite predicate Horn program P^2 which correctly computes several notions for manipulating predicates and functions on sequences, including:
  (a) *sameLength*$(\cdot,\cdot)$. This succeeds if and only if both arguments are 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 if the arguments are codes of sequences and the first sequence is shorter than the second sequence.
  (d) *notIncluded*$(\cdot,\cdot)$. This succeeds if and only if both arguments are codes of sequences and the first sequence is not an initial segment of the second sequence.

Let us denote by P_τ^- the finite predicate Horn program which is the union of programs P_τ^0, P^1, and P^2, its language by $\mathcal{L}^-$, and the least Herbrand model of P_τ^- by M_τ^-. After we add the four clauses below, the resulting program will be a finite predicate program. Those additional clauses will not contain any predicates of $\mathcal{L}^-$ in the head, so whatever stable model of the extended program we consider, the subset of the set of ground atoms of $\mathcal{L}^-$ contained in that stable model will always be M_τ^-. In particular, the meaning of each of the predicates listed above will always be the same.

Before we finish defining P_τ, we will need two unary predicates and one proposition:

- *inPath*$(\cdot)$, having as its intended interpretation the set of codes of sequences forming a path π through $\omega^{<\omega}$.
- *notInPath*$(\cdot)$, having as its intended interpretation the set of codes of sequences not in π.
- *pathIsFinite*, which will indicate that π is not an infinite path through τ.

Here are the final four clauses of P_τ:

1. *inPath*$(X) \longleftarrow$ *seq*(X), **not** *notInPath*(X)
2. *notInPath*$(X) \longleftarrow$ *inPath*(Y), *sameLength*(X, Y), *diff*(X, Y)
3. *notInPath*$(X) \longleftarrow$ *inPath*(Y), *shorter*(Y, X), *notIncluded*(Y, X)
4. *pathIsFinite* $\longleftarrow$ *inPath*(X), *notInTree*(X)

We will denote by P_τ the union of P_τ^- with the above four clauses, and the language of P_τ by $\mathcal{L}$. We need to establish the following:

I That the stable models of P_τ are in one-to-one correspondence with paths through $\omega^{<\omega}$.

II That *pathIsFinite* will appear in the stable model corresponding to path π if and only if π is not an infinite path through τ.

III That P_τ is indeed locally FC-normal.

The validity of item (II) is fairly easy to see. If we are attempting to follow a path π through τ and find ourselves at a point when $\sigma \in \pi$ but $\sigma \notin \tau$, we take note of the event by including the proposition *pathIsFinite*; this is the only circumstance under which *pathIsFinite* can be derived.

Next, we address item (III). That P_τ is locally FC-normal is immediate upon inspection. (Any program in which the restraints of nonmonotone rules never appear as heads of nonmonotone rules is trivially locally FC-normal.)

The rest of this proof is an argument to establish item (I) and is rather tedious and involved. If the reader is convinced by a more casual inspection of the rules of P_τ that there will be a one-to-one correspondence between stable models of P_τ and paths through the tree $\omega^{<\omega}$, then she might wish to skip to the end of the proof.

To establish the validity of item (I), let us first assume that M is a stable model of P_τ. We will show that the set of n such that *inPath*$(n) \in M$ codes a path through $\omega^{<\omega}$. First of all, let us show that there must be at least one such n. Every rule with head *notInPath*(n) has *inPath*(n') in the head for some n'. If we could not prove *inPath*(n') for *any* n', we could not prove *notInPath*(n) for any n, and we could use the clause

$$\textit{inPath}(n) \leftarrow \textit{seq}(n), \textbf{not}\ \textit{notInPath}(n)$$

to prove *inPath*(n). So there must be at least one n such that *inPath*$(n) \in M$.

Still assuming that M is a stable model of P_τ, now let us show that if *inPath*$(n) \in M$, n codes a sequence σ, and n' codes a sequence σ' with $|\sigma'| \leq |\sigma|$, then *inPath*$(n') \in M$ if and only if $\sigma' \subseteq \sigma$. If it is not the case that $\sigma' \subseteq \sigma$, then we have an immediate proof of *notInPath*(n'). With a proof of *notInPath*(n') we will not be able to prove *inPath*(n'). Now let us show that if $\sigma' \subseteq \sigma$ we will not have a proof of *notInPath*(n'). Assume we did have a proof of *notInPath*(n'). Either we used *inPath*(n'') for n'' coding a sequence σ'' with $|\sigma''| = |\sigma'|$ but $\sigma'' \neq \sigma'$; or we used *inPath*(n'') for n'' coding a sequence σ'' with $|\sigma''| \geq |\sigma'|$ and σ'' not extending σ'. If $|\sigma''| = |\sigma'|$ but $\sigma'' \neq \sigma'$, then we have a proof of *notInPath*(n'') and so could not have proved *inPath*(n''). Thus $|\sigma''| > |\sigma'|$, but σ'' does not extend σ'. If $|\sigma''| \leq |\sigma|$, then we have a proof of *notInPath*(n'') and so could not have proved *inPath*(n''). So it must be the case that $|\sigma''| > |\sigma|$ but σ'' does not extend σ (since $\sigma' \subseteq \sigma$ and σ'' does not extend σ'). That would give us a proof of *notInPath*(n), which would have made it impossible to prove *inPath*(n).

Our assumption that *notInPath*(n') had a proof leads inevitably to a contradiction, so *notInPath*(n') has no proof if *inPath*$(n) \in M$ and n and n' code sequences σ and σ' with $\sigma' \subseteq \sigma$. With no proof of *notInPath*(n'), we will have a proof of *inPath*(n'). So if *inPath*$(n) \in M$ and n codes σ, then for *every* n' coding $\sigma' \subseteq \sigma$ we will have *inPath*$(n') \in M$.

Finally, let us show that if M is a stable model then there is no upper bound on the length of sequences coded by n such that *inPath*$(n) \in M$. If there were an upper bound k, then there would have to be a proof of *notInPath*(n) for every n coding a sequence of length $k+1$. But each of these would have required *inPath*$(n') \in M$ for some n' coding a sequence of length greater than or equal to $k+1$ to prove, and k was the upper bound of lengths of sequences such that *inPath*$(n) \in M$ and n codes a sequence of length k. So there is no such k. By the above arguments, we have shown that if M is a stable model of P_τ, then the set of n such that *inPath*$(n) \in M$ code a chain of nested sequences, one sequence of length k for each $k \in \omega$. That is, the set of n such that *inPath*$(n) \in M$ codes a path through $\omega^{<\omega}$.

Now let π be a path through $\omega^{<\omega}$. It is easy to see that $M = \mathrm{Cl}_{\mathrm{mon}}(\{\textit{inPath}(n) | n \text{ codes } \sigma \in \pi\})$ is a stable model of P_τ. A close examination of the rules makes evident that *notInPath*(n) can not be derived if, for every n' such that *inPath*(n') has already been derived, n and n' code sequences σ and σ' with either $\sigma \subseteq \sigma'$ or $\sigma' \subseteq \sigma$. Thus M will not contain *notInPath*(n) for any n coding a $\sigma \in \pi$. That is enough to guarantee the derivability (using the context of M) of *inPath*(n) for all n coding $\sigma \in \pi$. The rest of M must also have derivations using M as context, as the rest of M was derived monotonically from $\{\textit{inPath}(n) | n \text{ codes } \sigma \in \pi\}$. □

By changing every occurrence of *notInPath* to $\neg$ *inPath* in P_τ, we obtain a CN-program P_τ^{CN} with identical properties to P_τ. With only one **not** in the program, it is easy to check that the program is default-normal, yielding the following corollaries:

Corollary 4.2. Given a computable tree $\tau \subseteq \omega^{<\omega}$, there is a finite default-normal predicate CN-program P_τ^{CN} and a proposition *pathIsFinite* in the language of P_τ^{CN} such that *pathIsFinite* is a member of every stable answer set of P_τ^{CN} if and only if τ is a finite-path tree.

Corollary 4.3. Given a computable tree $\tau \subseteq \omega^{<\omega}$, there is a normal default theory D_τ and a proposition *pathIsFinite* in the language of D_τ such that *pathIsFinite* is a member of every extension of D_τ if and only if τ is a finite-path tree.

Since skeptical consequence for logic programs, CN-programs, and default logics is known to be Π_1^1-complete ([10]) (and so in particular in the class of Π_1^1 problems); and since the set of finite-path computable trees is Π_1^1-complete ([17]), we have the following final corollary:

Corollary 4.4. Skeptical consequence for FC-normal logic programs, for default-normal CN-programs, and for normal default theories is Π_1^1-complete. That is:

- The set of pairs $\langle a, P \rangle$ where P is a computable locally FC-normal logic program and a is a member of every stable model of P is Π_1^1-complete.
- The set of pairs $\langle a, P \rangle$ where P is a computable default-normal CN-program and a is a member of every stable answer set of P is Π_1^1-complete.
- The set of pairs $\langle \varphi, D \rangle$ where D is a computable normal default theory and φ is a member of every extension of D is Π_1^1-complete.

5. Conclusions and Further Research

FC-normal logic programs are quite useful, in that for many of the applications for which stable model logic programming was introduced in the first place, we are principally concerned with finding *some* stable model of the program. But when we want to find out what *must* be true in any stable model, we gain nothing by restricting our attention to FC-normal logic programs, or even to locally FC-normal programs.

One big question, then, is whether there is a class of non-Horn programs that has both easily-identified members and skeptical reasoning of complexity less than Π^1_1. (Locally stratified programs have unique stable models which are easy to find once the stratification has been determined; but the set of locally stratified programs is itself Π^1_1-complete. See [2].) A precise description of the class of sets of sets which can be coded as the stable models of an FC-normal program (or of a locally FC-normal program) has not, as far as I know, been put forward either. More straightforward work stemming from the definitions set out in this paper would be to determine which results about normal default theories carry across to locally FC-normal logic programs, and to examine the finite computational complexity of locally FC-normal finite propositional logic programs.

I thank Denis Hirschfeldt for insightful comments and a careful reading of a preliminary version of this paper.

References

[1] Apt, K. R.: Logic programming, in: *Handbook of Theoretical Computer Science* (J. van Leeuven, Ed.), Cambridge, MA, MIT Press, 1990, 493–574.

[2] Cholak, P., Blair, H.: The complexity of local stratification, *Fund. Inform.*, **21**, 1994, 333–344.

[3] Cenzer, D., Remmel, J. B.: Π^0_1 classes in mathematics, in: *Handbook of Recursive Mathematics, Vol. 2* (Y. L. Ershov, S. S. Goncharov, V. W. Marek, A. Nerode, J. B. Remmel, Eds.), Amsterdam, North-Holland, 1998, 623–821.

[4] Gelfond, M., Lifschitz, V.: The stable semantics for logic programs, *Proceedings of the 9th Annual Symposium on Logic Programming* (R. Kowalski, K. Bowen, Eds.), Cambridge, MA, MIT Press, 1988, 1070–1080.

[5] Gelfond, M., Lifschitz, V.: Logic programs with classical negation, *Proceedings of the 7th International Conference on Logic Programming* (D. Warren, P. Szeredi, Eds.), Cambridge, MA, MIT Press, 1990, 579–597.

[6] Jockusch, C., Soare, R. I.: Degrees of members of Π^0_1 classes, *Pacific J. Math.*, **40**, 1972, 605–616.

[7] Jockusch, C., Soare, R. I.: Π^0_1 classes and degrees of theories, *Trans. Amer. Math. Soc.*, **173**, 1972, 33–56.

[8] Lloyd, J.: *Foundations of Logic Programming*, Berlin, Springer-Verlag, 1984, 2nd edition, 1987.

[9] Marek, W., Nerode, A., Remmel, J. B.: Nonmonotonic rule systems I, *Ann. Math. Artificial Intelligence*, **1**, 1990, 241–273.

[10] Marek, W., Nerode, A., Remmel, J. B.: Nonmonotonic rule systems II. *Ann. Math. Artificial Intelligence*, **5**, 1992, 229–263.

[11] Marek, W., Nerode, A., Remmel, J. B.: The stable models of a predicate logic program. *Journal of Logic Programming*, **21**, 1994, 129–154.

[12] Marek, W., Nerode, A., Remmel, J. B.: Context for belief revision: FC-normal nonmonotonic rule systems, *Ann. Pure Appl. Logic*, **67**, 1994, 269–324.

[13] Marek, W., Nerode, A., Remmel, J. B.: Complexity of recursive normal default logic, *Fund. Inform.*, **32**, 1997, 139–147.

[14] Marek, W., Nerode, A., Remmel, J. B.: Logic programs, well-orderings, and forward chaining, *Ann. Pure Appl. Logic*, **96**, 1999, 231–276.

[15] Moore,R. C.: Possible-world semantics for autoepistemic logic, *Proceedings of the Workshop on Non-Monotonic Reasoning* (R. Reiter, Ed.), 1984, 344–354.

[16] Reiter, R.: A logic for default reasoning, *Artificial Intelligence*, **13**, 1980, 81–132.

[17] Rogers, H.: *Theory of Recursive Functions and Effective Computability*, New York, McGraw-Hill, 1967.

[18] Scott, D.: Domains for denotational semantics, *Proceedings of ICALP-82*, Heidelberg, Springer, 1982, 577–613.