Basic Forward Chaining Construction for Logic Programs

V.W. Marek[1⋆], A. Nerode[2⋆⋆], J.B. Remmel[3⋆⋆⋆]

[1] Department of Computer Science, University Kentucky, Lexington, KY 40506–0027.
[2] Mathematical Sciences Institute, Cornell University, Ithaca, NY 14853.
[3] Department of Mathematics, University of California at San Diego, La Jolla, CA 92903.

1 Introduction and Motivation

One of the problems which motivated this paper is how do we deal with inconsistent information. For example, suppose that we want to develop an expert system using logic programming with negation as failure. It may be the case that the knowledge engineer gathers facts, i.e. clauses of the form $p \leftarrow$, rules without exceptions, i.e. clauses of the form $p \leftarrow q_1, \ldots q_n$, and rules with exception or rules of thumb, i.e. clauses of the form $p \leftarrow q_1, \ldots q_n, \neg r_1, \ldots, \neg r_m$, from several experts. One problem is that the resulting program may be inconsistent in the sense that the program has no stable model. That is, the experts may not be consistent. The question then becomes how can we eliminate some of the clauses so that we can get a consistent program. That is, at a minimum, we would like to select a subprogram of the original program which has a stable model. Various schemes have been proposed in the literature to do this [GS92, KL89]. For example, we may throw away the rules which came from what we feel are the most unreliable experts until we get a consistent program. However even in the case when the knowledge engineer consults only a single expert, the rules that the knowledge engineer produces may be inconsistent because the rules that he or she abstracted are not specific enough or simply because the expert did not give us a consistent set of rules.

The above scenario is one practical reason that we would desire some procedure to construct, for a given program which has no stable model, a maximal subprogram that does have a stable model. Another practical reason occurs when we are using a logic program to control a plant in real time, see [KN93a] for examples. In this case, the program may have a stable model but that stable model may be very complicated and we do not have enough time to compute the full stable model. It has been shown [MT91] that the problem of determining whether a finite propositional logic program has a stable model is NP-complete. Moreover, the authors have shown [MNR92a] that there are finite predicate logic

⋆ Research partially supported by NSF grant IRI-9400568.
⋆⋆ USARO MURI DAAH-04-96-10341, Center for Foundations of Intelligent Systems at Cornell University.
⋆⋆⋆ Research partially supported by NSF grant DMS-93064270.

programs which have stable models but which have no stable models which are hyperarithmetic so that there is no possible hope that one could compute the a stable model of the program no matter how much time one has. Thus if there are time problems, one may be satisfied by a procedure which would construct a subprogram of the original program and a stable model of the subprogram as long as both the subprogram and stable model of the subprogram can be computed rapidly, at the very least in polynomial time.

Indeed some see as a general problem with the stable model semantics the fact that there are many programs which have no stable models. For example, if we have any program P and p is new statement letter, the program P plus the clause $p \leftarrow \neg p$ has no stable model even if the original program P has a stable model. Thus a single superfluous clause which may have nothing to do with the rest of the program may completely destroy the possibility of the program possessing a stable model. This is one of the reasons that researchers have looked for alternatives to the stable model semantics such as the well-founded semantics [VGRS91].

In this paper, we shall present a basic Forward Chaining type construction which can be applied to any general logic program. The input of the construction will be any well-ordering of the non-Horn clauses of the program. The construction will then output a subprogram of the original program and a stable model of the subprogram. It will be the case that for any stable model M of the original program P, there will be a suitable ordering of the non-Horn clauses of the program so that the subprogram produced by our construction is just P itself and the stable model of subprogram produced by our construction will be M. Thus all stable models of the original program will be constructed by our Forward Chaining construction for suitable orderings. Moreover, we shall show that for finite propositional logic programs, our construction will run in polynomial time. That is, we shall prove that our Forward Chaining construction runs in order of the square of the length of the program.

We shall see that any stable model M of P can be produced via our Forward Chaining construction for some well-ordering $\prec$, i.e. every stable model of P is a stable submodel of P. In the case where our original program P is inconsistent in the sense that P has no stable models, we can view our Forward Chaining construction as a way of extracting a maximal consistent subset of clauses $C^{\prec} \subseteq P$ such that the system $C^{\prec}$ has stable model.

2 General logic programs

A *definite logic program* consists of clauses of the form

$$a \leftarrow a_1, \ldots, a_m$$

where a, $a_1, \ldots, a_m$ are atoms of some underlying language. We call such clauses *Horn program clauses* or simply Horn clauses. The set of atoms occurring in some clause of P is called the Herbrand base of P, and is denoted by H_P. We will be dealing here with the propositional case only.

A *general logic program* consists of clauses of the form

$$C = a \leftarrow a_1, \ldots, a_m, \neg b_1, \ldots, \neg b_n. \tag{1}$$

where $a_1, \ldots, a_m$, $b_1, \ldots, b_n$ are atoms.. Here $a_1, \ldots, a_n$ are called the *premises* of clause C, $b_1, \ldots, b_m$ are called the *constraints* of clause C, and a is called the *conclusion* of clause C.

Each Horn program can be identified with the a general program in which every clause has an empty set of constraints.

Definition 1. A subset $M \subseteq H_P$ is called a *model* of P if for all $C = a \leftarrow a_1, \ldots, a_m \neg b_1, \ldots, \neg b_n \in P$, whenever all the premises $a_1, \ldots, a_n$ of C are in M and all the constraints $b_1, \ldots, b_m$ of C are not in M, then the conclusion a of C belongs to M.

Given sets $M \subseteq H_P$ and $I \subseteq H_P$, an *M-deduction* of c from I in P is a finite sequence $\langle c_1, \ldots, c_k \rangle$ such that $c_k = c$ and for all $i \leq k$, each c_i either
(1) belongs to I, or (2) is the conclusion of an axiom, or
(3) is the conclusion of a clause $C \in P$ such that all the premises of C are included in $\{c_1, \ldots, c_{i-1}\}$ and all constraints of C are in $H_P \setminus M$ (see [MT93], also [RDB89]).
An *M-consequence* of I is an element of H_P occurring in some M-deduction from I. Let $C_M(I)$ be the set of all M-consequences of I in P. Clearly I is a subset of $C_M(I)$. However note that M enters solely as a restraint on the use of the clauses which may be used in an M-deduction from I. M contributes no members directly to $C_M(I)$, although members of M may turn up in $C_M(I)$ by an application of a clause which happens to have its conclusion in M. For a fixed M, the operator $C_M(\cdot)$ is monotonic. That is, if $I \subseteq J$, then $C_S(M) \subseteq C_M(J)$. Also, $C_M(C_M(I)) = C_M(I)$. However, for fixed I, the operator $C_M(I)$ is anti-monotonic in the argument M. That is if $M' \subseteq M$, then $C_M(I) \subseteq C_{M'}(I)$.

We say that $M \subseteq H_P$ is *grounded* in I if $M \subseteq C_M(I)$. We say that $M \subseteq H_P$ is a *stable model of* P over I of I if $C_M(I) = M$.

With each clause C of form (1), we associate a Horn clause of form (2)

$$C' = a \leftarrow a_1, \ldots, a_m \tag{2}$$

obtained from C by dropping all the constraints. The clause C' is called the *projection* of clause C. Let M be any subset of H_P and let $G(M, P)$ be the collection of all M-applicable clauses. That is, a clause C belongs to $G(M, P)$ if all the premises of C belong to M and all constraints of C are outside of M. We write $P|_M$ for the collection of all projections of all clauses from $G(M, P)$. The projection $P|_M$ is a Horn program. Our definition of stable model was different from but equivalent to that given by Gelfond and Lifschitz in [GL88].

3 The Forward Chaining Construction and Stable Submodels

Given a general program P, we then let $mon(P)$ denote the set of all Horn clauses of P and $nmon(P) = P \setminus mon(P)$. The elements of $nmon(P)$ will be

called *nonmonotonic* clauses.

Our Forward Chaining construction will take as an input a program P and a well-ordering $\prec$ of $nmon(P)$. The principal output of the Forward Chaining construction will be a subset $D^{\prec}$ of H_P. Although such subset is not, necessarily, a stable model of P, it will be a stable model of $A^{\prec}$ for a subset $A^{\prec} \subseteq P$. This subset, $A^{\prec}$, will also be computed out of our construction and will be the maximal set of clauses of P for which $D^{\prec}$ is a stable model. We thus call $D^{\prec}$ a *stable submodel* of P.

The first feature of our construction is that in every stage of our construction we will close the sets we construct under $mon(P)$. The point is that stable models are always closed under the operator associated with the Horn part of the program, and the applicability of a clause from $mon(P)$ is not restricted. We shall denote by $cl_{\rm mon}$ the monotone operator of closure under the clauses in $mon(P)$. Thus $cl_{\rm mon}(I) = T_{mon(P)} \uparrow \omega(I)$ is the least set Z of atoms from H_P such that $I \subseteq Z$ and Z is closed under every clause r of $mon(P)$. That is, if premises of such a clause are all in Z, then its conclusion also belongs to Z. The second important aspect of our construction is that when we inspect the clauses of $nmon(P)$ for a possible application, we look at the possible effect of their application on the applicability of those clauses which were previously applied. Rules that may invalidate applicability of previously used clauses are *not* used.

The execution of this idea requires some book-keeping. Our Forward Chaining construction will define two sequences of subsets of H_P: $\langle D^{\prec}_{\xi}\rangle_{\xi \leq |P|^+}$ and $\langle R^{\prec}_{\xi}\rangle_{\xi \leq |P|^+}$. $D^{\prec}_{\xi}$ will be the set of *elements derived* by stage ξ. $R^{\prec}_{\xi}$ will be the set of *elements restrained* by stage ξ. Here and below α^+ is the least cardinal greater than α. Thus, if P is countable, then $|P|^+$ is either finite or the first uncountable ordinal. We shall prove, however, that if $|P|$ is countably infinite, then the construction actually stops *below* the first uncountable ordinal and therefore, for denumerable P, the use of nondenumerable cardinals can be eliminated.

In addition, we shall define two sets of clauses, $I^{\prec}$ (for "inconsistent clauses") and $A^{\prec}$ (for "acceptable" clauses). These sets of clauses will depend on previously defined hierarchies.

3.1 Forward Chaining Construction

Definition 2. Let P be a general program and let $\prec$ be a well-ordering of $nmon(P)$. We define two sequences of sets of atoms from H_P, $\langle D_{\xi}\rangle$ as well as $\langle R_{\xi}\rangle$. The set D_{ξ} is the set of atoms *derived* by stage ξ and R_{ξ} is the set of atoms *rejected* by the stage ξ.

1. $D^{\prec}_0 = cl_{\rm mon}(\emptyset)$, $R^{\prec}_0 = \emptyset$;
2. If $\gamma = \beta + 1$ and there is a clause $C \in nmon(P)$ such that

$$prem(C) \subseteq D^{\prec}_{\beta}, \quad (\{c(C)\} \cup cons(C)) \cap D^{\prec}_{\beta} = \emptyset$$

and

$$cl_{\rm mon}(D^{\prec}_{\beta} \cup \{c(C)\}) \cap (cons(C) \cup R^{\prec}_{\beta}) = \emptyset$$

(we call such clause *applicable clause*), then let C_γ be the $\prec$-first applicable clause and set

$$D^\prec_\gamma = cl_{\mathrm{mon}}(D^\prec_\beta \cup \{c(C_\gamma)\}) \quad R^\prec_\gamma = R^\prec_\beta \cup cons(C_\gamma).$$

If there is no C such that

$$prem(C) \subseteq D^\prec_\beta, \quad (\{c(C)\} \cup cons(C)) \cap D^\prec_\beta = \emptyset$$

and

$$cl_{\mathrm{mon}}(D^\prec_\beta \cup \{c(C)\}) \cap (cons(C) \cup R^\prec_\beta) = \emptyset,$$

then set

$$D^\prec_\gamma = D^\prec_\beta \quad \text{and} \quad R^\prec_\gamma = R^\prec_\beta$$

3. If γ is a limit ordinal, then

$$D^\prec_\gamma = \bigcup_{\xi<\gamma} D^\prec_\xi \quad \text{and} \quad R^\prec_\gamma = \bigcup_{\xi<\gamma} R^\prec_\xi.$$

4. Finally let

$$D^\prec = D^\prec_{|P|^+} = \bigcup_{\xi<|P|^+} D^\prec_\xi \quad \text{and} \quad R^\prec = R^\prec_{|P|^+} = \bigcup_{\xi<|P|^+} R^\prec_\xi.$$

Sets $D^\prec$ and $R^\prec$ are sets of atoms *derived* and *rejected* during the forward chaining construction along the well-ordering $\prec$.

We define the set of inconsistent clauses, $I^\prec$, and the set of consistent clauses, $A^\prec$, relative to ordering $\prec$ as follows:

5. C is *inconsistent with* $\prec$ (or simply *inconsistent* if $\prec$ is fixed) if $prem(C) \in D^\prec$, $(\{c(C)\} \cup cons(C)) \cap D^\prec = \emptyset$, but $cl_{\mathrm{mon}}(D^\prec \cup \{c(C)\}) \cap (cons(C) \cup R^\prec) \neq \emptyset$. $I^\prec = \{C \in P : \ C \text{ is inconsistent with } \prec\}$;
6. $A^\prec = P \setminus I^\prec$

We then say that a subset $D \subseteq H_P$ is a *stable submodel* of P, if there is a well-ordering $\prec$ of $nmon(P)$ such that $D = D^\prec$.

The following observations should be clear: First, the clause that is used for construction of $D^\prec_{\gamma+1}$ from $D^\prec_\gamma$ is different from any clause used before in the construction. Therefore, by cardinality argument, the construction, eventually, stabilizes.

Next, both hierarchies $\langle D^\prec_\xi \rangle$ and $\langle R^\prec_\xi \rangle$ are increasing. Moreover, it is easy to prove by induction on ξ that $D^\prec_\xi \cap R^\prec_\xi = \emptyset$. Therefore $D^\prec \cap R^\prec = \emptyset$.

The sets $R^\prec_\xi$ accumulate the restraints of all clauses applied during the construction. Since $D^\prec \cap R^\prec = \emptyset$, the applicability of clauses applied during the construction is preserved at the end. This immediately implies the following result. First, let $P_\alpha = \{C_\xi : \xi < \alpha\}, P^\star = \{C_\alpha : \alpha < |P|^+ \text{ and } C_\alpha \text{ is defined}\}$. We have

Proposition 3. *$D_\xi^\prec$ is a stable model of P_ξ, and $D^p rec$ is a stable model of $P^\star$.*

We now have a result showing that the set $D^\prec$ we produced in the Forward Chaining construction behaves as promised:

Theorem 4. *Let P be a general program. Let $\prec$ be a well-ordering of $nmon(P)$. Then $D^\prec$ is a stable model of $A^\prec$. Hence if $I^\prec = \emptyset$, then $D^\prec$ is a stable model of P.*

We define the set of nonmonotonic generating clauses for a set $M \subseteq H_P$, $NG(M, P)$.

Definition 5. Let P be a general program. Let $M \subseteq H_P$.

$$NG(M, P) = \{C \in nmon(P) : prem(C) \subseteq M, cons(C) \cap M = \emptyset\}$$

Theorem 6. *If P is a general program, then every stable model of P is a stable submodel of P. That is, if M is a stable model of P, then there exists a well-ordering $\prec$ of $nmon(P)$ such that $D^\prec = M$. In fact, for every well-ordering $\prec$ such that $NG(M, P)$ forms an initial segment of $\prec$, $D^\prec = M$.*

While we stated Theorem 4 and Theorem 6 in full generality, we are most interested in the case when program P is finite or countable. In this case we can show that to construct stable models via forward chaining, one need consider orderings of type smaller or equal of order type ω.

Proposition 7. *Let P be a program such that $|H_P| \leq \omega$ and let M be a stable model of P. There exists a well-ordering $\prec'$ of $nmon(P)$ in type $\leq \omega$ such that $D^{\prec'} = M$. Moreover the forward Chaining construction stabilizes in at most ω steps.*

We note that Proposition 7 does not hold for all stable submodels. That is, the sets $D^\prec$ which are *not* stable models may have the property that they can only be obtained by means of orderings of the length $> \omega$.

Our construction of the set $D^\prec$ persists with respect to prolongation of the well-ordering (providing the Horn part is the same).

Proposition 8. *Let $P \subset P'$ be two sets of clauses such that $mon(P) = mon(P')$. Let $\prec'$ be a well-ordering of $nmon(P')$ and let $nmon(P)$ be an initial segment in $\prec'$. Finally, let $\prec = \prec' |_P$. Then $D^\prec \subseteq D^{\prec'}$ and $R^\prec \subseteq R^{\prec'}$.*

4 Complexity of Stable Submodels

4.1 Preliminaries

Let ω denote the set of natural numbers. The canonical index, $can(X)$, of finite set $X = \{x_1 < \ldots < x_n\} \subseteq \omega$ is defined as $2^{x_1} + \ldots + 2^{x_n}$ and the canonical index of $\emptyset$ is defined as 0. Let D_k be the finite set whose canonical index is k, i.e., $can(D_k) = k$.

We shall identify a clause r with a triple $\langle k, l, \varphi \rangle$ where $D_k = prem(r)$, and $D_l = cons(r)$, $\varphi = c(r)$. In this way, when $H_P \subseteq \omega$ we can think about P as a subset of ω as well. This given, we then say that a program P is *recursive* if H_P and P are recursive subsets of ω.

Next we shall define various types of recursive trees and Π^0_1 classes. Let $[,]: \omega \times \omega \to \omega$ be a 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. Extend our pairing function to code n-tuples for $n > 2$ by the usual inductive definition, that is, let $[x_1, \ldots, x_n] = [x_1, [x_2, \ldots, x_n]]$ for $n \geq 3$. Let $\omega^{<\omega}$ be the set of all finite sequences from ω and let $2^{<\omega}$ be the set of all finite sequences of 0's and 1's. Given $\alpha = \langle \alpha_1, \ldots, \alpha_n \rangle$ and $\beta = \langle \beta_1, \ldots, \beta_k \rangle$ in $\omega^{<\omega}$, write $\alpha \sqsubseteq \beta$ if α is initial segment of β, i.e. , if $n \leq k$ and $\alpha_i = \beta_i$ for $i \leq n$. In this paper, we identify each finite sequence $\alpha = \langle \alpha_1, \ldots, \alpha_n \rangle$ with its code $c(\alpha) = [n, [\alpha_1, \ldots, \alpha_n]]$ in ω. Let 0 be the code of the empty sequence $\emptyset$. When we say that a set $S \subseteq \omega^{<\omega}$ is recursive, recursively enumerable, etc., what we mean is that the set $\{c(\alpha) : \alpha \in S\}$ is recursive, recursively enumerable, etc. Define a *tree* T to be a nonempty subset of $\omega^{<\omega}$ such that T is closed under initial segments. Call a function $f : \omega \to \omega$ an infinite *path* through T provided that for all n, $\langle f(0), \ldots, f(n) \rangle \in T$. Let $[T]$ be the set of all infinite paths through T. Call a set A of functions a Π^0_1-class if there exists a recursive predicate R such that $A = \{f : \omega \to \omega : \forall n (R(n, [f(0), \ldots, f(n)])\}$. Call a Π^0_1-class A *recursively bounded* if there exists 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^0_1-class, then $A = [T]$ for some recursive tree $T \subseteq \omega^{<\omega}$. Say that a tree $T \subseteq \omega^{<\omega}$ is *highly recursive* if T is a recursive finitely branching tree and also there is a recursive procedure which, applied to $\alpha = \langle \alpha_1, \ldots, \alpha_n \rangle$ in T, produces a canonical index of the set of immediate successors of α in T. Then if A is a recursively bounded Π^0_1-class, it is easy to show that $A = [T]$ for some highly recursive tree $T \subseteq \omega^{<\omega}$, see [JS72b]. For any set $A \subseteq \omega$, let $A' = \{e : \{e\}^A(e) \text{ is defined}\}$ be the jump of A, let $\mathbf{0'}$ denote the jump of the empty set $\emptyset$. We write $A \leq_T B$ if A is Turing reducible to B and $A \equiv_T B$ if $A \leq_T B$ and $B \leq_T A$.

We say that there is an effective, one-to-one degree preserving correspondence between the set of stable models $Stab(P)$ of a recursive program 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 $\{e_2\}^M = f)$.
where $\{e\}^B$ denotes the function computed by the e^{th} oracle machine with oracle B. Also, write $\{e\}^B = A$ for a set A if $\{e\}^B$ is a characteristic function of A. For any function $f : \omega \to \omega$, $gr(f) = \{[x, f(x)] : x \in \omega\}$. 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 infinite paths through T via an algorithm with index e_2. Condition (iii) asserts that if $\{e_1\}^{gr(f)} = M_f$, then f is Turing equivalent to M_f. In the sequel we

shall not explicitly construct the indices e_1 and e_2, but it will be clear that such indices can be constructed in each case.

4.2 Complexity of the Forward Chaining Construction.

In this section we discuss complexity issues for sets of the form $D^{\prec}$, where P is a recursive program and $\prec$ is either some ordering of type ω or some finite ordering. First of all, recall that every stable model of P can be obtained as $D^{\prec}$ for a suitably chosen ordering $\prec$. This means that, since the stable models can be very complex, even if there is only one stable model, we cannot obtain results on complexity of $D^{\prec}$ without restricting the class of orderings. Our restriction is related to the fact that in any attempt to implement even a partial construction of $D^{\prec}$, we cannot go beyond ω. Moreover, ω (and finite ordinals) have the following property:

Lemma 9. *Let P be a program and let $\prec$ be a well-ordering of $nmon(P)$ of order type $\leq \omega$. Then the closure ordinal of the construction of the family $\langle D^{\prec}_{\xi}\rangle$ is at most ω.*

It is easy to see that the property indicated in Lemma 9 does not hold for ordinals greater than ω.
We shall restrict our attention now to the case when P is recursive and $\prec$ is a recursive well-ordering of type ω.

Proposition 10. *Let P be a recursive general program. Let $\prec$ be a recursive well-ordering of $nmon(P)$ of order type $\leq \omega$. Finally, let $D^{\prec}, R^{\prec}, I^{\prec}$, and $A^{\prec}$ be sets of atoms and of clauses defined in Definition 2. Then: $D^{\prec}$ is r.e. in $\mathbf{0}'$, $R^{\prec}$ is r.e. in $\mathbf{0}'$, $I^{\prec}$ is recursive in $\mathbf{0}''$, and $A^{\prec}$ is recursive in $\mathbf{0}''$.*

Corollary 11. *If P is a recursive program such that $nmon(P)$ is finite, then for any ordering $\prec$ of $nmon(P)$, $D^{\prec}$ is r.e., $R^{\prec}$ is finite, and $I^{\prec}$ is finite and $A^{\prec}$ is recursive.* □

Now let us look at the case of finite P. In our complexity considerations, every atom a will have the cost $||a||$. Next, for a clause $r = c \leftarrow a_1, \ldots, a_n, \neg b_1, \ldots, \neg b_m$ we define $||r|| = (\sum_{i\leq n} ||a_i||) + (\sum_{i\leq m} ||b_j||) + ||c||$. Finally, for a set Q of clauses we define

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

Theorem 12. *Suppose P is a finite general program and $\prec$ is some well-ordering of $nmon(P)$. Then $D^{\prec}, R^{\prec}, A^{\prec}$, and $I^{\prec}$ can be computed in time*

$$O(||mon(P)||\,||nmon(P)|| + ||nmon(P)||^2).$$

5 FC-Normal Programs

In this section we shall define FC-normal programs and state the basic results about such programs proved in [MNR93b]. We shall see that FC-normal programs have the property that the Forward Chaining construction always produces a stable model. In fact for FC-normal programs, one can drop the consistency check in the Forward Chaining construction and it will still always produce a stable model.

Definition 13. Let P be a program. We say that a subset $Con \subseteq \mathcal{P}(H_P)$ (where $\mathcal{P}(H_P)$ is the power set of H_P) is a *consistency property* over P if:
(1) $\emptyset \in Con$, (2) $\forall_{A,B \subseteq H_P}(A \subseteq B \quad \& \quad Con(B) \Rightarrow Con(A))$,
(3) $\forall A_{\subseteq H_P}(Con(A) \Rightarrow Con(cl_{mon}(A)))$, and
(4) whenever $\mathcal{A} \subseteq Con$ has the property that $A, B \in \mathcal{A} \rightarrow \exists_{C \in \mathcal{A}}(A \subseteq C \wedge B \subseteq C)$, then $Con(\bigcup \mathcal{A})$.

We note that conditions (1),(2), and (4) are Scott's conditions for information systems. Condition (3) connects "consistent" sets to the Horn part of the program; if A is consistent then adding elements derivable from A via Horn clauses preserves "consistency".

Definition 14. Let P be a program and let Con be a consistency property over P.

1. A clause $C = c \leftarrow a_1, \ldots, a_n, \neg b_1, \ldots, \neg b_k \in nmon(P)$ is *FC-normal* (with respect to Con) if $Con(V \cup \{c\})$ and not $Con(V \cup \{c, b_i\})$ for all $i \leq k$ whenever $V \subseteq H_P$ is such that $Con(V)$, $cl_{mon}(V) = V$, $a_1, \ldots, a_n \in V$, and $c, b_1, \ldots, b_k \notin V$.

2. P is a *FC-normal* (with respect to Con) program if all $r \in nmon(P)$ are FC-normal with respect to Con.

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

FC-normal programs have all the desirable properties that are possessed by normal default theories as defined by Reiter in [Rei80]. In fact, it is shown in [MNR93b] that when one translates FC-normal programs back into the language of default logics than one obtains a class of default theories called *extended FC-normal* default theories which properly contains all normal default theories. We next shall state the basic results about FC-normal programs from [MNR93b].

Theorem 15. *Let P be a FC-normal program then there exists a stable model of P.*

Theorem 16. *Let P be a FC-normal program with respect to consistency property Con and let I be a subset of H_P such that $I \in Con$. Then there exists a stable model M of P such that $I \subseteq M$.*

In fact all stable models of FC-normal programs can be constructed via a slightly simplified version of the Forward Chaining construction which we shall call the Normal Forward Chaining construction. To this end, fix some well-ordering $\prec$ of $nmon(P)$. That is, the well-ordering $\prec$ determines some listing of the clauses of $nmon(P)$,$\{r_\alpha : \alpha \in \gamma\}$ where γ is some ordinal. Let Θ_γ be the least cardinal such that $\gamma \leq \Theta_\gamma$. In what follows, we shall assume that the ordering among ordinals is given by $\in$. Our normal Forward Chaining construction will define an increasing sequence of sets $\{M_\alpha^\prec\}_{\alpha\in\Theta_\gamma}$. We will then define $M^\prec = \cup_{\alpha\in\Theta_\gamma} M_\alpha^\prec$. In [MNR93b] it is shown that $M^\prec$ is always an stable model of P.

The Normal Forward Chaining construction of $M^\prec$.

Case 0. Let $M_0^\prec = cl_{mon}(\emptyset)$.

Case 1. $\alpha = \eta + 1$ is a successor ordinal.
Given $M_\eta^\prec$, let $\ell(\alpha)$ be the least $\lambda \in \gamma$ such that

$$r_\lambda = s \leftarrow a_1, \ldots, a_p, \neg b_1, \ldots, \neg b_k$$

where $a_1, \ldots, a_p \in M_\eta^\prec$ and $b_1, \ldots, b_k$, $s \notin M_\eta^\prec$. If there is no such $\ell(\alpha)$, then let $M_{\eta+1}^\prec = M_\alpha^\prec = M_\eta^\prec$. Otherwise, let

$$M_{\eta+1}^\prec = M_\alpha^\prec = cl_{mon}(M_\eta^\prec \cup \{cln(r_{\ell(\alpha)})\}).$$

Case 2. α is a limit ordinal. Then let $M_\alpha^\prec = \bigcup_{\beta\in\alpha} M_\beta^\prec$.

This given, we have the following.

Corollary 17. *If P is a FC-normal program and $\prec$ is* any *well-ordering of $nmon(p)$, then*

1. *$M^\prec$ is a stable model of P.*
2. *(Completeness of the construction). Every stable model of P is of the form $M^\prec$ for a suitably chosen ordering $\prec$ of $nmon(P)$.*

It is quite straightforward to prove by induction that if P is FC-normal with respect to consistency property Con, then $M_\alpha^\prec \in Con$ for all α and hence $M^\prec \in Con$. Thus the following is an immediate consequence of Theorem 17(2).

Corollary 18. *Let P be a FC-normal program with respect to consistency property Con, then every stable model of P is in Con.*

We should also point out that if we restrict ourselves to countable programs P, i.e. if H_P is countable, then we can restrict ourselves to orderings of order type ω where ω is the order type of the natural numbers. That is, suppose we fix some well-ordering $\prec$ of $nmon(P)$ of order type ω. Thus, the well-ordering $\prec$ determines some listing of the clauses of $nmon(P)$,$\{r_n : n \in \omega\}$. Our normal Forward Chaining construction can be presented in an even more straightforward manner in this case. Our construction again will define an increasing sequence of sets $\{M_n^\prec\}_{n\in\omega}$ in stages. This given, we will then define $M^\prec = \cup_{n\in\omega} M_n^\prec$. By the *Countable Normal Forward Chaining construction of $M^\prec$* we mean Normal Forward Chaining Construction restricted to orderings of type ω.

Theorem 19. *If P is a countable FC-normal program, then:*
1. $M^{\prec}$ is a stable model of P if $M^{\prec}$ is constructed via the Countable Normal Forward Chaining algorithm with respect to $\prec$, where $\prec$ is any *well-ordering of $nmon(P)$ of order type ω.*
2. Every stable model of P is of the form $M^{\prec}$ for a suitably chosen well-ordering $\prec$ of $nmon(P)$ of order type ω where $P^{\prec}$ is constructed via the Countable Normal Forward Chaining algorithm.

FC-normal programs also possess what Reiter terms the "semi-monotonicity" property.

Theorem 20. *Let P_1 and P_2 be two FC-normal program such that $P_1 \subseteq P_2$ but $mon(P_1) = mon(P_2)$ (that is, P_1, P_2 have the same Horn part). Assume, in addition, that both are FC-normal with respect to the same consistency property. Then for every stable model M_1 of P_1, there is a stable model M_2 of P_2 such that*

1. *$M_1 \subseteq M_2$ and*
2. *$NG(M_1, P_1) \subseteq NG(M_2, P_2)$.*

FC-normal programs also satisfy the *orthogonality of stable models* property with respect to their consistency property.

Theorem 21. *Let P be a FC-normal program with respect to a consistency property Con. Then if M_1 and M_2 are two distinct stable models of P, $M_1 \cup M_2 \notin Con$.*

We end this section with three more theorems which are analogues of results that hold for normal default theories.

Theorem 22. *Let P be a FC-normal program with respect to a consistency property Con. Suppose that $cl_{mon}\{cln(r) : r \in nmon(P)\}$ is in Con. Then P has a unique stable model.*

Theorem 23. *Suppose P is a FC-normal program and that $D \subseteq nmon(P)$. Suppose further that M_1' and M_2' are distinct stable models of $D \cup mon(P))$. Then P has distinct stable models M_1 and M_2 such that $M_1' \subseteq M_1$ and $M_2' \subseteq M_2$.*

References

[ABW87] K. Apt, H.A. Blair, and A. Walker. Towards a theory of declarative knowledge. *Foundations of Deductive Databases and Logic Programming*, pages 89–142, 1987.

[Apt90] K. Apt. Logic programming. *Handbook of Theoretical Computer Science*, pages 493–574, 1990.

[GL88] M. Gelfond and V. Lifschitz. The stable semantics for logic programs. *Proceedings of the 5th International Symposium on Logic Programming*, pages 1070–1080, 1988.

[GS92] J. Grant and V.S. Subrahmanian. Reasoning about inconsistent knowledge bases. *IEEE Trans. on Knowledge and Data Engineering*, to appear.

[JS72b] C.G. Jockusch and R.I. Soare. π_1^0 classes and degrees of theories. *Transactions of American Mathematical Society*, 173:33–56, 1972.

[KL89] M. Kifer and E. Lozinskii. RI: A logic for reasoning about inconsistency. TARK IV, pages 253-262, 1989.

[KN93a] W. Kohn and A. Nerode. Models for Hybrid Systems: Automata, Topologies, Controllability, Observability. In: *Hybrid Systems*, R.L. Grossman, A. Nerode, A.P. Ravn, H. Rischel, eds. Springer LN in CS 736, pages 317-356, 1993.

[MNR90] W. Marek, A. Nerode, and J.B. Remmel. Nonmonotonic rule systems I. *Annals of Mathematics and Artificial Intelligence*, 1:241–273, 1990.

[MNR92c] W. Marek, A. Nerode, and J.B. Remmel. Nonmonotonic rule systems II. *Annals of Mathematics and Artificial Intelligence*, 5:229–263, 1992.

[MNR92a] W. Marek, A. Nerode, and J. B. Remmel. The stable models of predicate logic programs. *Proceedings of International Joint Conference and Symposium on Logic Programming*, pages 446–460, Boston, MA, 1992. MIT Press.

[MNR95] W. Marek, A. Nerode, and J. B. Remmel. Complexity of Normal Default Logic and Related Modes of Nonmonotonic Reasoning, Proceedings of 10th Annual IEEE Symposium on Logic in Computer Science, pp. 178-187, 1995.

[MNR93b] W. Marek, A. Nerode, and J. B. Remmel. Context for Belief Revision: FC-Normal Nonmonotonic Rule Systems, Annals of Pure and Applied Logic 67(1994) pp. 269-324.

[MT91] W. Marek and M. Truszczyński. Autoepistemic logic. *Journal of the ACM*, 38:588 – 619, 1991.

[MT93] W. Marek and M. Truszczyński. *Nonmonotonic Logic – Context-dependent reasonings* 1993, Springer Verlag.

[Prz87] T. Przymusinski, On the declarative semantics of stratified deductive databases and logic programs, *Foundations of Deductive Databases and Logic Programming*, pages 193–216, 1987.

[RDB89] M. Reinfrank, O. Dressler, and G. Brewka. On the relation between truth maintenance and non-monotonic logics. *Proceedings of IJCAI-89*, pages 1206–1212.

[Rei80] R. Reiter. A logic for default reasoning. *Artificial Intelligence*, 13:81–132, 1980.

[VGRS91] 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(1991).

This article was processed using the LaTeX macro package with LLNCS style