Stable Models and Non-Determinism in Logic Programs with Negation

Domenico Saccà†
Dipartimento di Sistemi,
Università della Calabria, Rende, Italy

Carlo Zaniolo
MCC
Austin, Texas, USA

ABSTRACT

Previous researchers have proposed generalizations of Horn clause logic to support negation and non-determinism as two separate extensions. In this paper, we show that the stable model semantics for logic programs provides a unified basis for the treatment of both concepts. First, we introduce the concepts of partial models, stable models, strongly founded models and deterministic models and other interesting classes of partial models and study their relationships. We show that the maximal deterministic model of a program is a subset of the intersection of all its stable models and that the well-founded model of a program is a subset of its maximal deterministic model. Then, we show that the use of stable models subsumes the use of the non-deterministic *choice* construct in LDL and provides an alternative definition of the semantics of this construct. Finally, we provide a constructive definition for stable models with the introduction of a procedure, called *backtracking fixpoint,* that non-deterministically constructs a total stable model, if such a model exists.

1. Introduction

The problem of negated goals in rules represents a fast progressing area of research in deductive databases. Thus, the concept of *stratified programs* that was introduced only three years ago [ABW, CH, N, V1] is now regarded as a standard notion, efficiently supported in systems such as NAIL! [U] and LDL [NT]. Much of the current research focuses on going beyond the limitations of stratified programs [KP]. The important concepts of *locally stratified* programs and perfect models were proposed for this purpose [P1, P2, P3]. More recent work aims at going beyond local stratification, in order to express situations such as the following example, which describes a game where one wins if the opponent has no moves:

$$wins(X) \leftarrow move(X,Y), \neg wins(Y).$$

Suppose that there is only one *move* fact, as follows:

$$move(a,b).$$

Then this program is not locally stratified; however, it has a meaningful minimal model (the winner is obviously a) and the program does not seem to be ambiguous or faulty. *Well-founded models* and *stable models* represent two important proposals for going beyond local stratification [GL, P4, PP, V2, VRS]. Recent research focuses on constructive characterizations for well-founded models [P4, R, V2] and on the notions of *partial models*, [P4]. The definition of partial models used in [P4, P5] is based on 3-valued logic: the Herbrand base of each program is partitioned into three sets, respectively, containing ground atoms that are known to be true, false, or are otherwise undefined. Total models correspond to the case where the set of undefined ground terms is empty.

An important difference between stable model semantics and well-founded model semantics is that, while a program has always a unique well-founded model (either total or partial), it can have several alternative stable models. In this paper, we show that this endows logic languages with the power to express don't-care non-determinism in a purely declarative framework. We also clarify the relationships between different semantics for negation, by proposing a more general definition of partial models, establishing a hierarchy between various models, and showing that well-founded models are contained in the deterministic

† This author's work was done in part under contract by MCC and was also supported by the Italian National Research Council as part of the project "Sistemi Informatici e Calcolo Parallelo".

Permission to copy without fee all or part of this matertial is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that the copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

© 1990 ACM 089791-352-3/90/0004/0205 $1.50

intersection of all stable models.

Let us illustrate these points with an example. We have a database of students taking courses, as follows:

takes(andy, engl).
takes(ann, math).
takes(mark, engl).
takes(mark, math).

Say that we want to find the courses taught and an arbitrary student for each course. Then we can write the following rule:

$$a\_st(St,Crs) \leftarrow takes(St,Crs),\ \neg dif\_st(St,Crs).$$
$$dif\_st(St,Crs) \leftarrow St1 \neq St,\quad takes(St1,Crs),\ a\_st(St1,Crs).$$

The intuitive meaning of this program is that a given student taking a course should be included in the *a_st* answer, unless a different student also taking the same course is already included in the answer.

As we will discuss in more details later, under the stable model semantics [GL], this program has four stable models, each containing one of the following four pairs:

Table 1. Four alternative solutions

<a_st(andy, engl),	*a_st(ann, math)>*
<a_st(mark, engl),	*a_st(mark, math)>*
<a_st(ann, math),	*a_st(mark, engl)>*
<a_st(andy, engl),	*a_st(mark, math)>*

Each of these four stable models satisfies the intended semantics: *Find an arbitrary student for each course.*

As this example illustrates, there is a real need for don't care non-determinism in logic programming applications. To satisfy this strong need, special constructs were introduced, such as the declarative constructs of *choice* in LDL [KN,NT], the *witness* operator in [AV], and the procedural *cut* construct in Prolog (although the cut serves many other purposes as well). However, no special construct is needed once a stable model semantics is used for logic programs, since the (multiple) stable models semantics subsumes the LDL choice construct, which, in turn, provides a declarative substitute to Prolog's cut [KN]. The well-founded model semantics is not suitable for the example application, inasmuch as it produces a partial model that blurs the meaning by assigning an "undefined" classification to all the *a_st* facts listed above.

The contribution of this paper is three-fold:

(1) an in-depth study of the properties of partial models, stable models and other interesting classes of models and their relationships;

(2) the identification of the power of stable models to express non-determinism, with a proof that they subsume the semantics of programs with LDL's choice construct;

(3) the invention of a constructive semantics for stable models via a generalized fixpoint procedure, called backtracking fixpoint.

The paper is organized as follows. In Section 2, we introduce partial models and stable models and, in Section 3 and Section 4, we define strongly-founded models, deterministic models and elucidate the relationships between stable models and well-founded models. In Section 5, we consider a generalization of locally stratified programs and for the new class of programs (weakly stratified) for which a stable model always exists. In Section 7 we show the relationship of stable model semantics with LDL's choice construct. In Section 6, we present the backtracking fixpoint procedure for constructing stable models. Because of space limitations, some of the proofs will be omitted; they can be found in [SZ].

2. Partial Models, Founded Models and Stable Models

Let us start by defining our language (Horn clauses plus negated goals in rules, as Prolog) and basic concepts and notation [L,U].

A *term* is a variable, a constant, or a complex term of the form $f(t_1, \ldots, t_n)$, where $t_1, \ldots, t_n$ are terms. An *atom* is a formula of the language that is of the form $p(\mathbf{t})$ where p is a predicate symbol of a finite arity (say n) and $\mathbf{t}$ is a sequence of terms of length n (*arguments*). A *literal* is either an atom (*positive literal*) or its negation (*negative literal*). An atom A, and its negation, i.e., the literal $\neg A$, are said to be the *complement* of each other. In general, if B is a literal, then $\neg B$ denotes the complement of B. The *absolute value* of a literal B, denoted $abs(B)$ is defined as $abs(B)=B$ if B is positive, and $abs(B)=\neg B$ if B is negative.

A *rule* is a formula of the language of the form

$$Q \leftarrow Q_1, \ldots, Q_m.$$

where Q is a atom (*head* of the rule) and $Q_1, \ldots, Q_m$ are literals (*goals* of the rule). A term, atom, literal or rule is *ground* if it is free of variables. A ground rule with no goals is a *fact*. A *logic program* is a set of rules. A rule without negative goals is called positive (a Horn clause); a program is called *positive* when all its rules are positive.

Given a logic program P, the *Herbrand universe* for P, denoted H_P, is the set of all possible ground terms recursively constructed by using constants and function symbols occurring in P. The *Herbrand Base* of P, denoted B_P, is the set of all possible ground atoms whose predicate symbols occur in P and whose arguments are elements of the H_P. A *ground instance* of a rule r in P is a rule obtained from r by replacing every variable X in r by $\phi(X)$, where ϕ is a mapping from all variables occurring in r to terms in the Herbrand universe. The set of all ground instances of r are denoted by *ground* (r); accordingly, *ground* (P) denotes $\cup_{r \in P}$ *ground* (r).

Let X be a set of ground literals; then $\neg X$ denotes the set $\{\neg A \mid A \in X\}$, X^+ (resp., X^-) denotes the set of all positive (resp., negative) literals in X. Finally, $\overline{X}$ denotes all elements of the Herbrand Base which do not occur in X, i.e., $\overline{X} = \{A \mid A \in B_P$ and neither A nor $\neg A$ is in $X\}$.

Definition 1. Let P be a logic program.

(a) Given a subset I of $B_P \cup \neg B_P$, I is an *interpretation* of P if it is *consistent*, i.e., there are no two complementary elements in it. Moreover, if $I^+ \cup \neg I^- = B_P$, the interpretation I is called *total;* and it is called *partial* otherwise.

(b) A *total model*, M of P is a total interpretation of P that makes each ground instance of each rule in P *true* (where a ground literal is *true* if and only if it belongs to M).

(c) A *minimal model* M of P is a total model for which there exists no other total model N such that N^+ is a proper subset of M^+. □

It is well-known that a positive program has a unique minimal model which represents its natural meaning. The set of positive literals in the minimal model can be determined using a fixpoint computation. This computation is based on the *immediate consequence transformation* T_P: $2^{B_P \cup \neg B_P} \rightarrow 2^{B_P}$ that is defined as $T_P(X) = \{A \mid A \leftarrow A_1, \ldots, A_n$ is in *ground*(P) and $A_i \in X$ for each $1 \le i \le n\}$. The transformation T_P is monotone in the complete lattice of set subsumption, and, then, the least fixpoint of T_P exists [T] and is denoted by $T_P^\infty(\emptyset)$. If P is a positive program then $M^+ = T_P^\infty(\emptyset)$, where M is the minimal model of P.

In order to analyze the meaning of programs with negation, we now introduce the notion of partial model.

Definition 2. A partial interpretation M of a program P is *a partial model* for P if for each $\neg A$ in M^-, every rule in *ground*(P) with head A contains at least one goal, say B, such that $\neg B$ is in M. □

The predicates which do not occur in M (i.e., those in $\overline{M}$) are not known to be true or false and, then, they can be thought of as "undefined facts". Our definition of partial model, that is different from others proposed in the past [P4], only guarantees that assuming a fact false cannot be later contradicted by changes in the value of undefined facts. The following intuitive property holds.

PROPOSITION 1. *Every partial model is a subset of some total model.*

PROOF. Let P be a program and M be a partial model of P. Consider the set $N = M \cup \overline{M}$. In order to prove the proposition it is sufficient to show that N is a total model. By construction, N is consistent and every element of B_P occurs in N; therefore N is a total interpretation of P. So we only need to prove that every rule in *ground*(P) is made true by N. Let A be the head of an arbitrary rule r in *ground*(P); since N is total, either A is in N^+ or $\neg A$ is in N^-. Let us only consider the case that $\neg A$ is in N^- as the proof is trivial in the former case. By construction, $\neg A$ is also in M^-; therefore, by definition of partial model, there exists a goal r, say B, such that $\neg B$ is in M. But M is a subset of N by construction; so $\neg B$ is also in N. Thus the rule r is made true by N. □

From the above proposition, it follows that the definition of partial model is actually a generalization of the definition of total model.

COROLLARY 1. *For total interpretations, Definitions 1b and 2 are equivalent.* □

Example 1. Consider the following program:

$$p(a) \leftarrow \neg p(b).$$

$$p(c) \leftarrow p(b).$$

$$p(d) \leftarrow p(c).$$

$\{p(a), \neg p(b), \neg p(c)\}$ is a partial model and is a subset of the total minimal model $\{p(a), \neg p(b), \neg p(c), \neg p(d)\}$. □

We now present the definition of stability for total models as introduced in [GL]. First of all, given a program P and a total model M for P, we define the *positive instantiation* of P w.r.t. M, denoted P_M, as follows: P_M is the positive program obtained from *ground*(P) by deleting (a) each rule that has a negative goal $\neg A$ with A in M, and (b) all negative goals from the remaining rules.

Definition 3. A total model M is *stable* if $T_{P_M}^\infty(\emptyset) = M^+$. □

Example 2. Consider the following program having 0-arity predicate symbols:

$$u \leftarrow \neg v.$$

$$v \leftarrow \neg u.$$

There are three total models: $M_1 = \{u, \neg v\}$, $M_2 = \{v, \neg u\}$, $M_3 = \{u, v\}$. Only M_1 and M_2 are stable. □

We next extend the notion of stability also to partial models. To this end, we first extend the definition of positive instantiation to the case of partial models. Given a program P and a partial model M for P, the *positive instantiation* of P w.r.t. M, denoted P_M, is the positive program obtained from *ground*(P) by deleting (a) each rule that has a negative goal $\neg A$ with A in M, (b) each rule where an undefined element occurs (i.e., the head or the absolute value of one of its goals is in $\overline{M}$) and (c) all negative goals from the remaining rules.

Definition 4. Let P be a logic program.

(a) A partial model M for P is *founded* if $T_{P_M}^{\infty}(\emptyset) = M^+$.

(b) A partial model M for P is *stable* it is founded and it is not a proper subset of any other founded model. □

Since no total model can be a subset of another model, we have the following property:

FACT 1. *For total models Definitions 2 and 4b are equivalent.* □

As proven next, in general, given a partial model M, $T_{P_M}^{\infty}(\emptyset)$ is a subset of M^+.

PROPOSITION 2. *Let M be a partial model of a logic program P. Then $T_{P_M}^{\infty}(\emptyset) \subseteq M^+$.*

PROOF. Consider the program P_M and the set $N = N^+ \cup N^-$, where $N^+ = M^+$ and N^- contains all negative literals in M^- whose complement is in the Herbrand Base of P_M. By construction of P_M and N, N is a total interpretation of P_M. We now show that N is actually a total model of P_M. In fact, consider an arbitrary rule r of P_M. Say that the head of r is A. If A is in N^+ then r is obviously made true by N. Suppose now that $\neg A$ is in N^-. Let $\hat{r}$ be the original rule in *ground*(P) from which r has been derived. By definition of partial model, $\hat{r}$ contains at least one goal, say B, such that $\neg B$ is in M. We have that B is positive because otherwise r would have not been derived (see part (a) of the definition of P_M). Hence, $\neg B$ is in M^- and then, in N^- by construction of N^-. Therefore, one of the goals of r is false and, then, r is made true by N. So N is a total model for P_M. Let L be the (unique) minimal model of the positive program P_M. Obviously $L^+ \subseteq N^+$. But $N^+ = M^+$ and $L^+ = T_{P_M}^{\infty}(\emptyset)$; so $T_{P_M}^{\infty}(\emptyset) \subseteq M^+$. □

The elements in $M^+ - T_{P_M}^{\infty}(\emptyset)$ can be thought of as assumptions, in the sense that they cannot be actually inferred in P_M. We next formalize this intuition about assumptions. To this end, we first present the notion of unfounded set as given in [VRS] and a related notion (assumption set).

Definition 5. Let I be a partial interpretation of a program P and X be a non-empty subset of B_P.

(a) X is an *unfounded set w.r.t. I* if for each A in X, every rule with head A in *ground*(P) has some goal g such that g is in X or $\neg g$ is in I.

(b) X is an *assumption set w.r.t. I* if for each A in X, every rule with head A in *ground*(P) has some goal which is not in $I - X$. □

Obviously every unfounded set is an assumption set but the converse is not true. For total interpretations, the two concepts coincide.

THEOREM 1. *A partial model M is founded if and only if no subset of M^+ is an assumption set w.r.t. M.*

PROOF. Let P be a logic program and M be a partial model for it.

(If-part). Suppose that no subset of M^+ is an assumption set w.r.t. M; we have to prove that M is founded. We proceed by contradiction and we assume that M is not founded, thus $T_{P_M}^{\infty}(\emptyset) \neq M^+$. But $T_{P_M}^{\infty}(\emptyset) \subseteq M^+$ by Proposition 2; so $T_{P_M}^{\infty}(\emptyset) \subset M^+$. Let $X = M^+ - T_{P_M}^{\infty}(\emptyset)$. By assumption, X is not empty. Let A be any element in X. There exists no rule r in *ground*(P) with head A such that each of its goal is in $M - X$ because otherwise, since the corresponding rule in P_M has all goals in $M^+ - X$ and $M^+ - X = T_{P_M}^{\infty}(\emptyset)$, A would be in $T_{P_M}^{\infty}(\emptyset)$ by definition of least fixpoint. Hence, X is an assumption set w.r.t. M (contradiction). Therefore $T_{P_M}^{\infty}(\emptyset) = M^+$ and M is founded.

(Only-If-part). Suppose now that M is founded; we prove that no subset of M^+ is an assumption set w.r.t. M. Again we proceed by contradiction and we assume that $X \subseteq M^+$ is an assumption set w.r.t. M. Let $N = N^+ \cup N^-$ be the (unique) minimal model of P_M. Obviously, $N^+ = M^+ = T_{P_M}^{\infty}(\emptyset)$. Consider now the set $L = L^+ \cup L^-$, where $L^+ = N^+ - X$ and $L^- = \neg X \cup N^-$. By construction, L is a total interpretation of P_M. We now show that L is actually a total model of P_M. In fact, consider an arbitrary rule r of P_M. Say that the

head of r is A. If A is in L^+ then r is obviously made true by L. Suppose now that $\neg A$ is in L^-. There are two possible cases:

(a) $\neg A$ is also in N^-. Hence, since r is made true by N, there exists a goal of r, say B, such that $\neg B$ is in N^-. By construction of L^-, $\neg B$ is also in L^- and, then, r is also made true by L.

(b) $\neg A$ is in $\neg X$. Let $\hat{r}$ be the original rule in *ground*(P) from which r has been derived. By definition of assumption set, there exists a goal of r, say B, such that B is not in $M - X$. B is in M^+ because otherwise r would have not been derived (see part (a) of the definition of P_M). Hence, B is in X and is a goal of r as well. Therefore, since one of the goals of r is false w.r.t. L, r is made true by L.

It follows that L is a model of P_M and we get a contradiction since $L^+ \subset N^+$ by construction and N is the minimal model of P_M. Therefore any subset X of M^+ is not an assumption set w.r.t. M. □

COROLLARY 2. *A total model M is stable if and only if no subset of M^+ is an assumption set w.r.t. M.* □

Example 3. Consider the program of Example 2:

$u \leftarrow \neg v.$

$v \leftarrow \neg u.$

and the stable model $M_1 = \{u, \neg v\}$. We have that $\{u\}$ is not an assumption set w.r.t. M_1 since the goal of the first rule is in $M_1 - \{u\}$. Note that $\{u,v\}$ is an assumption set (but not an unfounded set) w.r.t. the empty set. Consider now the following program:

$a \leftarrow \neg a.$

$b \leftarrow \neg c, d.$

$c \leftarrow d.$

$d \leftarrow b.$

We have that the $\{\neg b, \neg c, \neg d\}$ is the unique stable model. Note that $\{A\}$ is an assumption set w.r.t. it. Moreover, $\{b, c, d\}$ is an unfounded set w.r.t. the empty set. □

Note that every program has at least one (possibly partial) stable model since the empty set is always a founded model.

FACT 2. *There exists a stable model for every program.* □

3. 3-Valued Models, Strongly-Founded Models and Well-Founded Models

Following [P5], we define a 3-valed logic with T(*rue*), F(*false*), and U(*undefined*), ordered as follows $F < U < T$. Given a program P, an interpretation I, and a ground literal A, *value*(A) is T if A is in I, F if $\neg A$ is in I and U if the *abs* of A is in $\overline{I}$. Moreover, the value of a conjunction C of ground literals is the minimal value of the literals in the conjunction, i.e., *value*$(C) = \min_{A \text{ in } C}(\textit{value}(A))$. If C is empty the we assume that *value*$(C) = T$.

Definition 6. Let P be a program and I be an interpretation for it. Then I is a *3-valued model* for P if for each rule r in *ground*(P), *value*$(A) \geq$ *value*(C), where A is the head of r and C is the conjunction of the goals of r. □

We next show that 3-valued models are a subclass of partial models.

THEOREM 2. *Let P be a program and I be an interpretation for it. Then, I is a 3-valued model for P if and only if I is a partial model such that every subset of $\overline{I}$ is an assumption set w.r.t. I.* □

Example 4. Consider the following program:

$a.$

$b \leftarrow \neg c, c.$

$c \leftarrow b.$

$d \leftarrow e.$

Here we have that the total model $\{a, \neg b, \neg c, \neg d, \neg e\}$ is both stable and 3-valued. The partial model $\{a, c\}$ is 3-valued but not founded; the empty set is a founded partial model that is not 3-valued. □

We now turn our attention to a particular class of the models that are both 3-valued and founded. Such models correspond to the stable models as defined in [P5].

Definition 7. A partial model M is *strongly-founded* if M is 3-valued, founded and no subset of $\overline{M}$ is an unfounded set w.r.t. M. □

Example 5. Consider the program of Example 4. The partial model $\{a, \neg b, \neg c\}$ is founded and 3-valued but not strongly-founded since $\{d, e\}$ is an unfounded set w.r.t. M. □

As it will be shown next, strongly-founded models are fixpoints of the following monotonic transformation first defined in [VRS]. We first observe that, given a program P and an interpretation I for it, the union of all unfounded sets w.r.t. I (denoted by $U_P(I)$) is also an unfounded set w.r.t. I. Moreover, let $W_P(I) = T_P(I) \cup \neg U_P(I)$. Then we have the following theorem:

THEOREM 3. *Let P be a logic program.*

(a) *Every strongly-founded model for P is a fixpoint of W_P.*

(b) *If an interpretation M for P is a fixpoint of W_P, then M is a 3-valued model for P and no subset of $\overline{M}$ is an unfounded set w.r.t. M.* □

Example 6. Consider the following program:

$p \leftarrow \neg p.$

$a \leftarrow \neg p.$

$a \leftarrow b.$

$b \leftarrow a.$

The 3-valued model $\{a,b\}$ is a fixpoint of W_P but is not founded (thus, the converse of Part a of Theorem 3 does not hold). The 3-valued model $\{p\}$ is not a fixpoint of W_P although no subset of $\overline{M}$ is an unfounded set w.r.t. M (thus, the converse of Part b of Theorem 3 does not hold). □

Stable models are fixpoints of W_P since, as proven next, they are strongly-founded.

PROPOSITION 3. *Stable models are strongly-founded.* □

Since T_P, U_P and W_P are monotonic transformations in the complete lattice of set subsumption, W_P has a least fixpoint [T]. This defines the well-founded model of P [VRS].

Definition 8. Let P be a program. The *well-founded* model of P is the least fixpoint of W_P. □

PROPOSITION 4. *Let P be a program. The well-founded model for P is strongly-founded and is the intersection of all strongly-founded models for P.* □

Thus, by the above proposition, the well-founded model is the minimal strongly-founded model.

Example 7. Consider the following program:

$a \leftarrow \neg b.$

$b \leftarrow \neg a.$

$d \leftarrow \neg c, c.$

$c \leftarrow d.$

The well-founded model is $\{\neg c, \neg d\}$ and is the intersection of the two (total) stable models $\{a, \neg b, \neg c, \neg d\}$ and $\{b, \neg a, \neg c, \neg d\}$. □

4. Deterministic Models

In this section we show that stable models capture and express the notion of non-determinism in logic programs with negation. To elaborate this point, let us return to the example in the introduction.

Example 8. Consider the following program:

$takes(andy, engl).$
$takes(ann, math).$
$takes(mark, engl).$
$takes(mark, math).$
$a\_st(St, Crs) \leftarrow takes(St, Crs), \neg dif\_st(St, Crs).$
$dif\_st(St, Crs) \leftarrow$
$\quad St1 \neq St, takes(St1, Crs), a\_st(St1, Crs).$

Consider the following set:

$M^+ = \{takes(andy, engl), takes(ann, math), takes(mark, engl), takes(mark, math), a\_st(andy, engl), a\_st(ann, math), dif\_st(mark, engl), dif\_st(mark, math)\}.$

Let $M^- = \neg(B_P - M^+)$, where B_P is the Herbrand base of the program. Obviously $M = M^+ \cup M^-$ is a total model. It is easy to see that M is a stable model. But this is not the only stable model since a simple symmetry argument let us infer that there are four stable models each containing one of the four pairs in Table 1. Indeed, given the complete symmetry between these four models, the idea of one being preferable over another would be semantically unfounded. The symmetry of the situation instead suggests that we have here an instance of don't-care non-determinism —fully capturing the original intention "Find an arbitrary student for each course". Note that the well-founded model only contains the *takes* facts whereas all *a_st* predicates remain undefined. □

We can now establish a clear relationship between stable models and non-determinism.

Definition 9. Let P be a logic program.

(a) The intersection of all the stable models for P will be called the *deterministic set* for P.

(b) Every strongly-founded model which is contained in the deterministic set will be called a *deterministic model.*

(c) A *maximal* (resp., *minimal*) *deterministic model* for P is a deterministic model that is not a proper subset (resp., superset) of any other deterministic model. □

The next two examples show that the deterministic set is not necessarily a partial model and that a partial model contained in the deterministic set is not necessarily strongly-founded.

Example 9. Consider the following program P_1:

$u \leftarrow \neg q_1, \neg q_2.$

$q_1 \leftarrow a.$

$q_2 \leftarrow b.$

$a \leftarrow \neg b.$

$b \leftarrow \neg a.$

There are two stable models: $M_1 = \{b, q_2, \neg a, \neg q_1, \neg u\}$ and $M_2 = \{a, q_1, \neg b, \neg q_2, \neg u\}$ (note that both models are total). The deterministic set is $\{\neg u\}$ and obviously is not a partial model.

Consider now the following program P_2:

$a.$

$p \leftarrow \neg q, a.$

$q \leftarrow \neg p, a.$

$r \leftarrow p.$

$r \leftarrow q.$

Here there are two stable models $\{a, p, r, \neg q\}$ and $\{a, q, r, \neg p\}$. The deterministic set is $\{a, r\}$ and is a partial model in this case but not founded. □

We now show that the well-founded model is the unique minimal deterministic model.

PROPOSITION 5. *The well-founded model is the unique minimal deterministic model.*

PROOF. Let P be a logic program. Since a stable model is strongly-founded by Proposition 3, the well-founded model is a subset of every stable model by Proposition 4, thus it is contained in the deterministic set. Moreover, as the well-founded model is strongly-founded by Proposition 4, the well-founded model is a deterministic model. Finally, since it is the intersection of all strongly-founded models, none of its subsets is strongly-founded. It follows that the well-founded model is the unique minimal deterministic model. □

COROLLARY 3. *Every logic program has at least one deterministic model.* □

Note that for the program P_2 of Example 9 the well-founded model is $\{a\}$, which coincides with the maximal deterministic model. The next example shows that, in general, this is not the case.

Example 10. Consider the following program:

$a \leftarrow \neg b.$

$b \leftarrow \neg a.$

$a \leftarrow \neg c.$

$c \leftarrow \neg a, \neg b.$

There is only one stable model, namely, $M = \{a, \neg b, \neg c\}$ whereas the well-founded model is empty. Obviously M is the maximal deterministic model, i.e., the program is deterministic in a sense. The well-founded model is unable to realize that the fourth rule can never be fired, and thus cannot draw the necessary consequences.

Every program has a unique maximal deterministic model.

THEOREM 4. *The maximal deterministic model is unique.* □

We summarize the class/subclass relationships between the various models in the diagram of Figure 1.

$$\begin{tikzcd}
& \text{PARTIAL MODELS} \arrow[dl, dash] \arrow[dr, dash] & & \\
\text{3-VALUED MODELS} \arrow[dr, dash] & & \text{FOUNDED MODELS} \arrow[dl, dash] & \\
& \text{STRONGLY FOUNDED MODELS} \arrow[dl, dash] \arrow[dr, dash] & & \\
\text{STABLE MODELS} & & \text{DET. MODELS} \arrow[dl, dash] \arrow[dr, dash] & \\
& \text{MAX. DET. MODELS} & & \text{WELL-FOUNDED MODELS}
\end{tikzcd}$$

Fig. 1. *The class/subclass relationships between models.*

It is important to remember that the diagram of Figure 1 only represents a partial order. Thus the class of Strongly Founded Models is a subset of the class of 3-Valued Models and of Founded Models, but it is not equal to their intersection. Also, the class of well-founded model coincides with that of minimal deterministic models. Finally, the class relationship of Figure 1, should not be confused with the relationship between instances, which can instead be summarized as follows: the well-founded model of a program P is a subset of the maximal deterministic model for P, which, in turn, is a subset of each stable model for P.

The fact that the well-founded model can be a proper subset of the maximal deterministic model indicates that that well-founded models are not sufficient to fully capture the deterministic implications of a logic program. This supplies a first motivation for going beyond well-founded semantics. An even stronger motivation follows from the fact that stable models provide a formal ground for expressing 'non-deterministic' aspects of logic programming. This is the topic of the section following the next one.

5. Weakly Stratified Programs

A useful formalism to analyze semantics for negation is the *(ground) dependency graph.* Given a logic program P, the dependency graph of P, denoted G_P, is a directed graph whose nodes are all elements of B_P and whose arcs are defined as follows. Given two nodes A and B, there is an arc from A to B if there exists a rule in $ground(P)$ such that B is the head of the rule and A one of the goals; moreover, if A is negated then the arc is marked. We note that problems with the semantics of negation are generated by cycles with marked arcs.

A program P is *locally stratified* [P1, P2, P3] if (a) there are no cycles with marked arcs in G_P and (b) no node of G_P is the end node of a path having an infinite number of marked arcs. It is known that if a program is locally stratified then it has a total well-founded model [VRS] (and, then, this model is also stable). We now show that the larger class of programs for which only condition (a) holds have interesting properties. In fact, for such programs the existence of a total stable model (but not its uniqueness) is guaranteed. Note that in general such a total stable model is not well-founded model.

Definition 10. A program P is *weakly stratified* if there are no cycles with marked arcs in G_P. □

It turns out that every locally stratified program is also weakly stratified but the converse is not true. But for programs without function symbols, such as Datalog programs, the two notions coincide.

THEOREM 5. *If a program is weakly stratified then each of its stable models is total.* □

COROLLARY 4. *Every weakly stratified program has a total stable model.* □

We next show that, given a program and a founded model, the part of the program that is "relevant" for the model is weakly stratified.

Given a program P and a partial model M for P, the *enabled instantiation* of P w.r.t. M, denoted P_M^e, is the program obtained from $ground(P)$ by deleting each rule that has a goal not in M; in other words, P_M^e contains all "enabled" rules of $ground(P)$, i.e., those rules for which all goals are true.

PROPOSITION 6. *Let P be a program and M be a partial model for it. If M is founded then P_M^e is weakly stratified.* □

6. Choice Models

Having thus examined the theoretical aspects of non-determinism in programs with negation, let us next establish their relationship to the semantics of declarative constructs for defining non-determinism in deductive databases. For this purpose, we can limit our attention to the usual framework of total models. Thus from now on, we will say models to mean total models, and a model of P is represented only by its positive literals, i.e., the model M stands for $M \cup \neg(B_P - M)$. The notion of declarative semantics for non-determinism in logic-based languages has been studied in [KN], where the *choice* construct was introduced and it was shown that in terms of expressive power, it provides a declarative replacement for Prolog's cut. The choice construct is efficiently supported in the current LDL implementation, and it has proven critical in important applications [Z].

The meaning of a program with choice constructs is defined by its *choice models,* as discussed next.

Example 11. Consider the following program with choice.

$a\_st(St, Crs) \leftarrow takes(St, Crs),\ choice((Crs), (St)).$
$takes(andy, engl).$
$takes(ann, math).$
$takes(mark, engl).$
$takes(mark, math).$

The choice goal in the first rule specifies that the *a_st* predicate symbol must associate exactly one student to each course. Thus the above program has the following four "choice" models:

$$M_1=\{a\_st(andy, engl),\ a\_st(ann, math)\} \cup X,$$
$$M_2=\{a\_st(mark, engl),\ a\_st(mark, math)\} \cup X,$$
$$M_3=\{a\_st(mark, engl),\ a\_st(ann, math)\} \cup X,$$
$$M_4=\{a\_st(andy, engl),\ a\_st(mark, math)\} \cup X.$$

where X is the set of *takes* facts in Example 11. We will show later in this section, that the above sets correspond to the stable models of the program in Example 8. □

Let us now formalize the notion of choice model. A *choice predicate* is a predicate of the form *choice*((X),(Y)), where X and Y are disjunct lists of variables (note that X can be empty). A *choice rule* is a rule having one or more choice predicate as goals. Finally, a *choice program* is a positive program such that

(a) At least one of its rule is a choice rule,

(b) No choice predicate occurs in the head of any rule, and

(c) The head of each of its choice rules is not mutually recursive with some of the goals of the rule. (More about this stratification condition for choice will be said later.)

Let P be an choice program and suppose that the rules of P are numbered with distinct indices. The *positive version* of P, denoted by $PV(P)$, is the positive program obtained from P by removing all choice goals. Moreover, the *extended version* of P, denoted by $EV(P)$, is the positive program obtained from P by replacing each choice rule in P, say

$$r_i\colon\ A \leftarrow\ \mathbf{B}, \mathbf{C}.$$

where i is the index of the rule, $\mathbf{C}$ is the conjunction of all choice goals and $\mathbf{B}$ is the conjunction of all remaining goals, with the two following rules:

$$A \leftarrow\ \mathbf{B}, extChoice_i(\mathbf{Z}).$$

$$extChoice_i(\mathbf{Z}) \leftarrow\ \mathbf{B}.$$

where $\mathbf{Z}$ are all variables in the choice goals, listed in the order they occur in such goals.

Let I_1 and I_2 be two interpretations from two possibly different Herbrand bases of two, possibly different programs. Then we define I_1/I_2 as $\{A \mid A$ is in I_1 and the predicate symbol of A also occurs in $I_2\}$. It turns out that, when $I_1/I_2 = I_2$, then I_1 is identical to I_2 modulo additional literals whose predicate symbols are not in I_2.

PROPOSITION 7. *Let P be a choice program, M and N be the minimal models of $PV(P)$ and of $EV(P)$, respectively. Then $N/M = M$.* □

Note that the only predicates of $EV(P)$ which do not occur in $PV(P)$ are those with symbol $extChoice_i$. Consider any of such predicate, say $extChoice_i(\mathbf{Z})$ with arity $n = |\mathbf{Z}|$. This predicate correspond to an n-ary database relation R_i ([U]) whose attribute names are the variables in $\mathbf{Z}$ and whose tuples are $\{(\mathbf{z}) \mid extChoice_i(\mathbf{z})$ is in the minimal model of $EV(P)\}$. Given the rule r_i in P to which the new predicate is associated, we define the following set F of functional dependencies on R_i:

$$F = \{\mathbf{X} \rightarrow \mathbf{Y} \mid choice((\mathbf{X}),(\mathbf{Y})) \text{ is a goal of } r_i\}.$$

A *reduced version* of R_i is a maximal subset of R_i for which all the functional dependencies in F hold. Note that a reduced version of R_i is not necessarily unique and is empty if and only if R_i is empty.

We can now define a *reduced version* of P, denoted as $RV(P)$, as the program obtained from $EV(P)$ by replacing each rule of the form

$$extChoice_i(\mathbf{Z}) \leftarrow\ \mathbf{B}.$$

with the set of facts $\{extChoice_i(\mathbf{z}) \mid (\mathbf{z})$ is in $S_i\}$, where S_i denotes an arbitrarily chosen reduced version of R_i. Note that, as a reduced version of R_i is not necessarily unique, P may have several reduced versions.

Definition 11. Let P be an choice program. The minimal model of every reduced version of P is a *choice model* for P. □

Example 12. Consider the following choice program P:

$$\begin{aligned}&colored(G,C) \leftarrow color(C), glass(G),\\ &\qquad\qquad choice((C),(G)), choice((G),(C)).\\ &color(green).\\ &color(red).\\ &color(fuxia).\\ &glass(mine).\\ &glass(yours).\end{aligned}$$

$PV(P)$ is obtained from P by replacing the first rule with the following one:

$$colored(G,C) \leftarrow\ color(C), glass(G).$$

According to the minimal model of $PV(P)$, both *my* and *your* glass are colored with all three available colors. $EV(P)$ is obtained from P by replacing the first rule with the following two rules:

$$\begin{aligned}&colored(G,C) \leftarrow\ color(C), glass(G),\\ &\qquad\qquad extChoice(C,G).\\ &extChoice(C,G) \leftarrow\ color(C), glass(G).\end{aligned}$$

(Note that we do not need any index for there is only one choice rule in P.) The attribute names of the relation R are C and G and its tuples are $\{(a,b) \mid a$ is *green*, *red* or *fuxia* and b is *mine* or *yours*$\}$. The set of functional dependencies associated to R is $F = \{C \rightarrow G,\ G \rightarrow C\}$. A reduced version of R is $S = \{(mine,red),\ (yours,green)\}$. Therefore, the program, composed by the following rules:

$$\begin{aligned}&colored(G,C) \leftarrow\ color(C), glass(G),\\ &\qquad\qquad extChoice(C,G).\\ &extChoice(mine,red).\\ &extChoice(yours,green).\end{aligned}$$

and by all facts in P defining *color* and *glass* is a reduced version of P. According to this program, a choice model for P assign the color *red* to *my* glass and the color *green* to *your* glass. It is easy to see that each choice model assigns exactly one color to each glass in such a way that the two glasses do not share the same color. Therefore, the choice rule of P can be read as follows: "From all possible combinations of colors and glasses, choose a combination for which each glass is colored with at most one color and the same color is used for at most one glass". □

Thus choice models introduce the notion of non-deterministic choice in logic programming and, therefore, enlarge its expressive power. The very meaning of the choice construct can be summarized as follows. A number of constraints are imposed upon the database relation corresponding to the head predicate of the choice rule; then any maximal subset of this relation satisfying these constraints can be selected. A direct implementation of this definition is rather inefficient since it would require (i) computing the model for the positive version of the program and, then, (ii) checking for functional dependencies in various subsets and, (iii) computing the model of a reduced version of the program whose *size* (i.e., the total number of symbols in the program) is polynomially bounded in the size of the original program.

We will next propose an alternative definition of choice models in terms of stable models. This illustrates the ability of stable models to capture non-determinism and is also conducive to more efficient computation. In fact, we can map a program with choice into a program with negation whose size is linear in the size of the choice program. Moreover, as it will be shown in the next section, a choice model can be computed efficiently from the constructed program.

Given a choice program P, the *stable version* of P, denoted by $SV(P)$, is the program with negation obtained from P by the following two transformation steps:

(a) Replace each choice rule in P, say

r_i: $A \leftarrow$ **B**, **C**.

where i is the index of the rule, **C** is the conjunction of all choice goals and **B** is the conjunction of all remaining goals, with the two following three rules:

$A \leftarrow \mathbf{B}, chosen_i(\mathbf{Z})$.

$chosen_i(\mathbf{Z}) \leftarrow extChoice_i(\mathbf{Z}), \neg diffChoice_i(\mathbf{Z})$.

$extChoice_i(\mathbf{Z}) \leftarrow \mathbf{B}$.

where **Z** are all variables in the choice goals, listed in the order they occur in such goals, and

(b) For each goal *choice* ((**X**),(**Y**)) in C, add a new rule as follows;

$diffChoice_i(\mathbf{Z}) \leftarrow chosen_i(\mathbf{U}), \mathbf{Y} \neq \hat{\mathbf{Y}}$.

where **U** is a list of variable obtained from **Z** by replacing every variable Y in **Y** by a new variable $\hat{Y}$ and $\mathbf{Y} \neq \hat{\mathbf{Y}}$ is the conjunction of comparison predicates $Y \neq \hat{Y}$, one for each Y in **Y**.

PROPOSITION 8. *Let P be a choice program. Then $|SV(P)| = \Theta(|P| \times n)$, where $|SV(P)|$ and $|P|$ are the sizes of $SV(P)$ and P, respectively, and n is the maximal number of choice goals in any choice rule of P.* □

Example 13. Consider the choice program P of Example 12. The stable version of P is the program composed by the following rules:

$colored(G,C) \leftarrow color(C), glass(G), chosen(C,G)$.
$chosen(C,G) \leftarrow extChoice(C,G),$
$\neg diffChoice(C,G)$.
$extChoice(G,C) \leftarrow color(C), glass(G)$.
$diffChoice(C,G) \leftarrow chosen(C,\hat{G}), G \neq G'$.
$diffChoice(C,G) \leftarrow chosen(\hat{C},G), C \neq C'$.

and by all facts in P defining *color* and *glass*. (Note that, again, we did not introduce indices since there is only one choice rule in P.) It can been easily seen that $SV(P)$ can be read in the same way as P as follows: "From all possible combination of colors and glasses, choose a combination for which the same glass is colored with at most one color and the same color is used for at most one glass". □

It follows directly from the definition of $SV(P)$ that every total model of the stable version of a program is stable and corresponds to a choice model.

LEMMA 1. *Let P be a choice program. Then every stable model of $SV(P)$ is total and $SV(P)$ has at least one stable model.* □

THEOREM 6. *Let P be a choice program. Then*

(a) *for each choice model M for P, there exists a stable model N of $SV(P)$ such that $N/M = M$, and*

(b) *for each stable model N for $SV(P)$ there exists a choice model M for P such that $N/M = M$.* □

Thus, given a program with choice constructs there exists an equivalent program with negation, such that (disregarding new predicate symbols), the new program has a set of stable models which is exactly the set of choice models of the old program.

Example 14. Consider the program P of Example 12 and its stable version, $SV(P)$, of Example 13. It is easy to see that the choice models of P coincide with the stable models of $SV(P)$, modulo the predicates with symbols *chosen*, *extChoice* and *diffChoice*. Consider now the program of Example 11. Its stable version is the following program:

$a\_st(St, Crs) \leftarrow takes(St, Crs),\ chosen(Crs, St).$
$chosen(Crs, St) \leftarrow extChoice(Crs, St),$
$\qquad\qquad\neg diffChoice(Crs, St).$
$extChoice(Crs, St) \leftarrow takes(St, Crs).$
$diffChoice(Crs, St) \leftarrow chosen(Crs, \overline{St}),\ St \neq \overline{St}.$
$takes(andy, engl).$
$takes(ann, math).$
$takes(mark, engl).$
$takes(mark, math).$

Note that the program in Example 8 can be obtained from the program above by removing some redundant predicates. (Indeed the predicates *chosen* and *a_st* are identical, and so are *takes* and *extChoice* .) These redundancies follows from the generality of the construction used to build the stable versions. □

We note that, in the definition of choice program, we have introduced two restrictions. The first restriction is that the head of each choice rule is not mutually recursive with any of the goals (stratified programs w.rt. *choice*). It can be shown that this restriction can be lifted without changing substantially the results; however, the definition of stable versions becomes more complex. For simplicity of presentation and because every program with *choice* in recursive rules can be substituted by a program stratified w.r.t. *choice*, we will not discuss here this general case.

The second restriction is that the choice program does not contain negative goals. Obviously, general programs with *choice* and negated goals in recursive rules are intractable inasmuch as there is no semantics for general programs with negation. However, some kind of negation (e.g., stratified negation [ABW, CH, N, V1]) can be allowed. For instance, it can be shown that most results of this section can be extended to programs stratified w.r.t. negation.

7. Backtracking Fixpoint

We have seen that stable models provide a generalization to well-founded semantics capable of expressing the notion of non-determinism. An apparent deficiency of stable models is that no constructive way is currently known to realize this semantics. We next address this problem by the introduction of a generalized fixpoint computation that uses non-determinism and backtracking. The computation of stable models is described in Figure 2.

In the procedure of Figure 2, we use the immediate consequence transformation T_P and its least fixpoint $T_P^\infty(\emptyset)$. Moreover, we denote by $\hat{P}$ the Horn version of a program P, i.e., the positive program obtained from P by viewing each negative literal $\neg p(A)$ as a new positive literal with predicate symbol $\neg p$. Let X be a set of negative ground literals (regarded as facts); following the notation in [V2], we define $S_{\hat{P}}(X) = T^\infty_{\hat{P}\cup X}(\emptyset) - X$ — i.e., the positive literals in the least fixpoint (and minimum model) of $\hat{P}$ given a fixed set of negative ground literals X.

```
begin
  $M_0 := S_P(\emptyset)$; $\tilde{M}_0 := \emptyset$;
  $i$ := 0; stable := true; done := false;
  while stable and not done do
      $i := i+1$;
      if $C_i = \emptyset$ then
            done := true
      else
            $L_i := order(C_i)$;
            conflict_rule := true;
            while stable and conflict_rule do
                 if $L_i \neq \emptyset$ then
                     $r := next(L_i)$;
                     $\tilde{M}_i := \tilde{M}_{i-1} \cup N(r)$;
                     $M_i := S_P(\tilde{M}_i)$;
                     conflict_rule := conflict($M_i, \tilde{M}_i$);
                 else
                     $i := i-1$;
                     if $i$=0 then stable := false endif
                 endif
            end
      endif
  end;
  if stable then
  output $M_{i-1}$ "is a stable model"
  else
  output "No stable models"
  endif
end.
```

Fig. 2. *Stable Backtracking Fixpoint*

The procedure of Figure 2 starts at level 0 by determining all ground predicates that can be inferred using only positive ground literals. In terms of the S_P notation, $M_0 = S_P(\emptyset)$ is computed. No negative ground literal is assumed: we set $\tilde{M}_0 = \emptyset$. Then we move up to level 1. Here, we consider the set C_1 of all rules in $ground(P)$ with negative literals in their bodies such that (i) all positive literals are in M_0, and (ii) all negative literals, as well as the head predicate, are not in M_0. More in general, at the generic level $i \geq 1$, C_i denotes the set of all rules r in $ground(P)$ such that:

(i) all positive literals in the body of r are in M_{i-1},

(ii) r has at least one negative literal, say $\neg N$, such that neither $\neg N$ is in $\tilde{M}_{i-1}$, nor N is in M_{i-1},

(iii) the head of r is not in M_{i-1}.

Thus, r has the following form (assuming that the lists of P's and $\neg\hat{N}$'s are not empty):

$$H \leftarrow P_1, \ldots, P_n, \neg\hat{N}_1, \ldots, \neg\hat{N}_m, \neg N_1, \ldots, \neg N_l.$$

such that

(1) $H \notin M_{i-1}$;

(2) P_j $(1 \le j \le n) \in M_{i-1}$;

(3) $\neg\hat{N}_j$ $(1 \le j \le m) \in \tilde{M}_{i-1}$;

(4) N_j $(1 \le j \le l) \notin M_{i-1}$ and $\neg N_j$ $(1 \le j \le l) \notin \tilde{M}_{i-1}$.

If C_i is empty, then we are done, and M_{i-1} is a stable model. Otherwise, all the rules in C_i are inserted into the list L_i in an arbitrary order (see function *order*). Then the first rule r is removed from L_i (see function *next*) and taken into consideration. Say that r has a structure shown above; then we add $\{\neg N_1, \ldots, \neg N_l\}$, returned by the function call $N(r)$, to the set $\tilde{M}_{i-1}$ of all negative ground literals that have been assumed up to level $i-1$. In this way, we obtain $\tilde{M}_i$, the set of all negative ground literals assumed up to level i; we use such negative literals to infer all possible positive ground literals through the program P, i.e., we compute M_i as $S_P(\tilde{M}_i)$. At this point, we invoke the function $conflict(M_i, \tilde{M}_i)$ which returns true only when there exists some Q in M_i such that $\neg Q$ is in $\tilde{M}_i$. If there is no conflict, then we move up to the next level; otherwise, we remain at level i and we retry with another rule in L_i. If L_i happens to be empty then we backtrack to the level $i-1$ and select another rule for this level. If we eventually get back to level 0, no more alternatives are possible and the procedure stops by declaring that the program has no stable models.

PROPOSITION 9. *The procedure "Stable Backtracking Fixpoint" applied to a program P has the following properties:*

(a) *if the procedure terminates then the result returned by the procedure is correct, and*

(b) *when H_P is finite, the procedure always terminates.* □

Thus, if the procedure terminates on P returning "no stable model", then P has no stable model. If the procedure terminates and returns a set of facts M_{i-1}, then M_{i-1} constitutes a stable model. However, termination can only be guaranteed for certain classes of programs, such as Datalog programs. The possibility of non-termination, is typical of constructive semantics of programs with negation, including well-founded models [P4, V2], and stratified models [ABW]. Indeed it is interesting to compare our backtracking fixpoint with the standard stratified fixpoint computation which is routinely used in practical applications [ABW].

Say then that we are given a stratified program, where $<$ is the total order among predicate names induced by the stratification. Now, we need to order rules in $ground(P)$ as follows. Given two rules r_1 and r_2 with head predicates symbols q_1 and q_2, then $r_1 < r_2$ if $q_1 < q_2$. We can now introduce the following simple constraint to the function *order*: if $r_1 < r_2$, while $r_2 < r_1$ does not hold, then r_1 must appear before r_2 in $L_i := order(C_i)$. Under such a constraint the procedure of Figure 2 never backtracks on stratified programs, and thus it reduces to the standard and efficient fixpoint-based computation of stratified programs [ABW].

A second class of programs, for which our procedure behaves surprising well even for infinite universes, are the stable versions of choice programs.

PROPOSITION 10. *Let P be a choice program. Then the procedure "Stable Backtracking Fixpoint" applied to SV(P) has the following properties:*

(a) *it never backtracks and never outputs "no stable models";*

(b) *for each level i, $M_i \subseteq M_{i+1}$ and $M_i \subseteq M$, where M is a stable model of $SV(P)$.* □

Thus the operational semantics of stable models confirms what we already know from the LDL implementation: making a choice is a complex operation from the semantic viewpoint, but computationally it is a rather simple one.

8. Conclusions

Starting from a new definition of partial model, we have singled out a number of interesting classes of models. In particular, we have shown that the novelty of stable models w.r.t. the well-founded model is that they introduce a kind of non-determinism in a logic program. This property has been then confirmed by the fact that the non-deterministic choice construct used in LDL [KN, Z] can be explained in terms of stable model semantics. Finally we have presented a procedure for computing stable models. Although its termination cannot be guaranteed for infinite universe, if the procedure terminates then it determines a total stable model (if any). A number of new applications are made possible by the formal semantics of non-determinism proposed here; for instance, it is used in [Z] to model the notion of object identity in predicates.

ACKNOWLEDGEMENTS. The authors would like to thank Luigi Palopoli, for several seminal discussions, and Vladimir Lifschitz and Francesca Rossi for their comments on a earlier version of this paper.

References

[AV] Abiteboul S., and V. Vianu, "Fixpoint Extensions of First-Order Logic and Datalog-Like Languages," *Proc. Fourth Symposium on Logic in Computer Science (LICS)*, IEEE Computer Press, pp. 71-89, 1989.

[ABW] Apt, K., Bair, H., and Walker, A., "*Towards a Theory of Declarative Knowledge*," Minker, J. (ed.), Morgan Kaufman, Los Altos, 1987, pp. 89-148.

[CH] Chandra, A., Harel, D., "Horn Clauses and Generalization", *Journal of Logic Programming 2*, 1, 1985, pp. 320-340.

[GL] Gelfond, M., Lifschitz, V., "The Stable Model Semantics for Logic Programming", *Proc. 5th Int. Conf. and Symp. on Logic Programming*, MIT Press, Cambridge, Ma, 1988, pp. 1070-1080.

[KN] Krishnamurthy, R. and Naqvi, S.A., "Non-Deterministic Choice in Datalog", *Proc. 3rd Int. Conf. on Data and Knowledge Bases*, Morgan Kaufmann Pub., Los Altos, 1988, pp. 416-424.

[KP] Kolaitis, P.G. and Papadimitriou C.H., " Why not Negation by Fixpoint," *ACM SIGMOD-SIGACT Symp. on Principles of Database Systems*, March 1988, pp. 231-239.

[L] Lloyd, J.W., *Foundations of Logic Programming*, Springer Verlag, Berlin, 1987.

[N] Naqvi, S.A., "A Logic for Negation in Database Systems," in *Foundations of Deductive Databases and Logic Programming, (Minker, J. ed.)*, Morgan Kaufman, Los Altos, 1987.

[NT] Naqvi S. and S. Tsur, "*A Logical Data Language for Data and Knowledge Bases*," Computer Science Press, New York, 1989.

[P1] Przymusinski, T.C., "On the Semantics of Stratified Deductive Databases and Logic Programs", *Journal of Automated Reasoning*, to appear.

[P2] Przymusinski, T.C., "On the Declarative and Procedural Semantics of Deductive Databases and Logic Programs", in *Foundations of Deductive Databases and Logic Programming, (Minker, J. ed.)*, Morgan Kaufman, Los Altos, 1987, pp. 193-216.

[P3] Przymusinski, T.C., "Perfect Model Semantics", *Proc. 5th Int. Conf. and Symp. on Logic Programming*, MIT Press, Cambridge, Ma, 1988, pp. 1081-1096.

[P4] Przymusinski, T.C., "Every Logic Program has a Natural Stratification and an Iterated Fixed Point Model", *Proc. ACM Symp. on Principles of Database Systems*, pp. 11-21, 1989.

[P5] Przymusinski, T.C., "Well-Founded Models are Intersections of 3-Valued Stable Models," e-mail communication, April 1989.

[PP] Przymusinska, H, Przymusinski, T.C., "Weakly Perfect Model Semantics for Logic Programs", *Proc. 5th Int. Conf. and Symp. on Logic Programming*, MIT Press, Cambridge, Ma, 1988, pp. 1106-1120.

[R] Ross, K., "A Procedural Semantics for Well-Founded Negation in Logic Programs," *Proc. ACM Symp. on Principles of Database Systems*, pp. 22-31, 1989.

[SZ] Saccà D. and C. Zaniolo, "Partial Models, Stable Models and Non-Determinism in Logic Programs with Negation," MCC/ACT Technical Report, January 1990.

[U] Ullman, J.D., *Principles of Database and Knowledge-Base Systems, Vol. 1 and 2*, Computer Science Press, Rockville, Md., 1989.

[V1] Van Gelder, A., "Negation as Failure Using Tight Derivations for Logic Programs," *Proc. 3rd IEEE Symp. on Logic Programming*, Springer-Verlag, 1986, pp. 127-138.

[V2] Van Gelder, A., "The Alternating Fixpoint of Logic Programs with Negation", *Proc. ACM Symp. on Principles of Database Systems*, pp. 1-10, 1989.

[VRS] Van Gelder, A., Ross, K., Schlipf, J.S., "Unfounded Sets and Well-Founded Semantics for General Logic Programs", *ACM SIGMOD-SIGACT Symp. on Principles of Database Systems*, March 1988, pp. 221-230.

[Z] Zaniolo, C. "Object Identity and Inheritance in Deductive Databases: an Evolutionary Approach," *Procs. 1st Int. Conf. on Deductive and* Object-Oriented Databases, Kyoto, Japan, 1989.