Thirteen Definitions of a Stable Model

Vladimir Lifschitz

Department of Computer Sciences, University of Texas at Austin, USA
vl@cs.utexas.edu

Abstract. Stable models of logic programs have been studied by many researchers, mainly because of their role in the foundations of answer set programming. This is a review of some of the definitions of the concept of a stable model that have been proposed in the literature. These definitions are equivalent to each other, at least when applied to traditional Prolog-style programs, but there are reasons why each of them is valuable and interesting. A new characterization of stable models can suggest an alternative picture of the intuitive meaning of logic programs; or it can lead to new algorithms for generating stable models; or it can work better than others when we turn to generalizations of the traditional syntax that are important from the perspective of answer set programming; or it can be more convenient for use in proofs; or it can be interesting simply because it demonstrates a relationship between seemingly unrelated ideas.

1 Introduction

Stable models of logic programs have been studied by many researchers, mainly because of their role in the foundations of answer set programming (ASP) [30, 37]. This programming paradigm provides a declarative approach to solving combinatorial search problems, and it has found applications in several areas of science and technology [21]. In ASP, a search problem is reduced to computing stable models, and programs for generating stable models ("answer set solvers") are used to perform search.

This paper is a review of some of the definitions, or characterizations, of the concept of a stable model that have been proposed in the literature. These definitions are equivalent to each other when applied to "traditional rules" – with an atom in the head and a list of atoms, some possibly preceded with the negation as failure symbol, in the body:

$$A_0 \leftarrow A_1, \ldots, A_m, not\ A_{m+1}, not\ A_n. \tag{1}$$

But there are reasons why each of them is valuable and interesting. A new characterization of stable models can suggest an alternative picture of the intuitive meaning of logic programs; or it can lead to new algorithms for generating stable models; or it can work better when we turn to generalizations of the traditional syntax that are important from the perspective of answer set programming; or it can be more convenient for use in proofs, such as proofs of correctness of ASP programs; or, quite simply, it can intellectually excite us by demonstrating a relationship between seemingly unrelated ideas.

We concentrate here primarily on programs consisting of finitely many rules of type (1), although generalizations of this syntactic form are mentioned several times in the second half of the paper. Some work on the stable model semantics, for instance [15, 23, 39, 2], is not discussed here because it is about extending, rather than modifying, the definitions proposed earlier; this kind of work does not tell us much new about stable models of traditional programs.

The paper begins with comments on the relevant work that had preceded the invention of stable models – on the semantics of logic programming (Sect. 2) and on formal nomonotonic reasoning (Sect. 3). Early contributions that can be seen as characterizations of the class of stable models in terms of nonmonotonic logic are discussed in Sect. 4. Then we review the "standard" definition of stable models – in terms of reducts (Sect. 5) – and turn to its characterizations in terms of unfounded sets and loop formulas (Sect. 6). After that, we talk about the definitions of stable models in terms of circumscription (Sect. 7) and in terms of support relative to a well-ordering (Sect. 8), discuss two characterizations based on tightening (Sect. 9), and talk about the relationship between stable models and equilibrium logic (Sect. 10).

In recent years, two interesting modifications of the definition of the reduct were introduced (Sect. 11). We have learned also that a simple change in the definition of circumscription can give a characterization of stable models (Sect. 12).

This is an extended version of the conference paper [20].

2 Minimal Models, Completion, and Stratified Programs

2.1 Minimal Models vs. Completion

According to [47], a logic program without negation represents the least (and so, the only minimal) Herbrand model of the corresponding set of Horn clauses. On the other hand, according to [4], a logic program represents a certain set of first-order formulas, called the program's completion.

These two ideas are closely related to each other, but not equivalent. Take, for instance, the program

$$\begin{array}{l} p(a,b). \\ p(X,Y) \leftarrow p(Y,X). \end{array} \tag{2}$$

The minimal Herbrand model

$$\{p(a,b), p(b,a)\} \tag{3}$$

of this program satisfies the program's completion

$$\forall XY(p(X,Y) \leftrightarrow ((X = a \wedge Y = b) \vee p(Y,X))) \wedge a \neq b.$$

But there also other Herbrand interpretations satisfying the program's completion – for instance,

$$\{p(a,b), p(b,a), p(b,b)\}. \tag{4}$$

Another example of this kind, important in applications, is given by the recursive definition of transitive closure:

$$\begin{array}{l} q(X,Y) \leftarrow p(X,Y). \\ q(X,Z) \leftarrow q(X,Y), q(Y,Z). \end{array} \tag{5}$$

The completion of the union of this program with a definition of p has, in many cases, unintended models, in which q is weaker than the transitive closure of p that we want to define.

Should we say then that Herbrand minimal models provide a better semantics for logic programming than program completion? Yes and no. The concept of completion has a fundamental advantage: it is applicable to programs with negation. Such a program, viewed as a set of clauses, usually has several minimal Herbrand models, and some of them may not satisfy the program's completion. Such "bad" models reflect neither the intended meaning of the program nor the behavior of Prolog. For instance, the program

$$\begin{array}{l} p(a). \quad p(b). \quad q(a). \\ r(X) \leftarrow p(X), \mathit{not}\ q(X). \end{array} \tag{6}$$

has two minimal Herbrand models:

$$\{p(a), p(b), q(a), r(b)\} \tag{7}$$

("good") and

$$\{p(a), p(b), q(a), q(b)\} \tag{8}$$

("bad"). The completion of (6)

$$\begin{array}{c} \forall X(p(X) \leftrightarrow (X = a \vee X = b)) \wedge \forall X(q(X) \leftrightarrow X = a) \\ \wedge \forall X(r(X) \leftrightarrow (p(X) \wedge \neg q(X))) \wedge a \neq b \end{array}$$

characterizes the good model.

2.2 The Challenge

In the 1980s, the main challenge in the study of the semantics of logic programming was to invent a semantics that

- in application to a program without negation, such as (2), describes the minimal Herbrand model,
- in the presence of negation, as in example (6), selects a "good" minimal model satisfying the program's completion.

Such a semantics was proposed in two papers presented at the 1986 Workshop on Foundations of Deductive Databases and Logic Programming [1, 48]. That approach was not without defects, however. First, it is limited to programs in which recursion and negation "don't mix." Such programs are called stratified. Unfortunately, some useful Prolog programs do not satisfy this condition. For instance, we can say that a position in a two-person game is winning if there exists a move from it to a non-winning position (cf. [46]). This rule is not stratified: it recursively defines winning in terms of non-winning. A really good semantics should be applicable to rules like this.

Second, the definition of the semantics of stratified programs is somewhat complicated. It is based on the concept of the iterated least fixpoint of a program, and to prove the soundness of this definition one needs to show that this fixpoint doesn't depend on the choice of a stratification. A really good semantics should be a little easier to define.

The stable model semantics, as well as the well-founded semantics proposed in [49, 50], can be seen as an attempt to generalize and simplify the iterated fixpoint semantics of stratified programs.

3 Nonmonotonic Reasoning

Many events in the history of research on stable models can be only understood if we think of it as part of a broader research effort – the investigation of nonmonotonic reasoning. Early work in this area was motivated primarily by the desire to understand and automate the use of defaults by humans. When commonsense reasoning exploits a default, there is a possibility that taking into account new information may force us to retract the conclusions that we have made. The same kind of nonmonotonicity is observed in the behavior of Prolog programs with negation. For instance, rules (6) warrant the conclusion $r(b)$, but if the fact $q(b)$ is added to the program then this conclusion will be retracted.

As to the stable model semantics, three nonmonotonic formalisms are particularly relevant.

3.1 Circumscription

Circumscription [31, 32] is a syntactic transformation that turns a first-order sentence F into the conjunction of F with another formula, which expresses a minimality condition (the exact form of that condition depends on the "circumscription policy"). This additional conjunctive term involves second-order quantifiers.

Circumscription generalizes the concept of a minimal model from [47]. The iterated fixpoint semantics of stratified programs can be characterized in terms of circumscription also [19]. On the other hand, circumscription is similar to program completion in the sense that both are syntactic transformations that make a formula stronger. The relationship between circumscription and program completion was investigated in [43].

3.2 Default Logic

A default theory in the sense of [42] is characterized by a set W of "axioms" – first-order sentences, and a set D of "defaults" – expressions of the form

$$\frac{F \ : \ \mathsf{M}\,G_1, \ldots, \mathsf{M}\,G_n}{H}, \tag{9}$$

where $F, G_1, \ldots, G_n, H$ are first-order formulas. The letter M, according to Reiter, is to be read as "it is consistent to assume." Intuitively, default (9) is similar to the inference rule allowing us to derive the conclusion H from the premise F, except that the applicability of this rule is limited by the justifications $G_1, \ldots, G_n$; deriving H is allowed only if each of the justifications can be "consistently assumed."

This informal description of the meaning of a default is circular: to decide which formulas can be derived using one of the defaults from D we need to know whether the justifications of that default are consistent with the formulas that can be derived from W using the inference rules of classical logic and the defaults from D – including the default that we are trying to understand! But Reiter was able to turn his intuition about M into a precise semantics. His theory of defaults tells us under what conditions a set E of sentences is an "extension" for the default theory with axioms W and defaults D.

In Sect. 4 we will see that one of the earliest incarnations of the stable model semantics was based on treating rules as defaults in the sense of Reiter.

3.3 Autoepistemic Logic

According to [36], autoepistemic logic "is intended to model the beliefs of an agent reflecting upon his own beliefs." Moore's definition of propositional autoepistemic logic builds on the ideas of [35] and [34].

Formulas of this logic are constructed from atoms using propositional connectives and the modal operator L ("is believed"). Its semantics specifies, for any set A of formulas ("axioms"), which sets of formulas are considered "stable expansions" of A. Intuitively, Moore explains, the stable expansions of A are "the possible sets of beliefs that a rational agent might hold, given A as his premises."

In Sect. 4 we will see that one of the earliest incarnations of the stable model semantics was based on treating rules as autoepistemic axioms in the sense of Moore. The term "stable model" is historically related to "stable expansions" of autoepistemic logic.

3.4 Relations between Nonmonotonic Formalisms

The intuitions underlying circumscription, default logic, and autoepistemic logic are different from each other, but related. For instance, circumscribing (that is, minimizing the extent of) a predicate p is somewhat similar to adopting the default

$$\frac{\text{true} \ : \ \mathsf{M}\,\neg p(X)}{\neg p(X)}$$

(if it is consistent to assume that X does not have the property p, conclude that it doesn't). On the other hand, Moore observes that “a formula is consistent if its negation is not believed”; accordingly, Reiter's M is somewhat similar to the combination $\neg\mathsf{L}\neg$ in autoepistemic logic, and default (9), in propositional case, is somewhat similar to the autoepistemic formula

$$F \wedge \neg\mathsf{L}\neg G_1 \wedge \cdots \wedge \neg\mathsf{L}\neg G_n \rightarrow H.$$

However, the task of finding precise and general relationships between these three formalisms turned out to be difficult. Discussing technical work on that topic is beyond the scope of this paper.

4 Definitions A and B, in Terms of Translations into Nonmonotonic Logic

The idea of [13] is to think of the expression *not A* in a logic program as synonymous with the autoepistemic formula $\neg\mathsf{L}A$ (“A is not believed”). Since autoepistemic logic is propositional, the program needs to be grounded before this transformation is applied. After grounding, each rule (1) is rewritten as a formula:

$$A_1 \wedge \cdots \wedge A_m \wedge \neg A_{m+1} \wedge \cdots \wedge \neg A_n \rightarrow A_0, \tag{10}$$

and then L inserted after each negation. For instance, to explain the meaning of program (6), we take the result of its grounding

$$\begin{array}{l} p(a). \quad p(b). \quad q(a). \\ r(a) \leftarrow p(a), \mathit{not}\ q(a). \\ r(b) \leftarrow p(b), \mathit{not}\ q(b). \end{array} \tag{11}$$

and turn it into a collection of formulas:

$$\begin{array}{l} p(a), \quad p(b), \quad q(a), \\ p(a) \wedge \neg\mathsf{L}\, q(a) \rightarrow r(a), \\ p(b) \wedge \neg\mathsf{L}\, q(b) \rightarrow r(b). \end{array}$$

The autoepistemic theory with these axioms has a unique stable expansion, and the atoms from that stable expansion form the intended model (7) of the program.

This epistemic interpretation of logic programs – what we will call *Definition A* – is more general than the iterated fixpoint semantics, and it is much simpler. One other feature of Definition A that makes it attractive is the simplicity of the underlying intuition: negation as failure expresses the absence of belief.

The "default logic semantics" proposed in [3] is translational as well; it interprets logic programs as default theories. The head A_0 of a rule (1) turns into the conclusion of the default, the conjunction $A_1 \wedge \cdots \wedge A_m$ of the positive members of the body becomes the premise, and each negative member *not* A_i turns into the justification $\mathsf{M}\neg A_i$ ("it is consistent to assume $\neg A_i$"). For instance, the last rule of program (6) corresponds to the default

$$\frac{p(X) \ : \ \mathsf{M}\neg q(X)}{r(X)}. \tag{12}$$

There is no need for grounding, because defaults are allowed to contain variables. This difference between the two translations is not essential though, because Reiter's semantics of defaults treats a default with variables as the set of its ground instances. Grounding is simply "hidden" in the semantics of default logic.

This *Definition B* of the stable model semantics stresses an analogy between rules in logic programming and inference rules in logic. Like Definition A, it has an epistemic flavor, because of the relationship between the "consistency operator" M in defaults and the autoepistemic "belief operator" L (Sect. 3.4).

The equivalence between these two approaches to semantics of traditional programs follows from the fact that each of them is equivalent to Definition C of a stable model reviewed in the next section. This was established in [14] for the autoepistemic semantics and in [29] for the default logic approach.

5 Definition C, in Terms of the Reduct

Definitions A and B are easy to understand – assuming that one is familiar with formal nonmonotonic reasoning. Can we make these definitions direct and avoid explicit references to autoepistemic logic and default logic?

This question has led to the most widely used definition of the stable model semantics, *Definition C* [14]. The *reduct* of a program Π relative to a set M of ground atoms is obtained from Π by grounding followed by

(i) dropping each rule (1) containing a term *not* A_i with $A_i \in M$, and
(ii) dropping the negative parts *not* $A_{m+1}, \ldots,$ *not* A_n from the bodies of the remaining rules.

We say that M is a *stable model* of Π if the minimal model of (the set of clauses corresponding to the rules of) the reduct of Π with respect to M equals M. For instance, the reduct of program (6) relative to (7) is

$$\begin{array}{l} p(a). \quad p(b). \quad q(a). \\ r(b) \leftarrow p(b). \end{array} \tag{13}$$

The minimal model of this program is the set (7) that we started with; consequently, that set is a stable model of (6).

Definition C was independently invented in [12].

6 Definitions D and E, in Terms of Unfounded Sets and Loop Formulas

From [45] we learned that stable models can be characterized in terms of the concept of an unfounded set, which was introduced in [49] as part of the definition of the well-founded semantics. Namely, a set M of atoms is a stable model of a (grounded) program Π iff

(i) M satisfies Π,[1] and
(ii) no non-empty subset of M is unfounded for Π with respect to M.[2]

This *Definition D* can be refined using the concept of a loop, introduced many years later in [27]. A *loop* of a grounded program Π is a non-empty set L of ground atoms such that any elements A, A' of L can be connected by a chain

$$A = A_1, A_2, \ldots, A_{k-1}, A_k = A' \qquad (k > 1)$$

of elements of L satisfying the following condition: for any $i = 1, \ldots, k-1$, the program contains a rule R that such that A_i is the head of R and A_{i+1} is a member of the body of R. For instance, the result of grounding program (2)

$$\begin{array}{l} p(a,b). \\ p(a,a) \leftarrow p(a,a). \\ p(a,b) \leftarrow p(b,a). \\ p(b,a) \leftarrow p(a,b). \\ p(b,b) \leftarrow p(b,b). \end{array} \tag{14}$$

has 3 loops:

$$\{p(a,a)\},\ \{p(b,b)\},\ \{p(a,b), p(b,a)\}. \tag{15}$$

Program (11) has no loops.

According to [17], if we require in condition (i) above that M satisfy the *completion* of the program, rather than the program itself, then it will be possible to relax condition (ii) and require only that no *loop* contained in M be unfounded; there will be no need then to refer to arbitrary non-empty subsets in that condition.

In [27] loops are used in a different way. That paper associates with every loop L of Π a certain propositional formula, called the *loop formula* for L. According to *Definition E*, M is a stable model of Π iff M satisfies the completion of Π conjoined with the loop formulas for all loops of Π. For instance, the loop formulas for the loops (15) of program (14) are

$$\begin{array}{l} p(a,a) \rightarrow \text{false}, \\ p(b,b) \rightarrow \text{false}, \\ (p(a,b) \vee p(b,a)) \rightarrow \text{true}. \end{array}$$

[1] That is, M satisfies the propositional formulas (10) corresponding to the rules of Π.

[2] To be precise, unfoundedness is defined with respect to a partial interpretation, not a set of atoms. But we are only interested here in the special case when the partial interpretation is complete, and assume that complete interpretations are represented by sets of atoms in the usual way.

The first two loop formulas eliminate all nonminimal models of the completion of (14).

The invention of loop formulas has led to the creation of systems for generating stable models that use SAT solvers for search ("SAT-based answer set programming"). Several systems of this kind performed well in a recent competition [5].

7 Definition F, in Terms of Circumscription

We saw in Sect. 4 that a logic program can be viewed as shorthand for an autoepistemic theory or a default theory. The characterization of stable models described in [24, Sect. 3.4.1] relates logic programs to the third nonmonotonic formalism reviewed above, circumscription. Like Definitions A and B, it is based on a translation, but the output of that translation is not simply a circumscription formula; it involves also some additional conjunctive terms.

The first step of that translation consists in replacing the occurrences of each predicate symbol p in the negative parts $\neg A_{m+1} \wedge \cdots \wedge \neg A_n$ of the formulas (10) corresponding to the rules of the program with a new symbol p' and forming the conjunction of the universal closures of the resulting formulas. The sentence obtained in this way is denoted by $C(\Pi)$. For instance, if Π is (6) then $C(\Pi)$ is

$$p(a) \wedge p(b) \wedge q(a) \wedge \forall X (p(X) \wedge \neg q'(X) \rightarrow r(X)).$$

The translation of Π is a conjunction of two sentences: the circumscription of the old (non-primed) predicates in $C(\Pi)$ and the formulas asserting, for each of the new predicates, that it is equivalent to the corresponding old predicate. For instance, the translation of (6) is

$$\mathrm{CIRC}[C(\Pi)] \wedge \forall X (q'(X) \leftrightarrow q(X)); \tag{16}$$

the circumscription operator CIRC is understood here as the minimization of the extents of p, q, r.

The stable models of Π can be characterized as the Herbrand interpretations satisfying the translation of Π, with the new (primed) predicates removed from them ("forgotten").

An interesting feature of this *Definition F* is that, unlike Definitions A–E, it does not involve grounding. We can ask what non-Herbrand models of the translation of a logic program look like. Can it be convenient in some cases to represent possible states of affairs by such "non-Herbrand stable models" of a logic program? A non-Herbrand model may include an object that is different from the values of all ground terms, or there may be several ground terms having the same value in it; can this be sometimes useful?

We will return to the relationship between stable models and circumscription in Sect. 12.

8 Definition G, in Terms of Support

As observed in [1], an Herbrand model M of a grounded program satisfies the program's completion iff every element A of M is *supported*, in the sense that the program contains a rule (1) such that

$$A_0 = A, \quad A_1, \ldots, A_m \in M, \quad A_{m+1}, \ldots, A_n \notin M. \tag{17}$$

A stronger form of this condition, given in [6], characterizes the class of stable models. According to his *Definition G*, an Herbrand model M of a grounded program is stable if there exists a well-ordering $\leq$ of M such that every element A of M is *supported relative to this well-ordering*, in the sense that the program contains a rule (1) satisfying conditions (17) and

$$A_1, \ldots, A_m < A_0.$$

For instance, the stable model (3) of program (14) is supported relative to the order $p(a, b) < p(b, a)$. On the other hand, model (4) is supported, but it is easy to see that it is not supported relative to any ordering of its elements.[3]

When M is finite, well-orderings of M can be described in terms of functions from atoms to integers, and supportedness relative to an order relation can be described by a formula of difference logic – the extension of propositional logic that includes variables for integers and atomic formulas of the form $x - y \geq c$. This observation suggests the posibility of using solvers for difference logic to generate stable models [38].

9 Definitions H and I, in Terms of Tightening and the Situation Calculus

We will talk now about two characterizations of stable models that are based, like Definition F, on translations into classical logic that use auxiliary predicates.

Programs that have no loops, such as (11), are called *tight*. The stable models of a tight program are identical to the Herbrand models of its completion [8]. *Definition H* [51] is based on a process of "tightening" that makes an arbitrary traditional program tight. This process uses two auxiliary symbols: the object constant 0 and the unary function constant s ("successor"). The tightened program uses also auxiliary predicates with an additional numeric argument. Intuitively, $p(X, N)$ expresses that there exists a sequence of N "applications" of rules of the program that "establishes" $p(X)$. The stable models of a program are described then as Herbrand models of the completion of the result of its tightening, with the auxiliary symbols "forgotten."

[3] To be precise, the characterization of stable models in [6] is limited to finite models, and it uses different terminology.

We will not reproduce here the definition of tightening, but here is an example: the result of tightening program (6) is

$$\begin{array}{l}
p(a, s(N)). \quad p(b, s(N)). \quad q(a, s(N)). \\
r(X, s(N)) \leftarrow p(X, N), \textit{not } q(X). \\
p(X) \leftarrow p(X, N). \\
q(X) \leftarrow q(X, N). \\
r(X) \leftarrow r(X, N).
\end{array}$$

Rules in line 1 tell us that $p(a)$ can be established in any number of steps that is greater than 0; similarly for $p(b)$ and $q(a)$. According to line 2, $r(X)$ can be established in $N + 1$ steps if $p(X)$ can be established in N steps and $q(X)$ cannot be established at all (note that an occurrence of a predicate does not get an additional numeric argument if it is negated). Finally, an atom holds if it can be established by some number N of rule applications.

Definition I [25] treats a rule in a logic program as an abbreviated description of the effect of an action – the action of "applying" that rule – in the situation calculus.[4] For instance, if the action corresponding to the last rule of (6) is denoted by $\textit{lastrule}(X)$ then that rule can be viewed as shorthand for the situation calculus formula

$$p(X, S) \wedge \neg\exists S(q(X, S)) \rightarrow r(X, do(\textit{lastrule}(X), S))$$

(if $p(X)$ holds in situation S and $q(X)$ does not hold in any situation then $r(X)$ holds after executing action $\textit{lastrule}(X)$ in situation S).

In this approach to stable models, the situation calculus function *do* plays the same role as adding 1 to N in Wallace's theory. Instead of program completion, Lin and Reiter use the process of turning effect axioms into successor state axioms, which is standard in applications of the situation calculus.

10 Definition J, in Terms of Equilibrium Logic

The logic of here-and-there, going back to the early days of modern logic [16], is a modification of classical propositional logic in which propositional interpretations in the usual sense – assignments, or sets of atoms – are replaced by pairs (X, Y) of sets of atoms such that $X \subseteq Y$. (We think of X as the set of the atoms that are true "here", and Y as the set of the atoms that are true "there.") The semantics of this logic defines when (X, Y) *satisfies* a formula F.

In [40], the logic of here-and-there was used as a starting point for defining a nonmonotonic logic closely related to stable models. According to Pearce, a pair (Y, Y) is an *equilibrium model* of a propositional formula F if F is satisfied in the logic of here-and-there by (Y, Y) but is not satisfied by (X, Y) for any proper subset X of Y. Pearce showed that a set M of atoms is a stable model

[4] See [44] for a detailed description of the situation calculus [33] as developed by the Toronto school.

of a program Π iff (M, M) is an equilibrium model of the set of propositional formulas (10) corresponding to the grounded rules of Π.

This *Definition J* is important for two reasons. First, it suggests a way to extend the concept of a stable model from traditional rules – formulas of form (1) – to arbitrary propositional formulas: we can say that M is a stable model of a propositional formula F if (M, M) is an equilibrium model of F. This is valuable from the perspective of answer set programming, because many "nonstandard" constructs commonly used in ASP programs, such as choice rules and weight constraints, can be viewed as abbreviations for propositional formulas [11]. Second, Definition J is a key to the theorem about the relationship between the concept of strong equivalence and the logic of here-and-there [22].

11 Definitions K and L, in Terms of Modified Reducts

In [7] the definition of the reduct reproduced in Sect. 5 is modified by including the positive members of the body, along with negative members, in the description of step (i), and by removing step (ii) altogeher. In other words, in the modified process of constructing the reduct relative to M we delete from the program all rules (1) containing in their bodies a term A_i such that $A_i \notin M$ or a term *not* A_i such that $A_i \in M$; the other rules of the program remain unchanged. For instance, the modified reduct of program (6) relative to (7) is

$$\begin{array}{l} p(a). \quad p(b). \quad q(a). \\ r(b) \leftarrow p(b), \mathit{not}\ q(b). \end{array}$$

Unlike the reduct (13), this modified reduct contains negation as failure in the last rule. Generally, unlike the reduct in the sense of Sect. 5, the modified reduct of a program has several minimal models.

According to *Definition K*, M is a stable model of Π if M is a minimal model of the modified reduct of Π relative to M.

In [9] the definition of the reduct is modified in a different way. The reduct of a program Π in the sense of Ferraris is obtained from the formulas (10) corresponding to the grounding rules of Π by replacing every maximal subformula of F that is not satisfied by M with "false". For instance, the formulas corresponding to the grounded rules (11) of (6) are the formulas

$$\begin{array}{l} p(a), \quad p(b), \quad q(a), \\ \text{false} \rightarrow \text{false}, \\ p(b) \wedge \neg\,\text{false} \rightarrow r(b). \end{array}$$

Definition L: M is a stable model of Π if M is a minimal model of the reduct of Π in the sense of Ferraris relative to M.

Definitions K and L are valuable because, like definition J, they can be extended to some nontraditional programs. Definition K was introduced, in fact, in connection with the problem of extending the stable model semantics to programs with aggregates. Definition L provides a satisfactory solution to the problem of aggregates as well. Furthermore, it can be applied in a straightforward way to arbitrary propositional formulas, and this generalization of the stable model semantics turned out to be equivalent to the generalization based on equilibrium logic that was mentioned at the end of Sect. 10.

12 Definition M, in Terms of Modified Circumscription

The authors of [10] defined a modification of circumscription called the *stable model operator*, SM. According to their *Definition M*, an Herbrand interpretation is a stable model of Π iff it satisfies SM$[F]$ for the conjunction F of the universal closures of the formulas (10) corresponding to the rules of Π.

Syntactically, the difference between SM and circumscription is really minor. If F contains neither implications nor negations then SM$[F]$ does not differ from CIRC$[F]$ at all. If F has "one level of implications" and no negations (as, for instance, when F corresponds to a set of traditional rules without negation, such as (2) and (5)), SM$[F]$ is equivalent to CIRC$[F]$. But SM becomes essentially different from CIRC as soon as we allow negation in the bodies of rules.

The difference between SM$[F]$ and the formulas used in Definition F is that the former does not involve auxiliary predicates and consequently does not require additional conjunctive terms relating auxiliary predicates to the predicates occurring in the program.

Definition M combines the main attractive feature of Definitions F, H, and I – no need for grounding – with the main attractive feature of Definitions J and L – applicability to formulas of an arbitrarily complex logical form. This fact makes it possible to give a semantics for an ASP language with choice rules and the *count* aggregate without any references to grounding [18].

Among the other definitions of a stable model discussed in this paper, Definition J, based on equilibrium logic, is the closest relative of Definition M. Indeed, in [41] the semantics of equilibrium logic is expressed by quantified Boolean formulas, and we can say that Definition M eliminated the need to ground the program using the fact that the approach of that paper can be easily extended from propositional formulas to first-order formulas.

A characterization of stable models that involves grounding but is otherwise similar to Definition M is given in [28]. It has emerged from research on the nonmonotonic logic of knowledge and justified assumptions [26].

13 Conclusion

Research on stable models has brought us many pleasant surprises.

At the time when the theory of iterated fixpoints of stratified programs was the best available approach to semantics of logic programming, it was difficult to expect that an alternative as general and as simple as Definition C would be found. And prior to the invention of Definition L, who would think that Definition C can be extended to choice rules, aggregates and more without paying any price in terms of the simplicity of the process of constructing the reduct?

A close relationship between stable models and the logic of here-and-there – a nonclassical logic that had been invented decades before the emergence of logic programming – was a big surprise. The possibility of defining stable models by twisting the definition of circumscription just a little was a surprise too.

There was a time when the completion semantics, the well-founded semantics, and the stable model semantics – and a few others – were seen as rivals; every person interested in the semantics of negation in logic programming would tell you then which one was his favorite. Surprisingly, these bitter rivals turned out to be so closely related to each other on a technical level that they eventually became good friends. One cannot study the algorithms used today for generating stable models without learning first about completion and unfounded sets.

And maybe the biggest surprise of all was that an attempt to clarify some semantic issues related to negation in Prolog was destined to be enriched by computational ideas coming from research on the design of SAT solvers and to give rise to a new knowledge representation paradigm, answer set programming.

Acknowledgements

I am grateful to the editors for the invitation to contribute this paper to a volume in honor of my old friend and esteemed colleague Yuri Gurevich.

Many thanks to Michael Gelfond, Tomi Janhunen, Joohyung Lee, Nicola Leone, Yuliya Lierler, Fangzhen Lin, Victor Marek, and Mirek Truszczyński for their comments. This work was partially supported by the National Science Foundation under Grant IIS-0712113.

References

1. Krzysztof Apt, Howard Blair, and Adrian Walker. Towards a theory of declarative knowledge. In Jack Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 89–148. Morgan Kaufmann, San Mateo, CA, 1988.
2. Marcello Balduccini and Michael Gelfond. Logic programs with consistency-restoring rules.[5]. In *Working Notes of the AAAI Spring Symposium on Logical Formalizations of Commonsense Reasoning*, 2003.
3. Nicole Bidoit and Christine Froidevaux. Minimalism subsumes default logic and circumscription in stratified logic programming. In *Proceedings LICS-87*, pages 89–97, 1987.
4. Keith Clark. Negation as failure. In Herve Gallaire and Jack Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum Press, New York, 1978.
5. Marc Denecker, Joost Vennekens, Stephen Bond, Martin Gebser, and Mirosław Truszczynski. The second answer set programming system competition.[6]. In *Proceedings of the International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, 2009.
6. Charles Elkan. A rational reconstruction of nonmonotonic truth maintenance systems. *Artificial Intelligence*, 43:219–234, 1990.

[5] `http://www.krlab.cs.ttu.edu/papers/download/bg03.pdf`
[6] `http://www.cs.kuleuven.be/∼dtai/events/asp-competition/paper.pdf`

7. Wolfgang Faber, Nicola Leone, and Gerald Pfeifer. Recursive aggregates in disjunctive logic programs: Semantics and complexity. In *Proceedings of European Conference on Logics in Artificial Intelligence (JELIA)*, 2004.
8. François Fages. A fixpoint semantics for general logic programs compared with the well–supported and stable model semantics. *New Generation Computing*, 9:425–443, 1991.
9. Paolo Ferraris. Answer sets for propositional theories. In *Proceedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, pages 119–131, 2005.
10. Paolo Ferraris, Joohyung Lee, and Vladimir Lifschitz. A new perspective on stable models. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pages 372–379, 2007.
11. Paolo Ferraris and Vladimir Lifschitz. Weight constraints as nested expressions. *Theory and Practice of Logic Programming*, 5:45–74, 2005.
12. Kit Fine. The justification of negation as failure. In *Proceedings of the Eighth International Congress of Logic, Methodology and Philosophy of Science*, pages 263–301. North Holland, 1989.
13. Michael Gelfond. On stratified autoepistemic theories. In *Proceedings of National Conference on Artificial Intelligence (AAAI)*, pages 207–211, 1987.
14. Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In Robert Kowalski and Kenneth Bowen, editors, *Proceedings of International Logic Programming Conference and Symposium*, pages 1070–1080. MIT Press, 1988.
15. Michael Gelfond and Vladimir Lifschitz. Logic programs with classical negation. In David Warren and Peter Szeredi, editors, *Proceedings of International Conference on Logic Programming (ICLP)*, pages 579–597, 1990.
16. Arend Heyting. Die formalen Regeln der intuitionistischen Logik. *Sitzungsberichte der Preussischen Akademie von Wissenschaften. Physikalisch-mathematische Klasse*, pages 42–56, 1930.
17. Joohyung Lee. A model-theoretic counterpart of loop formulas. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pages 503–508. Professional Book Center, 2005.
18. Joohyung Lee, Vladimir Lifschitz, and Ravi Palla. A reductive semantics for counting and choice in answer set programming. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 472–479, 2008.
19. Vladimir Lifschitz. On the declarative semantics of logic programs with negation. In Jack Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 177–192. Morgan Kaufmann, San Mateo, CA, 1988.
20. Vladimir Lifschitz. Twelve definitions of a stable model. In *Proceedings of International Conference on Logic Programming (ICLP)*, pages 37–51, 2008.
21. Vladimir Lifschitz. What is answer set programming? In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 1594–1597. MIT Press, 2008.
22. Vladimir Lifschitz, David Pearce, and Agustin Valverde. Strongly equivalent logic programs. *ACM Transactions on Computational Logic*, 2:526–541, 2001.
23. Vladimir Lifschitz, Lappoon R. Tang, and Hudson Turner. Nested expressions in logic programs. *Annals of Mathematics and Artificial Intelligence*, 25:369–389, 1999.
24. Fangzhen Lin. *A Study of Nonmonotonic Reasoning.* PhD thesis, Stanford University, 1991.
25. Fangzhen Lin and Raymond Reiter. Rules as actions: A situation calculus semantics for logic programs. *Journal of Logic Programming*, 31:299–330, 1997.

26. Fangzhen Lin and Yuting Zhao. ASSAT: Computing answer sets of a logic program by SAT solvers. In *Proceedings of National Conference on Artificial Intelligence (AAAI)*, pages 112–117. MIT Press, 2002.
27. Fangzhen Lin and Yuting Zhao. ASSAT: Computing answer sets of a logic program by SAT solvers. *Artificial Intelligence*, 157:115–137, 2004.
28. Fangzhen Lin and Yi Zhou. From answer set logic programming to circumscription via logic of GK. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, 2007.
29. Victor Marek and Mirosław Truszczyński. Stable semantics for logic programs and default theories. In *Proceedings North American Conf. on Logic Programming*, pages 243–256, 1989.
30. Victor Marek and Mirosław Truszczyński. Stable models and an alternative logic programming paradigm. In *The Logic Programming Paradigm: a 25-Year Perspective*, pages 375–398. Springer Verlag, 1999.
31. John McCarthy. Circumscription—a form of non-monotonic reasoning. *Artificial Intelligence*, 13:27–39,171–172, 1980.
32. John McCarthy. Applications of circumscription to formalizing common sense knowledge. *Artificial Intelligence*, 26(3):89–116, 1986.
33. John McCarthy and Patrick Hayes. Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, volume 4, pages 463–502. Edinburgh University Press, Edinburgh, 1969.
34. Drew McDermott. Nonmonotonic logic II: Nonmonotonic modal theories. *Journal of ACM*, 29(1):33–57, 1982.
35. Drew McDermott and Jon Doyle. Nonmonotonic logic I. *Artificial Intelligence*, 13:41–72, 1980.
36. Robert Moore. Semantical considerations on nonmonotonic logic. *Artificial Intelligence*, 25(1):75–94, 1985.
37. Ilkka Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 25:241–273, 1999.
38. Ilkka Niemelä. Stable models and difference logic. *Annals of Mathematics and Artificial Intelligence*, 53:313–329, 2008.
39. Ilkka Niemelä and Patrik Simons. Extending the Smodels system with cardinality and weight constraints. In Jack Minker, editor, *Logic-Based Artificial Intelligence*, pages 491–521. Kluwer, 2000.
40. David Pearce. A new logical characterization of stable models and answer sets. In Jürgen Dix, Luis Pereira, and Teodor Przymusinski, editors, *Non-Monotonic Extensions of Logic Programming (Lecture Notes in Artificial Intelligence 1216)*, pages 57–70. Springer, 1997.
41. David Pearce, Hans Tompits, and Stefan Woltran. Encodings for equilibrium logic and logic programs with nested expressions. In *Proceedings of Portuguese Conference on Artificial Intelligence (EPIA)*, pages 306–320, 2001.
42. Raymond Reiter. A logic for default reasoning. *Artificial Intelligence*, 13:81–132, 1980.
43. Raymond Reiter. Circumscription implies predicate completion (sometimes). In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pages 418–420, 1982.
44. Raymond Reiter. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems.* MIT Press, 2001.

45. Domenico Saccá and Carlo Zaniolo. Stable models and non-determinism in logic programs with negation. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 205–217, 1990.
46. Maarten van Emden and Keith Clark. The logic of two-person games. In *Micro-PROLOG: Programming in Logic*, pages 320–340. Prentice-Hall, 1984.
47. Maarten van Emden and Robert Kowalski. The semantics of predicate logic as a programming language. *Journal of ACM*, 23(4):733–742, 1976.
48. Allen Van Gelder. Negation as failure using tight derivations for general logic programs. In Jack Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 149–176. Morgan Kaufmann, San Mateo, CA, 1988.
49. Allen Van Gelder, Kenneth Ross, and John Schlipf. Unfounded sets and well-founded semantics for general logic programs. In *Proceedings of the Seventh ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, March 21-23, 1988, Austin, Texas*, pages 221–230. ACM Press, 1988.
50. Allen Van Gelder, Kenneth Ross, and John Schlipf. The well-founded semantics for general logic programs. *Journal of ACM*, 38(3):620–650, 1991.
51. Mark Wallace. Tight, consistent and computable completions for unrestricted logic programs. *Journal of Logic Programming*, 15:243–273, 1993.