Twelve Definitions of a Stable Model

Vladimir Lifschitz

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

Abstract. 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

This 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 (ASP); 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, [13], [18], [33], [2]) is not discussed here simply 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 (Section 2) and on formal nomonotonic reasoning (Section 3). Early contributions

M. Garcia de la Banda and E. Pontelli (Eds.): ICLP 2008, LNCS 5366, pp. 37–51, 2008.

that can be seen as characterizations of the class of stable models in terms of nonmonotonic logic are discussed in Section 4. Then we review the definition of stable models in terms of reducts (Section 5) and turn to its characterizations in terms of unfounded sets and loop formulas (Section 6). After that, we talk about three definitions of a stable model that use translations into classical logic (Sections 7 and 8) and about the relation between stable models and equilibrium logic (Section 9).

In recent years, two interesting modifications of the definition of the reduct were introduced (Section 10). And we learned that a simple change in the definition of circumscription can give a characterization of stable models (Section 11).

2 Minimal Models, Completion, and Stratified Programs

2.1 Minimal Models vs. Completion

According to [41], 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

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

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, one that makes p identically true.

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

$$\begin{array}{l} q(X,Y) \leftarrow p(X,Y). \\ q(X,Z) \leftarrow q(X,Y), q(Y,Z). \end{array} \tag{3}$$

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

$$\begin{array}{l} p(a). \quad p(b). \quad q(a). \\ r(X) \leftarrow p(X), \mathit{not}\ q(X). \end{array} \tag{4}$$

has two minimal Herbrand models:

$$\{p(a), p(b), q(a), r(b)\} \tag{5}$$

("good") and

$$\{p(a), p(b), q(a), q(b)\} \tag{6}$$

("bad"). The completion of (4)

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

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 (4), 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], [44]. 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. [40]). 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 [42,43], 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. Three theories of nonmonotonic reasoning are particularly relevant.

3.1 Circumscription

Circumscription [28,28,29] 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 [41]. The iterated fixpoint semantics of stratified programs can be characterized in terms of circumscription also [20]. 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 [37].

3.2 Default Logic

A default theory in the sense of [36] is characterized by a set W of "axioms"—first-order sentences, and a set D of "defaults"—expressions of the form

$$\frac{F \ : \ \mathsf{M}\, G_1, \ldots, \mathsf{M}\, G_n}{H}, \tag{7}$$

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 (7) 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 Section 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 [32], autoepistemic logic "is intended to model the beliefs of an agent reflecting upon his own beliefs." The definition of propositional autoepistemic logic builds on the ideas of [30] and [31].

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 Section 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 (7), 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 [14] 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:

$$A_1 \wedge \cdots \wedge A_m \wedge \neg A_{m+1} \wedge \cdots \wedge \neg A_n \rightarrow A_0, \tag{8}$$

and then L inserted after each negation. For instance, to explain the meaning of program (4), we take the result of its grounding

$$\begin{array}{l} p(a). \quad p(b). \quad q(a). \\ r(a) \leftarrow p(a), \mathit{not}\ q(a). \\ r(b) \leftarrow p(b), \mathit{not}\ q(b). \end{array} \tag{9}$$

and turn it into a collection of formulas:

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

The autoepistemic theory with these axioms has a unique stable expansion, and the atoms from that stable expansion form the intended model (5) 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 (4) corresponds to the default

$$\frac{p(X) \ : \ \mathsf{M}\, \neg q(X)}{r(X)}. \tag{10}$$

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 (Section 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 [12] for the autoepistemic semantics and in [26] 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* [12]. The *reduct* of a program Π relative to a set M of 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 X equals X. For instance, the reduct of program (4) relative to (5) is

$$\begin{array}{l} p(a). \quad p(b). \quad q(a). \\ r(b) \leftarrow p(b). \end{array} \tag{11}$$

The minimal model of this program is the set (5) that we started with; consequently, that set is a stable model of (4).

Definition C was independently invented in [10].

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

According to [39], stable models can be characterized in terms of the concept of an unfounded set, which was introduced in [42] 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 nonempty subset of M is unfounded for Π with respect to M.[2]

According to [17], this *Definition D* can be refined using the concept of a *loop*, introduced many years later by [23]. If we require, in condition (i), 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 nonempty subsets in that condition.

In [23] loops are used in a different way. They associated with every loop X of Π a certain propositional formula, called the *loop formula* for X. According to their *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.

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 ASP system competition [11].

[1] That is, M satisfies the propositional formulas (8) 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.

7 Definition F, in Terms of Circumscription

We saw in Section 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 [25, Section 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 (8) 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 (4) 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 (4) is

$$\mathrm{CIRC}[C(\Pi)] \wedge \forall X(q'(X) \leftrightarrow q(X)); \tag{12}$$

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 Section 11.

8 Definitions G and H, 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.

For a class of logic programs called tight, stable models are identical to Herbrand models of the program's completion [6]. (Programs (2) and (3), used above to illustrate peculiarities of the completion semantics, are not tight.) *Definition G*

[45] 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"). Besides, the tightened program uses 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."

We will not reproduce here the definition of tightening, but here is an example: the result of tightening program (4) is

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

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 H [21] 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.[3] For instance, if the action corresponding to the last rule of (4) is denoted by $\mathit{lastrule}(X)$ then that rule can be viewed as shorthand for the situation calculus formula

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

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

9 Definition I, in Terms of Equilibrium Logic

The logic of here-and-there, going back to the early days of modern logic [15], 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)

[3] See [38] for a detailed description of the situation calculus [27] as developed by the Toronto school.

of sets of atoms such that $X \subseteq Y$. (We think of X as the set of 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 [35], the logic of here-and-there was used as a starting point for defining a nonmonotonic logic closely related to stable models. According to that paper, 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. A set M of atoms is a stable model of a program Π iff (M, M) is an equilibrium model of the set of propositional formulas (8) corresponding to the grounded rules of Π.

This *Definition I* 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 [7]. Second, Definition I is a key to the theorem about the relationship between the concept of strong equivalence and the logic of here-and-there [19].

10 Definitions J and K, in Terms of Modified Reducts

In [5] the definition of the reduct reproduced in Section 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 (4) relative to (5) is

$$p(a). \quad p(b). \quad q(a).$$
$$r(b) \leftarrow p(b), not\ q(b).$$

Unlike the reduct (11), this modified reduct contains negation as failure in the last rule. Generally, unlike the reduct in the sense of Section 5, the modified reduct of a program has several minimal models.

According to *Definition J*, M is a stable model of Π iff 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 [9] is obtained from the formulas (8) 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 (9) of (4) are the formulas

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

Definition K: M is a stable model of Π iff M is a minimal model of the reduct of Π in the sense of [9] relative to M.

Definitions J and K are valuable because, like definition I, they can be extended to some nontraditional programs. The former was introduced, in fact, in connection with the problem of extending the stable model semantics to programs with aggregates. The latter 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 Section 9.

11 Definition L, in Terms of Modified Circumscription

In [8] a modification of circumscription is defined that is called the *stable model operator*, SM. According to their *Definition L*, an Herbrand interpretation M is a stable model of Π iff M satisfies SM[F] for the conjunction F of the universal closures of the formulas (8) 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 (3)), 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 L combines the main attractive feature of Definitions F, G, and H—no need for grounding—with the main attractive feature of Definitions I and K—applicability to formulas of arbitrarily complex logical form. In [16] this fact is used to give a semantics for an ASP language with choice rules and aggregates without any references to grounding.

Among the other definitions of a stable model discussed in this paper, Definition I, based on equilibrium logic, is the closest relative of Definition L. Indeed, in [34] the semantics of equilibrium logic is expressed by quantified Boolean formulas, and we can say that Definition L 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 L is given in [24]. It has emerged from research on the nonmonotonic logic of knowledge and justified assumptions [22].

12 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 K, who could 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 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

Many thanks to Michael Gelfond, Joohyung Lee, Nicola Leone, Yuliya Lierler, Fangzhen Lin,Victor Marek, and Mirek Truszczyński for comments on a draft of this note. I am also grateful to Mirek and to Andrea Formisano for the invitation to contribute a paper to the special session on stable models planned as part of ICLP'08. This work was partially supported by the National Science Foundation under Grant IIS-0712113.

References

1. Apt, K., Blair, H., Walker, A.: Towards a theory of declarative knowledge. In: Minker, J. (ed.) Foundations of Deductive Databases and Logic Programming, pp. 89–148. Morgan Kaufmann, San Mateo (1988)
2. Balduccini, M., Gelfond, M.: Logic programs with consistency-restoring rules. In: Working Notes of the AAAI Spring Symposium on Logical Formalizations of Commonsense Reasoning (2003), `http://www.krlab.cs.ttu.edu/papers/download/bg03.pdf`
3. Bidoit, N., Froidevaux, C.: Minimalism subsumes default logic and circumscription in stratified logic programming. In: Proc. LICS 1987, pp. 89–97 (1987)
4. Clark, K.: Negation as failure. In: Gallaire, H., Minker, J. (eds.) Logic and Data Bases, pp. 293–322. Plenum Press, New York (1978)

5. Faber, W., Leone, N., Pfeifer, G.: Recursive aggregates in disjunctive logic programs: Semantics and complexity. In: Alferes, J.J., Leite, J. (eds.) JELIA 2004. LNCS, vol. 3229. Springer, Heidelberg (2004)
6. Fages, F.: A fixpoint semantics for general logic programs compared with the well–supported and stable model semantics. New Generation Computing 9, 425–443 (1991)
7. Ferraris, P., Lifschitz, V.: Weight constraints as nested expressions. Theory and Practice of Logic Programming 5, 45–74 (2005)
8. Ferraris, P., Lee, J., Lifschitz, V.: A new perspective on stable models. In: Proceedings of International Joint Conference on Artificial Intelligence (IJCAI), pp. 372–379 (2007)
9. Ferraris, P.: Answer sets for propositional theories. In: Baral, C., Greco, G., Leone, N., Terracina, G. (eds.) LPNMR 2005. LNCS, vol. 3662, pp. 119–131. Springer, Heidelberg (2005)
10. Fine, K.: The justification of negation as failure. In: Proceedings of the Eighth International Congress of Logic, Methodology and Philosophy of Science, pp. 263–301. North Holland, Amsterdam (1989)
11. Gebser, M., Liu, L., Namasivayam, G., Neumann, A., Schaub, T., Truszczyński, M.: The first answer set programming system competition. In: Baral, C., Brewka, G., Schlipf, J. (eds.) LPNMR 2007. LNCS, vol. 4483, pp. 3–17. Springer, Heidelberg (2007)
12. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: Kowalski, R., Bowen, K. (eds.) Proceedings of International Logic Programming Conference and Symposium, pp. 1070–1080. MIT Press, Cambridge (1988)
13. Gelfond, M., Lifschitz, V.: Logic programs with classical negation. In: Warren, D., Szeredi, P. (eds.) Proceedings of International Conference on Logic Programming (ICLP), pp. 579–597 (1990)
14. Gelfond, M.: On stratified autoepistemic theories. In: Proceedings of National Conference on Artificial Intelligence (AAAI), pp. 207–211 (1987)
15. Heyting, A.: Die formalen Regeln der intuitionistischen Logik. In: Sitzungsberichte der Preussischen Akademie der Wissenschaften. Physikalisch-mathematische Klasse, pp. 42–56 (1930)
16. Lee, J., Lifschitz, V., Palla, R.: A reductive semantics for counting and choice in answer set programming. In: Proceedings of the AAAI Conference on Artificial Intelligence (AAAI), pp. 472–479 (2008)
17. Lee, J.: A model-theoretic counterpart of loop formulas. In: Proceedings of International Joint Conference on Artificial Intelligence (IJCAI), pp. 503–508 (2005)
18. Lifschitz, V., Tang, L.R., Turner, H.: Nested expressions in logic programs. Annals of Mathematics and Artificial Intelligence 25, 369–389 (1999)
19. Lifschitz, V., Pearce, D., Valverde, A.: Strongly equivalent logic programs. ACM Transactions on Computational Logic 2, 526–541 (2001)
20. Lifschitz, V.: On the declarative semantics of logic programs with negation. In: Minker, J. (ed.) Foundations of Deductive Databases and Logic Programming, pp. 177–192. Morgan Kaufmann, San Mateo (1988)
21. Lin, F., Reiter, R.: Rules as actions: A situation calculus semantics for logic programs. Journal of Logic Programming 31, 299–330 (1997)
22. Lin, F., Zhao, Y.: ASSAT: Computing answer sets of a logic program by SAT solvers. In: Proceedings of National Conference on Artificial Intelligence (AAAI), pp. 112–117 (2002)

23. Lin, F., Zhao, Y.: ASSAT: Computing answer sets of a logic program by SAT solvers. Artificial Intelligence 157, 115–137 (2004), http://www.cs.ust.hk/faculty/flin/papers/assat-aij-revised.pdf
24. Lin, F., Zhou, Y.: From answer set logic programming to circumscription via logic of GK. In: Proceedings of International Joint Conference on Artificial Intelligence, IJCAI (2007)
25. Lin, F.: A Study of Nonmonotonic Reasoning. PhD thesis, Stanford University (1991)
26. Marek, V., Truszczyński, M.: Stable semantics for logic programs and default theories. In: Proc. North American Conf. on Logic Programming, pp. 243–256 (1989)
27. McCarthy, J., Hayes, P.: Some philosophical problems from the standpoint of artificial intelligence. In: Meltzer, B., Michie, D. (eds.) Machine Intelligence, vol. 4, pp. 463–502. Edinburgh University Press, Edinburgh (1969)
28. McCarthy, J.: Circumscription—a form of non-monotonic reasoning. Artificial Intelligence 13, 27–39, 171–172 (1980)
29. McCarthy, J.: Applications of circumscription to formalizing common sense knowledge. Artificial Intelligence 26(3), 89–116 (1986)
30. McDermott, D., Doyle, J.: Nonmonotonic logic I. Artificial Intelligence 13, 41–72 (1980)
31. McDermott, D.: Nonmonotonic logic II: Nonmonotonic modal theories. Journal of ACM 29(1), 33–57 (1982)
32. Moore, R.: Semantical considerations on nonmonotonic logic. Artificial Intelligence 25(1), 75–94 (1985)
33. Niemelä, I., Simons, P.: Extending the Smodels system with cardinality and weight constraints. In: Minker, J. (ed.) Logic-Based Artificial Intelligence, pp. 491–521. Kluwer, Dordrecht (2000)
34. Pearce, D., Tompits, H., Woltran, S.: Encodings for equilibrium logic and logic programs with nested expressions. In: Brazdil, P.B., Jorge, A.M. (eds.) EPIA 2001. LNCS, vol. 2258, pp. 306–320. Springer, Heidelberg (2001)
35. Pearce, D.: A new logical characterization of stable models and answer sets. In: Dix, J., Pereira, L., Przymusinski, T. (eds.) NMELP 1996. LNCS (LNAI), vol. 1216, pp. 57–70. Springer, Heidelberg (1997)
36. Reiter, R.: A logic for default reasoning. Artificial Intelligence 13, 81–132 (1980)
37. Reiter, R.: Circumscription implies predicate completion (sometimes). In: Proceedings of International Joint Conference on Artificial Intelligence (IJCAI), pp. 418–420 (1982)
38. Reiter, R.: Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems. MIT Press, Cambridge (2001)
39. Saccá, D., Zaniolo, C.: Stable models and non-determinism in logic programs with negation. In: Proceedings of ACM Symposium on Principles of Database Systems (PODS), pp. 205–217 (1990)
40. van Emden, M., Clark, K.: The logic of two-person games. In: Micro-PROLOG: Programming in Logic, pp. 320–340. Prentice-Hall, Englewood Cliffs (1984)
41. van Emden, M., Kowalski, R.: The semantics of predicate logic as a programming language. Journal of ACM 23(4), 733–742 (1976)
42. Van Gelder, A., Ross, K.A., Schlipf, J.S.: 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, Austin, Texas, March 21-23, 1988, pp. 221–230. ACM Press, New York (1988)

43. Van Gelder, A., Ross, K., Schlipf, J.: The well-founded semantics for general logic programs. Journal of ACM 38(3), 620–650 (1991)
44. Van Gelder, A.: Negation as failure using tight derivations for general logic programs. In: Minker, J. (ed.) Foundations of Deductive Databases and Logic Programming, pp. 149–176. Morgan Kaufmann, San Mateo (1988)
45. Wallace, M.: Tight, consistent and computable completions for unrestricted logic programs. Journal of Logic Programming 15, 243–273 (1993)