A New Perspective on Stable Models*

Paolo Ferraris[1], **Joohyung Lee**[2] **and Vladimir Lifschitz**[1]

[1]Department of Computer Sciences
University of Texas at Austin
1 University Station C0500
Austin, TX 78705
{otto,vl}@cs.utexas.edu

[2]Dept. of Computer Science and Engineering
Arizona State University
South Mill Avenue 574
Tempe, AZ 85281
joolee@asu.edu

Abstract

The definition of a stable model has provided a declarative semantics for Prolog programs with negation as failure and has led to the development of answer set programming. In this paper we propose a new definition of that concept, which covers many constructs used in answer set programming (including disjunctive rules, choice rules and conditional literals) and, unlike the original definition, refers neither to grounding nor to fixpoints. Rather, it is based on a syntactic transformation, which turns a logic program into a formula of second-order logic that is similar to the formula familiar from the definition of circumscription.

1 Introduction

Two widely used definitions of the semantics of logic programs—in terms of program completion [Clark, 1978] and in terms of stable models [Gelfond and Lifschitz, 1988]—look very different from each other. The former treats a logic program as shorthand for its completion, which is a first-order formula. For instance, the program

$$\begin{array}{l} p(a), \\ q(b), \\ r(x) \leftarrow p(x), \textit{not } q(x) \end{array} \tag{1}$$

is shorthand for

$$\begin{array}{r} \forall x(p(x) \leftrightarrow x = a) \wedge \forall x(q(x) \leftrightarrow x = b) \\ \wedge \forall x(r(x) \leftrightarrow (p(x) \wedge \neg q(x))). \end{array} \tag{2}$$

On the other hand, according to the stable model semantics, (1) is shorthand for the set of the ground instances of its rules:

$$\begin{array}{l} p(a), \\ q(b), \\ r(a) \leftarrow p(a), \textit{not } q(a), \\ r(b) \leftarrow p(b), \textit{not } q(b). \end{array} \tag{3}$$

The definition of a stable model describes a fixpoint construction that determines which sets of atomic formulas from (3) are considered "stable models"; it turns out that the only stable model of (3) is

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

In spite of this difference between the two definitions, there is often a close relationship between the completion of a program and its stable models. For instance, in every model of (2) (in the sense of first-order logic) that satisfies the unique names assumption $a \neq b$, the elements of set (4) are true, and all other ground atoms are false.

Practical needs of answer set programming (ASP) have led to the invention of several declarative programming constructs that are not used in Prolog. Clark's completion semantics is not applicable to these constructs, at least directly. For instance, the last rule of the program

$$\begin{array}{l} p(a), \\ p(b), \\ \{q(x)\ :\ p(x)\} \end{array} \tag{5}$$

is a "choice rule" containing a "conditional literal" [Simons *et al.*, 2002]. Intuitively, this rule says: for any x such that $p(x)$, choose arbitrarily whether or not to include $q(x)$ in the stable model. The semantics of programs with choice rules, like the original stable model semantics, is defined in terms of grounding and a fixpoint condition. For instance, grounding turns the last line of (5) into the ground choice rule

$$\{q(a), q(b)\}.$$

As it turns out, program (5) has 4 stable models:

$$\begin{array}{l} \{p(a),\ p(b)\}, \\ \{p(a),\ p(b),\ q(a)\}, \\ \{p(a),\ p(b),\ q(b)\}, \\ \{p(a),\ p(b),\ q(a),\ q(b)\}. \end{array} \tag{6}$$

In this paper we propose a new definition of a stable model, which covers many constructs used in ASP (including disjunctive rules, choice rules, cardinality constraints and conditional literals) and refers neither to grounding nor to fixpoints. Rather, like the definition of program completion, the new definition of a stable model is based on a transformation that turns the given logic program into a formula of classical logic. To be precise, the result of this transformation is a *second-order* formula, which looks similar to the formula familiar from the definition of circumscription [McCarthy, 1980; 1986] in the form adopted in [Lifschitz, 1994].

*The first and third authors were partially supported by the National Science Foundation under Grant IIS-0412907. The second author was partially supported by DTO AQUAINT.

The new definition and examples of its use are discussed in Section 2 below. In Section 3 we relate our definition to a theorem from [Lin, 1991], to the encoding of propositional logic programs by quantified Boolean formulas due to Pearce, Tompits and Woltran [2001],[1] and to recent research on first-order equilibrium logic [Pearce and Valverde, 2004; 2005]. A theorem about strong equivalence, illustrating the nature of the ongoing work on reformulating the theory of stable models on the basis of the new definition, is stated in Section 4. Finally, in Section 5 we propose a way to generalize the concept of program completion that is similar to the new definition of a stable model.

Our treatment of stable models may be of interest for three reasons. First, it provides a new perspective on the place of stable models within the field of nonmonotonic reasoning. We can distinguish between "translational" nonmonotonic formalisms, such as program completion and circumscription, and "fixpoint" formalisms—default logic [Reiter, 1980][2] and autoepistemic logic [Moore, 1985]. In the past, stable models were seen as part of the "fixpoint tradition." In fact, the invention of stable models was an outgrowth of earlier work on the relationship between logic programming and autoepistemic logic [Gelfond, 1987]; the first journal paper on answer sets [Gelfond and Lifschitz, 1991] emphasized their relation to default logic. The remarkable similarity between the new definition of a stable model and the definition of circumscription is rather curious from this point of view.

Second, we expect that the new definition of stable models will provide a unified framework for useful answer set programming constructs defined and implemented by several different research groups, such as choice rules, cardinality constraints and conditional literals (Helsinki University of Technology), disjunctive rules and aggregates [Faber *et al.*, 2004] (Vienna University of Technology and University of Calabria), and ASET-Prolog constructs [Gelfond, 2002, Section 5.2] (Texas Tech University).

Finally, we hope that this definition of a stable model will serve as a basis for a new approach to proving program correctness in ASP, which will be more straightforward than the one based on grounding and fixpoint definitions [Ferraris and Lifschitz, 2005a, Sections 3.3–3.5, 3.7]. These correctness proofs will use equivalent transformations of formulas of classical logic as the main tool.

2 Definition and Examples

2.1 Logic Programs as First-Order Formulas

The concept of a stable model will be defined here for first-order sentences (formulas without free variables); logic programs are viewed in this paper as alternative notation for first-order sentences of special kinds.[3]

To rewrite a "traditional" program, such as (1), as a first-order sentence, we

- replace every comma by $\wedge$ and every *not* by $\neg$,
- turn every rule *Head* $\leftarrow$ *Body* into a formula by rewriting it as the implication *Body* $\rightarrow$ *Head*, and
- form the conjunction of the universal closures of these formulas.

For instance, we think of (1) as alternative notation for the sentence

$$p(a) \wedge q(b) \wedge \forall x((p(x) \wedge \neg q(x)) \rightarrow r(x)). \qquad (7)$$

We are going to treat $\neg F$ as shorthand for $F \rightarrow \bot$, so that the last conjunctive term can be further expanded into

$$\forall x((p(x) \wedge (q(x) \rightarrow \bot)) \rightarrow r(x)).$$

In the spirit of [Ferraris and Lifschitz, 2005b], (5) is understood as

$$p(a) \wedge p(b) \wedge \forall x(p(x) \rightarrow (q(x) \vee \neg q(x))). \qquad (8)$$

Since the last conjunctive term is logically valid, the class of models of formula (8) would not change if we dropped that term; but the class of its *stable* models, as defined below, would be affected. In this sense, the last conjunctive term is essential.

Finally, here is an example of turning a cardinality constraint [Simons *et al.*, 2002] into a first-order formula. The rule

$$p \leftarrow 10\,\{q(x)\,:\,r(x)\}\,20$$

corresponds to the sentence

$$(\exists_{10}x(q(x) \wedge r(x)) \wedge \neg\exists_{21}x(q(x) \wedge r(x))) \rightarrow p, \qquad (9)$$

where $\exists_n x F(x)$ is understood as an abbreviation for

$$\exists x_1 \cdots x_n \left(\bigwedge_{1\le i\le n} F(x_i) \wedge \bigwedge_{1\le i<j\le n} x_i \neq x_j \right).$$

2.2 Review of Circumscription

Since the new definition of a stable model looks similar to the definition of circumscription, we will begin with a brief review of the latter, for the special case when all predicate constants occurring in the formula are circumscribed in parallel [Lifschitz, 1994, Section 7.1].

Both definitions use the following notation. If p and q are predicate constants of the same arity then $p = q$ stands for the formula

$$\forall \mathbf{x}(p(\mathbf{x}) \leftrightarrow q(\mathbf{x})),$$

[1]The fact that circumscription is related to program completion has been known for a long time [Reiter, 1982; Lifschitz, 1985]. The relationship between circumscription and the Pearce-Tompits-Woltran transformation is discussed in [Ferraris *et al.*, 2006, Appendix B].

[2]The translational definition of default logic, proposed in [Lifschitz, 1990], is rather complicated: it uses *third-order* variables.

[3]In the propositional case, this approach to the syntax of ASP is not new. The possibility of interpreting choice rules and weight constraints in terms of nested conjunctions, disjunctions and negations was demonstrated in [Ferraris and Lifschitz, 2005b, Section 4.1]. General aggregates can be described in terms of nested implications [Ferraris, 2005, Section 4]. Including second ("strong," "classical," or "true") negation without introducing an additional connective is discussed in [Ferraris and Lifschitz, 2005a, Section 3.9].

and $p \leq q$ stands for

$$\forall \mathbf{x}(p(\mathbf{x}) \rightarrow q(\mathbf{x})),$$

where **x** is a tuple of distinct object variables. If **p** and **q** are tuples $p_1, \ldots, p_n$ and $q_1, \ldots, q_n$ of predicate constants then $\mathbf{p} = \mathbf{q}$ stands for the conjunction

$$p_1 = q_1 \wedge \cdots \wedge p_n = q_n,$$

and $\mathbf{p} \leq \mathbf{q}$ for

$$p_1 \leq q_1 \wedge \cdots \wedge p_n \leq q_n.$$

Finally, $\mathbf{p} < \mathbf{q}$ is an abbreviation for $\mathbf{p} \leq \mathbf{q} \wedge \neg(\mathbf{p} = \mathbf{q})$.

In second-order logic, we will apply the same notation to tuples of predicate variables.

Given a first-order sentence F, by CIRC$[F]$ we denote the second-order sentence

$$F \wedge \neg \exists \mathbf{u}((\mathbf{u} < \mathbf{p}) \wedge F(\mathbf{u})),$$

where **p** stands for the list of all predicate constants occurring in F, **u** is a list of distinct predicate variables of the same length, and $F(\mathbf{u})$ is the formula obtained from F by substituting the variables **u** for the constants **p**. Intuitively, the second conjunctive term of CIRC$[F]$ expresses that the extents of the predicates **p** are minimal subject to condition F.

For example, if F is

$$p(a) \wedge \forall x(p(x) \rightarrow q(x)) \tag{10}$$

then CIRC$[F]$ is

$$\begin{array}{l} p(a) \wedge \forall x(p(x) \rightarrow q(x)) \\ \quad \wedge \neg \exists uv(((u,v) < (p,q)) \wedge u(a) \wedge \forall x(u(x) \rightarrow v(x))). \end{array} \tag{11}$$

Using methods for eliminating second-order quantifiers discussed in [Lifschitz, 1994] and [Doherty *et al.*, 1997], we can simplify (11) and convert it into

$$\forall x(p(x) \leftrightarrow x = a) \wedge \forall x(q(x) \leftrightarrow x = a). \tag{12}$$

There are cases when CIRC$[F]$ is not equivalent to any first-order formula, as, for instance, when F is

$$p(a) \wedge \forall x(p(x) \rightarrow p(f(x))). \tag{13}$$

In this example, a model of CIRC$[F]$ is any interpretation that represents p as the set of the values of the terms $a, f(a), f(f(a)), \ldots$.

2.3 Stable Models

Given a first-order sentence F, by SM$[F]$ we denote the second-order sentence

$$F \wedge \neg \exists \mathbf{u}((\mathbf{u} < \mathbf{p}) \wedge F^*(\mathbf{u})),$$

where **p** stands for the list of all predicate constants $p_1, \ldots, p_n$ occurring in F, **u** is a list of n distinct predicate variables $u_1, \ldots, u_n$, and $F^*(\mathbf{u})$ is defined recursively:

- $p_i(t_1, \ldots, t_m)^* = u_i(t_1, \ldots, t_m)$;
- $(t_1 = t_2)^* = (t_1 = t_2)$;
- $\bot^* = \bot$;
- $(F \odot G)^* = F^* \odot G^*$, where $\odot \in \{\wedge, \vee\}$;
- $(F \rightarrow G)^* = (F^* \rightarrow G^*) \wedge (F \rightarrow G)$;
- $(QxF)^* = QxF^*$, where $Q \in \{\forall, \exists\}$.

Note that the operator $F \mapsto F^*(\mathbf{u})$ replaces each predicate constant with the corresponding predicate variable, and that it commutes with all propositional connectives except implication and with both quantifiers. If, in the definition of this operator, we drop the second conjunctive term in the clause for implication, then $F^*(\mathbf{u})$ will turn into the formula $F(\mathbf{u})$ referred to in the definition of circumscription. That conjunctive term is the only difference between the definitions of CIRC and SM.

A model of F is *stable* if it satisfies SM$[F]$.

Example 1 If F is (10) then $F^*(u, v)$ is

$$u(a) \wedge \forall x((u(x) \rightarrow v(x)) \wedge (p(x) \rightarrow q(x)))$$

and SM$[F]$ is

$$\begin{array}{l} p(a) \wedge \forall x(p(x) \rightarrow q(x)) \\ \quad \wedge \neg \exists uv(((u,v) < (p,q)) \wedge u(a) \wedge \forall x((u(x) \rightarrow v(x)) \\ \hfill \wedge (p(x) \rightarrow q(x)))). \end{array}$$

It is clear that this formula is equivalent to (11), and consequently to (12).

In logic programming notation, (10) can be written as

$$\begin{array}{l} p(a), \\ q(x) \leftarrow p(x). \end{array}$$

The completion of this program

$$\forall x(p(x) \leftrightarrow x = a) \wedge \forall x(q(x) \leftrightarrow p(x))$$

is equivalent to (12) as well. In this example, all three transformations—SM, CIRC and completion—produce essentially the same result.

Example 2 If F is (13) then, as in the previous example, it is clear that SM$[F]$ is equivalent to CIRC$[F]$. Consequently, the stable models of (13) can be characterized by the condition stated at the end of the previous section: p is represented by the set of the values of the terms $a, f(a), f(f(a)), \ldots$.

In logic programming notation, (13) can be written as

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

The completion of this program

$$\forall x(p(x) \leftrightarrow (x = a \vee \exists y(x = f(y) \wedge p(y)))$$

is weaker than SM$[F]$: some (non-Herbrand[4]) models of the completion of (14) are not stable.

It is easy to see that the operator SM produces essentially the same result as CIRC whenever it is applied to a formula corresponding to a set of Horn rules, as in the examples

[4]An *Herbrand interpretation* of a signature σ containing at least one object constant is an interpretation such that (i) its universe is the set of all ground terms of σ, and (ii) every ground term represents itself. Clearly, an Herbrand interpretation can be characterized by the set of ground atoms to which it assigns the value *true*.

above.[5] But if negation in the bodies of rules is allowed then this may be no longer the case, as we will see Section 2.4.

What we can say, on the other hand, about this more general case is that stable *Herbrand* models of the corresponding formula exactly correspond to the stable models of the program in the sense of the original definition from [Gelfond and Lifschitz, 1988]:

Proposition 1 *Let σ be a signature containing at least one object constant, and Π a finite set of rules of the form*

$$A_0 \leftarrow A_1, \ldots, A_m, \textit{not } A_{m+1}, \ldots, \textit{not } A_n, \qquad (15)$$

where $A_0, \ldots, A_n$ are atomic formulas of σ not containing equality. For any set X of ground terms of σ, the following conditions are equivalent:

- *X is a stable model of Π in the sense of the 1988 definition;*
- *the Herbrand interpretation of σ that makes the elements of X true and all other ground atoms false is a stable model of the formula corresponding to Π.*

This theorem shows that the new definition of a stable model, restricted to the "traditional" syntax, is a generalization of the 1988 definition to non-Herbrand models. In Section 3.1 we will see that our definition generalizes also the definition proposed in [Ferraris, 2005] and used in [Ferraris and Lifschitz, 2005a; Ferraris *et al.*, 2006].

2.4 Further Examples

Proposition 2 below allows us to simplify the application of the operator SM to formulas containing negation. In its statement, $\mathbf{p}$ is the list of predicate constants occurring in F, and $\mathbf{u}$ is a list of distinct predicate variables of the same length as $\mathbf{p}$.

Proposition 2 *If a formula F begins with $\neg$ then the formula*

$$\mathbf{u} \leq \mathbf{p} \rightarrow (F^*(\mathbf{u}) \leftrightarrow F)$$

is logically valid.

Example 3 Let F be formula (7), corresponding to logic program (1). Then SM$[F]$ is

$$\begin{array}{l} p(a) \wedge q(b) \wedge \forall x((p(x) \wedge \neg q(x)) \rightarrow r(x)) \\ \quad \wedge \neg \exists uvw(((u,v,w) < (p,q,r)) \wedge u(a) \wedge v(b) \\ \qquad\qquad \wedge \forall x(((u(x) \wedge (\neg q(x))^*) \rightarrow w(x)) \\ \qquad\qquad\quad \wedge ((p(x) \wedge \neg q(x)) \rightarrow r(x)))). \end{array}$$

It is clear that the implication in the last line can be dropped. Furthermore, since the subformula $(u, v, w) < (p, q, r)$ contains the conjunctive term $v \leq q$, from Proposition 2 we can conclude that $(\neg q(x))^*$ can be equivalently replaced here by $\neg q(x)$. Consequently, SM$[F]$ can be rewritten as

$$\begin{array}{l} p(a) \wedge q(b) \wedge \forall x((p(x) \wedge \neg q(x)) \rightarrow r(x)) \\ \quad \wedge \neg \exists uvw(((u,v,w) < (p,q,r)) \wedge u(a) \wedge v(b) \\ \qquad \wedge \forall x((u(x) \wedge \neg q(x)) \rightarrow w(x))). \end{array}$$

Using the methods for eliminating second-order quantifiers described at the end of [Lifschitz, 1994, Section 3.3], we can convert this formula into the completion (2) of program (1). We conclude that in this case the stable models of the program are identical to the models of its completion.

We can further conclude that there is a unique Herbrand stable model in this case, and that it corresponds to the set (4) of ground atoms. This fact follows also from Proposition 1.

Example 4 If F is formula (8), corresponding to logic program (5), then a similar calculation converts SM$[F]$ into

$$\begin{array}{l} p(a) \wedge p(b) \wedge \forall x(p(x) \rightarrow (q(x) \vee \neg q(x))) \\ \wedge \neg \exists uv(((u,v) < (p,q)) \wedge u(a) \wedge u(b) \\ \qquad \wedge \forall x(u(x) \rightarrow (v(x) \vee \neg q(x)))). \end{array}$$

After the elimination of second-order quantifiers, this formula becomes

$$\begin{array}{l} \forall x(p(x) \leftrightarrow (x = a \vee x = b)) \\ \wedge \forall x(q(x) \rightarrow (x = a \vee x = b)). \end{array}$$

The stable models of (5) can be characterized as the interpretations that (i) represent p by the set of values of a and b, and (ii) represent q by a subset of that set. Consequently, (8) has 4 Herbrand stable models, and they correspond to sets (6).

We call a formula *negative* if every occurrence of every predicate constant in this formula belongs to the antecedent of an implication. Clearly any formula of the form $\neg F$ is negative, because this expression is shorthand for $F \rightarrow \bot$ (Section 2.1). Proposition 2 can be generalized to arbitrary negative formulas.

3 Relation to Earlier Work

3.1 Propositional Case

In the propositional case, the operator SM turns into the encoding of formulas of equilibrium logic by quantified Boolean formulas proposed in [Pearce *et al.*, 2001] and reviewed in [Ferraris *et al.*, 2006, Appendix B]. In view of the Pearce-Tompits-Woltran theorem, as restated in that review, it follows that in the propositional case our definition of a stable model is equivalent to the definition of a stable model (answer set) proposed in [Ferraris, 2005] and reviewed in [Ferraris *et al.*, 2006, Appendix A].

3.2 Lin's Transformation

Theorem 5 from [Lin, 1991] relates stable models of "traditional programs" (as in Proposition 1 above) to circumscription. It involves a syntactic transformation that can be described as a sequence of three steps. First, each rule is turned into a formula that may contain new predicate constants—"doubles" p' of the predicate constants p occurring in the rule. Second, the new predicate constants are circumscribed in parallel. Third, the result is conjoined with the equivalences $p' = p$. We will show that this idea is applicable to arbitrary first-order sentences, and that the result of this transformation is closely related to the operator SM.

To do this, we need parallel circumscription of a slightly more general kind than defined in Section 2.2. In the definition of circumscription, there is no need to assume that $\mathbf{p}$ stands for the list of *all* predicate constants occurring in F; $\mathbf{p}$ may include only some of these constants. The result of

[5]This assertion remains true if we allow the heads of rules to be disjunctions of atomic formulas.

circumscribing the predicate constants $\mathbf{p}$ in a first-order sentence F will be denoted by CIRC$[F; \mathbf{p}]$. For instance, if F is (10) then CIRC$[F; q]$ is

$$\begin{array}{l} p(a) \wedge \forall x(p(x) \rightarrow q(x)) \\ \quad \wedge \neg \exists v((v < q) \wedge p(a) \wedge \forall x(p(x) \rightarrow v(x))), \end{array}$$

which is equivalent to

$$p(a) \wedge \forall x(p(x) \leftrightarrow q(x)).$$

Let F be a first-order sentence, and let $\mathbf{p}$ be the list of all predicate constants occurring in F. Take a list $\mathbf{p}'$ of distinct predicate constants that do not occur in F, of the same length as $\mathbf{p}$. By L$[F; \mathbf{p}']$ we denote the formula

$$\text{CIRC}[F^*(\mathbf{p}'); \mathbf{p}'] \wedge (\mathbf{p}' = \mathbf{p}).$$

This formula turns out to be equivalent to SM$[F]$ conjoined with explicit definitions of the new predicate constants $\mathbf{p}'$:

Proposition 3 L$[F; \mathbf{p}']$ *is equivalent to* SM$[F] \wedge (\mathbf{p}' = \mathbf{p})$.

This is immediate from the definitions of L and SM, using the fact that $F^*(\mathbf{p})$ is equivalent to F.

It follows that SM$[F]$ is equivalent to L$[F; \mathbf{p}']$ with the predicate constants $\mathbf{p}'$ replaced by existentially quantified predicate variables:

Corollary 1 SM$[F]$ *is equivalent to* $\exists \mathbf{u}\,$L$[F; \mathbf{u}]$.

3.3 Equilibrium Logic

The definition of first-order equilibrium logic below is similar to the one proposed in [Pearce and Valverde, 2005, Section 7], except that ground terms are not identified here with their values; as a result, different ground terms are allowed to have the same value. Our definition describes essentially Kripke models with two worlds ("here" and "there") that have the same universe, interpret all function constants in the same way, and satisfy the minimality condition introduced in [Pearce, 1997].

If I is an interpretation of a signature σ (in the sense of classical logic) then by σ^I we denote the extension of σ obtained by adding pairwise distinct symbols ξ^*, called *names*, for all elements ξ of the universe of I as object constants. We will identify I with its extension to σ^I defined by $I(\xi^*) = \xi$. The value that I assigns to a ground term t of signature σ^I will be denoted by t^I.

By σ_f we denote the part of σ consisting of its function constants (including object constants, which are viewed as function constants of arity 0). We will represent an interpretation I of σ as the pair $\langle I|_{\sigma^f}, I' \rangle$, where I' is the set of all atomic formulas, formed using predicate constants from σ and names ξ^*, which are satisfied by I.

An *HT-interpretation* of σ is a triple $\langle I^f, I^h, I^t \rangle$, where

- I^f is an interpretation of σ_f, and
- I^h, I^t are sets of atomic formulas formed using predicate constants from σ and object constants ξ^* for arbitrary elements ξ of the universe of I^f, such that $I^h \subseteq I^t$.

The *satisfaction* relation between an HT-interpretation $I = \langle I^f, I^h, I^t \rangle$ and a sentence F of the signature $\sigma^{\langle I^f, I^h \rangle}$ is defined recursively:

- $I \models p(t_1, \dots, t_n)$ if $p((t_1^{I^f})^*, \dots, (t_n^{I^f})^*) \in I^h$;
- $I \models t_1 = t_2$ if $t_1^{I^f} = t_2^{I^f}$;
- $I \not\models \bot$;
- $I \models F \wedge G$ if $I \models F$ and $I \models G$; similarly for $\vee$;
- $I \models F \rightarrow G$ if
  - (i) $I \not\models F$ or $I \models G$, and
  - (ii) $\langle I, I^t \rangle \models F \rightarrow G$;
- $I \models \forall x F(x)$ if, for each ξ from the universe of I^f, $I \models F(\xi^*)$; similarly for $\exists$.

(In (ii) we understand satisfaction as in classical logic.)

An HT-interpretation of the form $\langle I, J, J \rangle$ is an *equilibrium model* of F if

- $\langle I, J, J \rangle \models F$, and
- for any proper subset J' of J, $\langle I, J', J \rangle \not\models F$.

This definition provides a precise model-theoretic counterpart of the operator SM:

Proposition 4 *An interpretation $\langle I, J \rangle$ is a stable model of a sentence F iff $\langle I, J, J \rangle$ is an equilibrium model of F.*

4 Strong Equivalence

To turn the definition of a stable model proposed in this paper into a tool that can help us in the design of provably correct ASP programs, we need to find appropriate counterparts of the theorems that are used in correctness proofs today.[6] The "traditional" theorems about stable models will roughly correspond to the special cases of these new theorems in which the formulas involved are propositional combinations of ground atoms, perhaps of a special syntactic form, and our attention is restricted to Herbrand models.

To give an example illustrating this general point, we state here a counterpart of the characterization of strong equivalence [Lifschitz *et al.*, 2001] due to Fangzhen Lin [2002].

About first-order sentences F and G we say that F is *strongly equivalent* to G if, for every sentence H (possibly of a larger signature), $F \wedge H$ has the same stable models as $G \wedge H$ (or, to put it differently, if, for every H, SM$[F \wedge H]$ is equivalent to SM$[G \wedge H]$). In the following theorem, $\mathbf{p}$ is the list of predicate constants occurring in at least one of the sentences F, G, and $\mathbf{p}'$ is a list of new, distinct predicate constants of the same length as $\mathbf{p}$.

Proposition 5 *F is strongly equivalent to G iff the formula*

$$\mathbf{p}' \leq \mathbf{p} \rightarrow (F^*(\mathbf{p}') \leftrightarrow G^*(\mathbf{p}'))$$

is logically valid.

Using this theorem we can show, for instance, that $\neg\forall x F(x)$ is strongly equivalent to $\exists x \neg F(x)$. (This is a predicate logic counterpart of the fact that $\neg(F \wedge G)$ is strongly equivalent to $\neg F \vee \neg G$.) Indeed, in view of Proposition 2, the implications

$$\begin{array}{l} \mathbf{p}' \leq \mathbf{p} \rightarrow ((\neg\forall x F(x))^* \leftrightarrow \neg\forall x F(x)), \\ \mathbf{p}' \leq \mathbf{p} \rightarrow ((\exists x \neg F(x))^* \leftrightarrow \exists x \neg F(x)); \end{array}$$

[6] See, for instance, [Ferraris and Lifschitz, 2005a, Sections 2.1–2.4, 2.6–3.1].

are logically valid; the right-hand sides of the two equivalences are classically equivalent to each other.

For our proof of the "only if" part of Proposition 5 it is not essential that the definition of strong equivalence allows the signature of H to be larger than the signature of F and G. It follows that F is strongly equivalent to G whenever $F \wedge H$ has the same stable models as $G \wedge H$ for all sentences H of the same signature as F and G.

Relations between logic programs with variables, somewhat similar to strong equivalence as defined above but more limited in scope, are discussed in [Pearce and Valverde, 2005, Section 7] and [Eiter *et al.*, 2006].

5 A New Perspective on Program Completion

5.1 Pointwise Stable Models

As observed in [Lee and Lin, 2006], program completion is similar in some ways to the concept of pointwise circumscription—the modification of McCarthy's original definition that was proposed in [Lifschitz, 1987]. According to either definition, circumscribing a predicate constant p makes the extent of p "minimal," but minimality is understood in different versions differently. According to the original definition of circumscription, to make the extent of a predicate smaller means to replace it by a proper subset. In the pointwise version, to make the extent of a predicate smaller means to decrement it by a single point. The pointwise minimality condition is, generally, weaker than minimality according to McCarthy; similarly, program completion is generally weaker than the stability condition.

In this section, we define a weakened, "pointwise" version of the operator SM that can be viewed as a generalization of program completion to arbitrary first-order formulas.

If p and q are predicate constants of the same arity k then $p \overset{1}{<} q$ stands for the formula

$$\exists \mathbf{x}(q(\mathbf{x}) \wedge \forall \mathbf{y}(p(\mathbf{y}) \leftrightarrow (q(\mathbf{y}) \wedge \mathbf{x} \neq \mathbf{y}))),$$

where $\mathbf{x}$, $\mathbf{y}$ are disjoint tuples of distinct object variables $x_1, \dots, x_k, y_1, \dots, y_k$, and $\mathbf{x} \neq \mathbf{y}$ is shorthand for

$$\neg(x_1 = y_1 \wedge \cdots \wedge x_k = y_k).$$

This formula expresses that the extent of p can be obtained from the extent of q by removing one element. If $\mathbf{p}$ and $\mathbf{q}$ are tuples $p_1, \dots, p_n$ and $q_1, \dots, q_n$ of predicate constants then $\mathbf{p} \overset{1}{<} \mathbf{q}$ stands for the disjunction

$$\bigvee_{1 \le i \le n} \left((p_i \overset{1}{<} q_i) \wedge \bigwedge_{1 \le j \le n,\ j \neq i} (p_j = q_j) \right),$$

and similarly for tuples of predicate variables.

Given a first-order sentence F, by PSM$[F]$ we denote the second-order sentence

$$F \wedge \neg \exists \mathbf{u}((\mathbf{u} \overset{1}{<} \mathbf{p}) \wedge F^*(\mathbf{u})),$$

where $\mathbf{p}$, $\mathbf{u}$ and $F^*(\mathbf{u})$ are as in the definition of SM (Section 2.3). A model of F is *pointwise stable* if it satisfies PSM$[F]$. Clearly, every stable model is pointwise stable.

Unlike SM$[F]$, the weaker formula PSM$[F]$ can be always rewritten without second-order quantifiers:

Proposition 6 *Formula* PSM$[F]$ *is equivalent to*

$$F \wedge \bigwedge_{1 \le i \le n} \neg \exists \mathbf{x}^i (p_i(\mathbf{x}^i) \wedge G_i(\mathbf{x}^i)),$$

where $G_i(\mathbf{x}^i)$ *stands for*

$$F^*(p_1, \dots, p_{i-1}, \lambda \mathbf{y}^i (p_i(\mathbf{y}^i) \wedge \mathbf{y}^i \neq \mathbf{x}^i), p_{i+1}, \dots, p_n)$$

and $\mathbf{x}^i$, $\mathbf{y}^i$ *are disjoint tuples of distinct variables.*[7]

For instance, if F is $p(a) \wedge p(b)$ then $F^*(u)$ is $u(a) \wedge u(b)$, so that $F^*(\lambda y(p(y) \wedge y \neq x))$ is

$$p(a) \wedge a \neq x \wedge p(b) \wedge b \neq x,$$

and PSM$[F]$ is

$$p(a) \wedge p(b) \wedge \neg \exists x(p(x) \wedge p(a) \wedge a \neq x \wedge p(b) \wedge b \neq x).$$

This formula can be simplified:

$$p(a) \wedge p(b) \wedge \neg \exists x(p(x) \wedge a \neq x \wedge b \neq x).$$

In this example, PSM$[F]$ is obviously equivalent to the completion of F:

$$\forall x(p(x) \leftrightarrow (x = a \vee x = b)).$$

This fact is an instance of the general theorem stated below.

5.2 Relation to Program Completion

Lloyd and Topor [1984] noted that the process of completing a program can be extended in an obvious way to rules of a more general form than allowed in [Clark, 1978]. It is essential that the head of a rule be an atom, but the body can be an arbitrary first-order formula.

Proposition 7 below refers to completion in this more general sense, but it does introduce a restriction on the syntactic form of the bodies of rules. The rules we consider in this section have the form

$$p_0(\mathbf{t}^0) \leftarrow p_1(\mathbf{t}^1) \wedge \cdots \wedge p_m(\mathbf{t}^m) \wedge N, \qquad (16)$$

where $\mathbf{t}^i$ are tuples of terms and N is a negative formula (see Section 2.4). For instance, every rule of form (15) has also form (16): take N to be $\neg A_{m+1} \wedge \cdots \wedge \neg A_n$.

A rule of form (16) is *acyclic* if for each $i = 1, \dots, m$ such that p_i is p_0, the formula

$$N \rightarrow \mathbf{t}^0 \neq \mathbf{t}^i$$

is logically valid. For instance, each of the rules (1) is obviously acyclic—its body doesn't contain the predicate constant occurring in the head. Any rule of form (16) can be made acyclic by a strongly equivalent transformation: conjoin the body with the formulas $\mathbf{t}^0 \neq \mathbf{t}^i$ for all $i = 1, \dots, m$ such that p_i is p_0. For instance, the second rule of (14) can be rewritten as the acyclic rule

$$p(f(x)) \leftarrow p(x), f(x) \neq x.$$

For this reason, the requirement in the statement of the theorem below that each of the given rules be acyclic is not an essential limitation.

Proposition 7 *For any finite set F of acyclic rules, the completion of F is equivalent to* PSM$[F]$.

In view of this fact, PSM$[F]$ can be viewed as an extension of the concept of program completion to arbitrary first-order formulas.

[7] On the use of λ-notation in first-order logic, see [Lifschitz, 1994, Section 3.1].

5.3 Tight Formulas

François Fages [1991] showed that if a logic program satisfies a certain syntactic condition, which is now called "tightness," then its stable models can be characterized as the models of its completion. This theorem and its generalizations (see [Erdem and Lifschitz, 2003]) play an important role in answer set programming.

Consider, for instance, logic programs consisting of rules of form (16). According to the definition of a tight program, to decide whether such a program is tight we should look at its "predicate dependency graph." The vertices of this graph are the predicate constants occurring in the program, and its edges lead from p_0 to $p_1, \dots, p_m$ for the rules (16) that the program consists of. The program is called tight if its predicate dependency graph is acyclic.

Proposition 8 below extends Fages's theorem to the general framework introduced in this note. To define the predicate dependency graph for an arbitrary first-order sentence, we need a few auxiliary definitions.

Recall that an occurrence of a subformula or a predicate constant in a formula F is *positive* if the number of implications in F containing that occurrence in the antecedent is even; it is *strictly positive* if that number is 0.[8] In (7), for instance, both occurrences of q are positive, but only the first is strictly positive. The key idea of our definition of the predicate dependency graph for an arbitrary formula F is to concentrate on the implications $G \to H$ that have strictly positive occurrences in F; such implications generalize the concept of a rule in traditional logic programs.

We say that a predicate constant p *depends* on a predicate constant q in an implication $G \to H$ if

- p has a strictly positive occurrence in H, and
- q has a positive occurrence in G that does not belong to any occurrence of a negative formula in G.

The *predicate dependency graph* of a formula F is the directed graph such that

- its vertices are the predicate constants occurring in F, and
- it has an edge from a vertex p to a vertex q if p depends on q in an implication that has a strictly positive occurrence in F.

For instance, the predicate dependency graph of formula (7) has three vertices p, q, r and one edge, from r to q. This is the same graph as the one given by the more special definition reviewed above applied to the "logic programming representation" (1) of formula (7).

Just as in the special case above, we say that a formula F is *tight* if its predicate dependency graph is acyclic. For instance, formulas (7)–(10) are tight; formula (13) is not tight, because its predicate dependency graph is a self-loop.

Proposition 8 *For any tight sentence F,* PSM$[F]$ *is equivalent to* SM$[F]$.

[8] Note that we apply the term "negative" to formulas, and the terms "positive" and "strictly positive" to occurrences of subformulas and predicate constants in a formula.

6 Conclusion

The definition of a stable model proposed in this paper is applicable both to rules covered by the original 1988 definition and to rules of several more general kinds used in answer set programming. Instead of grounding and fixpoints, it refers to a translation into classical logic, and is in this sense close to the definitions of program completion and circumscription.

The relationship between the original definition of a stable model and the definition proposed here can be compared with the relationship between two definitions of a causal theory—the original definition introduced in [McCain and Turner, 1997] and its generalization proposed in [Lifschitz, 1997]. The original definition uses a fixpoint construction; the generalization is based on a translation into classical logic.

Another definition of a stable model for first-order order sentences is given independently by Lin and Zhou [2007]. It refers to grounding, but in other ways it is similar to ours.

Extending main results of the theory of stable models to the general framework described above is a topic of future work.

Acknowledgements

We are grateful to Pedro Cabalar, Martin Gebser and Hudson Turner for useful comments on this paper.

References

[Clark, 1978] Keith Clark. Negation as failure. In Herve Gallaire and Jack Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum Press, New York, 1978.

[Doherty *et al.*, 1997] Patrick Doherty, Witold Łukaszewicz, and Andrzey Szałas. Computing circumscription revisited: A reduction algorithm. *Journal of Automated Reasoning*, 18(3):297–336, 1997.

[Eiter *et al.*, 2006] Thomas Eiter, Michael Fink, Hans Tompits, Patrick Traxler, and Stefan Woltran. Replacements in non-ground answer-set programming. In *Proceedings of International Conference on Principles of Knowledge Representation and Reasoning (KR)*, 2006.

[Erdem and Lifschitz, 2003] Esra Erdem and Vladimir Lifschitz. Tight logic programs. *Theory and Practice of Logic Programming*, 3:499–518, 2003.

[Faber *et al.*, 2004] Wolfgang Faber, Nicola Leone, and Gerard Pfeifer. Recursive aggregates in disjunctive logic programs: Semantics and complexity. In *Proceedings of European Conference on Logics in Artificial Intelligence (JELIA)*, 2004. Revised version: `http://www.wfaber.com/research/papers/jelia2004.pdf`.

[Fages, 1991] 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.

[Ferraris and Lifschitz, 2005a] Paolo Ferraris and Vladimir Lifschitz. Mathematical foundations of answer set programming. In *We Will Show Them! Essays in Honour of Dov Gabbay*, pages 615–664. King's College Publications, 2005.

[Ferraris and Lifschitz, 2005b] Paolo Ferraris and Vladimir Lifschitz. Weight constraints as nested expressions. *Theory and Practice of Logic Programming*, 5:45–74, 2005.

[Ferraris *et al.*, 2006] Paolo Ferraris, Joohyung Lee, and Vladimir Lifschitz. A generalization of the Lin-Zhao theorem. *Annals of Mathematics and Artificial Intelligence*, 2006. To appear.

[Ferraris, 2005] Paolo Ferraris. Answer sets for propositional theories. In *Proceedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, pages 119–131, 2005.

[Gelfond and Lifschitz, 1988] 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, 1988.

[Gelfond and Lifschitz, 1991] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385, 1991.

[Gelfond, 1987] Michael Gelfond. On stratified autoepistemic theories. In *Proceedings of National Conference on Artificial Intelligence (AAAI)*, pages 207–211, 1987.

[Gelfond, 2002] Michael Gelfond. Representing knowledge in A-Prolog. *Lecture Notes in Computer Science*, 2408:413–451, 2002.

[Lee and Lin, 2006] Joohyung Lee and Fangzhen Lin. Loop formulas for circumscription. *Artificial Intelligence*, 170(2):160–185, 2006.

[Lifschitz *et al.*, 2001] Vladimir Lifschitz, David Pearce, and Agustin Valverde. Strongly equivalent logic programs. *ACM Transactions on Computational Logic*, 2:526–541, 2001.

[Lifschitz, 1985] Vladimir Lifschitz. Computing circumscription. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pages 121–127, 1985.

[Lifschitz, 1987] Vladimir Lifschitz. Pointwise circumscription. In Matthew Ginsberg, editor, *Readings in nonmonotonic reasoning*, pages 179–193. Morgan Kaufmann, San Mateo, CA, 1987.

[Lifschitz, 1990] Vladimir Lifschitz. On open defaults. In John Lloyd, editor, *Computational Logic: Symposium Proceedings*, pages 80–95. Springer, 1990.

[Lifschitz, 1994] Vladimir Lifschitz. Circumscription. In D.M. Gabbay, C.J. Hogger, and J.A. Robinson, editors, *The Handbook of Logic in AI and Logic Programming*, volume 3, pages 298–352. Oxford University Press, 1994.

[Lifschitz, 1997] Vladimir Lifschitz. On the logic of causal explanation. *Artificial Intelligence*, 96:451–465, 1997.

[Lin and Zhou, 2007] 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. This volume.

[Lin, 1991] Fangzhen Lin. *A Study of Nonmonotonic Reasoning*. PhD thesis, Stanford University, 1991.

[Lin, 2002] Fangzhen Lin. Reducing strong equivalence of logic programs to entailment in classical propositional logic. In *Proceedings of International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 170–176, 2002.

[Lloyd, 1984] John Lloyd. *Foundations of Logic Programming*. Springer-Verlag, 1984.

[McCain and Turner, 1997] Norman McCain and Hudson Turner. Causal theories of action and change. In *Proceedings of National Conference on Artificial Intelligence (AAAI)*, pages 460–465, 1997.

[McCarthy, 1980] John McCarthy. Circumscription—a form of non-monotonic reasoning. *Artificial Intelligence*, 13:27–39,171–172, 1980.

[McCarthy, 1986] John McCarthy. Applications of circumscription to formalizing common sense knowledge. *Artificial Intelligence*, 26(3):89–116, 1986.

[Moore, 1985] Robert Moore. Semantical considerations on nonmonotonic logic. *Artificial Intelligence*, 25(1):75–94, 1985.

[Pearce and Valverde, 2004] David Pearce and Agustin Valverde. Towards a first order equilibrium logic for nonmonotonic reasoning. In *Proceedings of European Conference on Logics in Artificial Intelligence (JELIA)*, pages 147–160, 2004.

[Pearce and Valverde, 2005] David Pearce and Agustin Valverde. A first order nonmonotonic extension of constructive logic. *Studia Logica*, 80:323–348, 2005.

[Pearce *et al.*, 2001] 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.

[Pearce, 1997] 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-Verlag, 1997.

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

[Reiter, 1982] Raymond Reiter. Circumscription implies predicate completion (sometimes). In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pages 418–420, 1982.

[Simons *et al.*, 2002] Patrik Simons, Ilkka Niemelä, and Timo Soininen. Extending and implementing the stable model semantics. *Artificial Intelligence*, 138:181–234, 2002.