

Stable Models and Circumscription

Paolo Ferraris^a, Joohyung Lee^b, Vladimir Lifschitz^a

^a*Department of Computer Sciences, University of Texas at Austin,
1 University Station C0500, Austin, TX 78712*

^b*Department of Computer Science and Engineering, Arizona State University,
699 South Mill Avenue, Tempe, AZ 85281*

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 John McCarthy's definition of circumscription.

Key words: Answer set programming, Circumscription, Nonmonotonic reasoning, Program completion, Stable models

1 Introduction

Answer set programming (ASP) is a form of logic programming oriented towards combinatorial search problems. It is an outgrowth of early work on autoepistemic logic and default logic. ASP was identified as a new programming paradigm in 1999 [20, 26, 31], and it has found by now a number of serious applications. Syntactically, ASP programs look like Prolog programs, but the computational mechanisms used in ASP are different: they are based on the ideas that have led to the creation of fast satisfiability solvers for propositional logic.

This paper describes a new perspective on the semantics of ASP languages, which relates it to John McCarthy's work on circumscription [28, 29].

Two widely used definitions of the semantics of logic programs—in terms of program completion [1] and in terms of stable models [12]—look very differ-

ent from each other. The former treats a logic program as shorthand for its completion, which is a first-order formula. For instance, the program

$$\begin{aligned} p(a), \\ q(b), \\ r(x) \leftarrow p(x), \text{ not } q(x) \end{aligned} \tag{1}$$

is shorthand for the formula

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

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

$$\begin{aligned} p(a), \\ q(b), \\ r(a) \leftarrow p(a), \text{ not } q(a), \\ r(b) \leftarrow p(b), \text{ not } q(b). \end{aligned} \tag{3}$$

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 ASP have led to the invention of several declarative programming constructs that are not used in Prolog. The completion semantics is not applicable to these constructs, at least directly. For instance, the last rule of the program

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

is a “choice rule” containing a “conditional literal” [37]. 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:

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

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 [29] in the form adopted in [15].

The new definition and examples of its use are discussed in Section 2 below. In Section 3 we relate our definition to the encoding of propositional logic programs by quantified Boolean formulas given in [33], to the definition of an answer set from [5], to a theorem from [22], and to recent research on first-order equilibrium logic [34, 35]. 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. Proofs of theorems are relegated to the appendix.

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 [36]¹ and autoepistemic logic [30]. In the past, stable models were seen as part of the “fixpoint tradition.” In fact,

¹ The translational definition of default logic, proposed in [17], is rather complicated: it uses *third-order* variables.

the invention of stable models was an outgrowth of earlier work on the relationship between logic programming and autoepistemic logic [10]; the first journal paper on answer sets [13] 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 [3] (Vienna University of Technology and University of Calabria), and ASET-Prolog constructs [11, 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 [8, Sections 3.3–3.5, 3.7]. These correctness proofs will use equivalent transformations of formulas of classical logic as the main tool.

This is an extended version of the conference paper [7].

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.²

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.

² 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 [9, Section 4.1]. General aggregates can be described in terms of nested implications [5, Section 4]. Including second (“strong,” “classical,” or “true”) negation without introducing an additional connective is discussed in [8, Section 3.9].

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

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

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

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

Furthermore, we understand the head of a disjunctive rule as the disjunction of its elements, and we view a constraint (rule with the empty head) as the negation of its body. For instance, the disjunctive rule

$$p(x); q(x) \leftarrow r(x)$$

corresponds to the formula

$$\forall x(r(x) \rightarrow (p(x) \vee q(x))),$$

and the constraint

$$\leftarrow p(x), \textit{not } q(x) \quad (8)$$

to the formula

$$\forall x\neg(p(x) \wedge \neg q(x)).$$

Note that the formula $\neg p$ can be written in logic programming notation in two ways: as the rule with the empty body and the head *not p*, and also as the constraint with the body *p*.

In the spirit of [9], program (5) is understood as

$$p(a) \wedge p(b) \wedge \forall x(p(x) \rightarrow (q(x) \vee \neg q(x))). \quad (9)$$

Since the last conjunctive term is logically valid, the class of models of formula (9) 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 [37] into a first-order formula. The rule

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

corresponds to the sentence

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

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

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

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 [18, 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

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

and $p \leq q$ stands for

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

where \mathbf{x} is a tuple of distinct object variables. If \mathbf{p} and \mathbf{q} are tuples p_1, \dots, p_n and q_1, \dots, q_n of predicate constants then $\mathbf{p} = \mathbf{q}$ 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 apply the same notation to tuples of predicate variables.

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

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

where \mathbf{p} stands for the list of all predicate constants occurring in F , \mathbf{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 \mathbf{u} for the constants \mathbf{p} . Intuitively, the second conjunctive term of $\text{CIRC}[F]$ expresses that the extents of the predicates \mathbf{p} are minimal subject to condition F .

For example, if F is

$$p(a) \wedge \forall x(p(x) \rightarrow q(x)) \quad (11)$$

then $\text{CIRC}[F]$ is

$$\begin{aligned} p(a) \wedge \forall x(p(x) \rightarrow q(x)) \\ \wedge \neg \exists uv(((u, v) < (p, q)) \wedge u(a) \wedge \forall x(u(x) \rightarrow v(x))). \end{aligned} \quad (12)$$

Formula (11) expresses that the set represented by p includes the point represented by a and is contained in the set represented by q . The minimality condition in the second line of (12) expresses that the sets represented by p and q cannot be made smaller without losing this property, which amounts to saying that both sets are singletons. Accordingly, (12) can be equivalently rewritten as the first-order formula

$$\forall x(p(x) \leftrightarrow x = a) \wedge \forall x(q(x) \leftrightarrow x = a). \quad (13)$$

[15, Section 5]. There are cases when $\text{CIRC}[F]$ is not equivalent to any first-order formula, as, for instance, when F is

$$p(a) \wedge \forall x(p(x) \rightarrow p(f(x))). \quad (14)$$

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

2.3 Stable Models

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

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

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

- $p_i(t_1, \dots, t_m)^* = u_i(t_1, \dots, t_m)$;
- $(t_1 = t_2)^* = (t_1 = t_2)$;
- $\perp^* = \perp$;
- $(F \wedge G)^* = F^* \wedge G^*$;
- $(F \vee G)^* = F^* \vee G^*$;

- $(F \rightarrow G)^* = (F^* \rightarrow G^*) \wedge (F \rightarrow G)$;
- $(\forall x F)^* = \forall x F^*$;
- $(\exists x F)^* = \exists x F^*$.

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.

The following useful fact is easy to check by induction on F :

Lemma 1 $F^*(\mathbf{p})$ is equivalent to F .

A model of F is *stable* if it satisfies $\text{SM}[F]$.

Example 1 If F is (11) then $F^*(u, v)$ is

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

and $\text{SM}[F]$ is

$$p(a) \wedge \forall x(p(x) \rightarrow q(x)) \\ \wedge \neg \exists uv(((u, v) < (p, q)) \wedge u(a) \wedge \forall x((u(x) \rightarrow v(x)) \wedge (p(x) \rightarrow q(x)))).$$

It is clear that this formula is equivalent to (12), and consequently to (13).

In logic programming notation, (11) can be written as

$$p(a), \\ q(x) \leftarrow p(x).$$

The completion of this program

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

is equivalent to (13) as well. In this example, all three transformations—SM, CIRC and completion—produce essentially the same result.

Example 2 If F is (14) then, as in the previous example, it is clear that $\text{SM}[F]$ is equivalent to $\text{CIRC}[F]$. Consequently, the stable models of (14) 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 $a, f(a), f(f(a)), \dots$

In logic programming notation, (14) can be written as

$$\begin{aligned} p(a), \\ p(f(x)) \leftarrow p(x). \end{aligned} \tag{15}$$

The completion of this program

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

is weaker than $\text{SM}[F]$; some (non-Herbrand³) models of the completion of (15) 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 above. (This assertion remains true if we allow the heads of rules to be disjunctions of atomic formulas.) But if negation in the bodies of rules is allowed then this may be no longer the case, as we will see in Section 2.5.

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 [12]:

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

$$A_0 \leftarrow A_1, \dots, A_m, \text{ not } A_{m+1}, \dots, \text{ not } A_n, \tag{16}$$

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

- X is a stable model of Π in the sense of [12];
- 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-

³ Recall that 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*.

Herbrand models. A similar theorem, relating our definition to [5], is stated in Section 3.1.

2.4 Negative Formulas

Proposition 1 below allows us to simplify the result of applying the operator SM to formulas containing parts of the form $\neg F$, such as rules (16) with $m < n$.

We call a formula *negative* if every occurrence of every predicate constant in it belongs to the antecedent of an implication. For instance, any formula of the form $\neg F$ is negative, because this expression is shorthand for $F \rightarrow \perp$ (Section 2.1). Also, any equality $t_1 = t_2$ is trivially a negative formula.

In the statement of Proposition 1, \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 1 *For any negative formula F , the formula*

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

is logically valid.

Proof: by induction on F .

The role of constraints (see Section 2.1) in ASP is determined by the fact that the effect of adding a constraint to a logic program is to eliminate its stable models violating this constraint [8, Proposition 4]. In the new framework, this fact can be expressed as follows:

Proposition 2 *For any sentence F and any negative sentence G , $\text{SM}[F \wedge G]$ is equivalent to $\text{SM}[F] \wedge G$.*

Proof: by Proposition 1,

$$\begin{aligned} \text{SM}[F \wedge G] &= (F \wedge G) \wedge \neg \exists \mathbf{u} ((\mathbf{u} < \mathbf{p}) \wedge (F \wedge G)^*(\mathbf{u})) \\ &= (F \wedge \neg \exists \mathbf{u} ((\mathbf{u} < \mathbf{p}) \wedge F^*(\mathbf{u}) \wedge G^*(\mathbf{u}))) \wedge G \\ &\Leftrightarrow (F \wedge \neg \exists \mathbf{u} ((\mathbf{u} < \mathbf{p}) \wedge F^*(\mathbf{u}) \wedge G)) \wedge G \\ &\Leftrightarrow (F \wedge \neg \exists \mathbf{u} ((\mathbf{u} < \mathbf{p}) \wedge F^*(\mathbf{u}))) \wedge G \\ &= \text{SM}[F] \wedge G. \end{aligned}$$

Examples of the use of Propositions 1 and 2 are given in the next section.

2.5 Further Examples

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

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

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 1 we can conclude that $(\neg q(x))^*$ can be equivalently replaced here by $\neg q(x)$. Consequently, $\text{SM}[F]$ can be rewritten as

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

Using the methods for eliminating second-order quantifiers described at the end of [18, 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 (9), corresponding to logic program (5), then a similar calculation converts $\text{SM}[F]$ into

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

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

$$\forall x(p(x) \leftrightarrow (x = a \vee x = b)) \wedge \forall x(q(x) \rightarrow (x = a \vee x = b)). \quad (17)$$

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, (9) has 4 Herbrand stable models, and they correspond to sets (6).

Example 5 Let us add constraint (8) to program (5), discussed in the previous example. By Proposition 2, $\text{SM}[F]$ will be then equivalent to the conjunction of (17) with this constraint, which can be further rewritten as

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

3 Relation to Earlier Work

3.1 Stable Models of Propositional and Universal Formulas

In the propositional case, the operator SM turns into the encoding of formulas of equilibrium logic by quantified Boolean formulas proposed in [33] and reviewed in [6, Appendix B].

Theorem 2 below shows that Herbrand stable models of a universal sentence (the universal closure of a quantifier-free first-order formula) can be characterized in terms of the definition of an answer set proposed in [5].

For any formula F , by F^\neq we denote the formula obtained from F by replacing each atomic subformula of the form $t_1 = t_2$, where t_1 and t_2 are ground terms, with \top if t_1 equals t_2 , and with \perp otherwise. This notation will be also applied to sets of formulas. For any quantifier-free formula F , $\text{Ground}(F)$ stands for the set of formulas that can be obtained from F by substituting ground terms for all its variables.

Theorem 2 *Let σ be a signature containing at least one object constant, and F a quantifier-free formula of σ . For any set X of ground atoms of σ , the following conditions are equivalent:*

- X is an answer set of $\text{Ground}(F)^\neq$ in the sense of [5];
- the Herbrand interpretation of σ that makes the elements of X true and all other ground atoms false is a stable model of the universal closure of F .

3.2 Lin's Transformation

Theorem 5 from [22] 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” q 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 $q \leftrightarrow p$. 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 circumscribing the predicate constants \mathbf{p} in a first-order sentence F will be denoted by $\text{CIRC}[F; \mathbf{p}]$. For instance, if F is (11) then $\text{CIRC}[F; q]$ is

$$p(a) \wedge \forall x(p(x) \rightarrow q(x)) \wedge \neg \exists v((v < q) \wedge p(a) \wedge \forall x(p(x) \rightarrow v(x))),$$

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{q} of distinct predicate constants that do not occur in F , of the same length as \mathbf{p} . By $\text{L}[F; \mathbf{q}]$ we denote the formula

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

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

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

Proof: immediate from the definitions of L and SM, using Lemma 1.

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

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

3.3 Equilibrium Logic

The definition of first-order equilibrium logic below is similar to the one proposed in [35, 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 [32].

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 $\sigma^{(I^f, I^h)}$ is defined recursively:

- $I \models p(t_1, \dots, t_n)$ if $p((t_1^I)^*, \dots, (t_n^I)^*) \in I^h$;
- $I \models t_1 = t_2$ if $t_1^I = t_2^I$;
- $I \not\models \perp$;
- $I \models F \wedge G$ if $I \models F$ and $I \models G$;
- $I \models F \vee G$ if $I \models F$ or $I \models G$;
- $I \models F \rightarrow G$ 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 , $I \models F(\xi^*)$;
- $I \models \exists x F(x)$ if, for some ξ from the universe of I^f , $I \models F(\xi^*)$.

(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:

Theorem 3 *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 now in such correctness proofs.⁴ Theorems about stable models proved in the past 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 given in [23].

According to the original definition of strong equivalence [21], (propositional) logic programs Π_1 and Π_2 are said to be strongly equivalent to each other if, for any logic program Π , $\Pi_1 \cup \Pi$ has the same stable models as $\Pi_2 \cup \Pi$. A more general definition was proposed in [8, Section 2.6]: a (propositional) formula F is said to be strongly equivalent to a formula G if any formula F' that contains an occurrence of F has the same stable models as the formula G' obtained from F' by replacing that occurrence with G . This condition is more general not only because it is applicable to arbitrary propositional formulas, but also because F is allowed here to be any subformula of F' , not necessarily a “subconjunction.”

The definition of strong equivalence below is similar to this more general condition, except that it is further extended to formulas with free variables, and the process of replacing an occurrence of F with G is generalized accordingly.

This generalization can be conveniently described in terms of *predicate expressions* $\lambda \mathbf{x}F(\mathbf{x})$, where \mathbf{x} is a list of distinct object variables, and $F(\mathbf{x})$ is a formula. For any formula $H(w)$, where w is a predicate variable of arity equal to the length of \mathbf{x} , by $H(\lambda \mathbf{x}F(\mathbf{x}))$ we denote the formula obtained from $H(w)$ by replacing each atomic subformula of the form $w(\mathbf{t})$, where \mathbf{t} is a tuple of terms, with $F(\mathbf{t})$. For instance, if $H(w)$ is $w(a) \vee w(b)$ then $H(\lambda x \neg p(x))$ stands for $\neg p(a) \vee \neg p(b)$.

Consider two first-order formulas $F(\mathbf{x})$ and $G(\mathbf{x})$, where \mathbf{x} is the list of variables that are free in at least one of them. Let w be a predicate variable of arity equal to the length of \mathbf{x} . We say that $F(\mathbf{x})$ is *strongly equivalent* to $G(\mathbf{x})$

⁴ See, for instance, [8, Sections 2.1–2.4, 2.6–3.1].

if, for any formula $H(w)$, possibly of a larger signature, that has neither free variables other than w nor bound second-order variables, $\text{SM}[H(\lambda\mathbf{x}F(\mathbf{x}))]$ is equivalent to $\text{SM}[H(\lambda\mathbf{x}G(\mathbf{x}))]$.

In the statement of the following theorem, \mathbf{p} is the list of all predicate constants occurring in at least of the formulas F , G , and \mathbf{q} is a tuple of new, pairwise distinct predicate constants of the same length as \mathbf{p} .

Theorem 4 *F is strongly equivalent to G iff the formula*

$$(\mathbf{q} \leq \mathbf{p}) \rightarrow (F^*(\mathbf{q}) \leftrightarrow G^*(\mathbf{q})) \quad (18)$$

is logically valid.

Using this theorem we can check, for instance, that $\neg\forall yF(y)$ is strongly equivalent to $\exists y\neg F(y)$. (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 1, the implications

$$\begin{aligned} (\mathbf{q} \leq \mathbf{p}) &\rightarrow ((\neg\forall yF(y))^* \leftrightarrow \neg\forall yF(y)), \\ (\mathbf{q} \leq \mathbf{p}) &\rightarrow ((\exists y\neg F(y))^* \leftrightarrow \exists y\neg F(y)) \end{aligned}$$

are logically valid; it remains to observe that the right-hand sides of the two equivalences are equivalent to each other.

We can add a few comments regarding the concept of strong equivalence. First, the choice of this term suggests that if F is strongly equivalent to G then F is equivalent to G (in the sense of classical logic). This is indeed true, and easily follows from Theorem 4 and Lemma 1. Second, if F is strongly equivalent to G then replacing an occurrence of F in any formula by G produces a strongly equivalent result; this follows from Lemma 7, used in the proof of Theorem 4 in the appendix. Finally, the fact that the definition of strong equivalence allows $H(w)$ to contain object, function and predicate constants not occurring in F or G turns out to be inessential. Our proof of Theorem 4 shows that if F is not equivalent to G then this can be always established by a counterexample of the same signature.

5 Pointwise Stable Models and Tight Formulas

5.1 Pointwise Stable Models

As observed in [14], program completion is similar in some ways to the concept of pointwise circumscription—the modification of McCarthy’s original defini-

tion that was proposed in [16]. According to either definition of circumscription, circumscribing a predicate constant p makes the extent of p “minimal,” but minimality is understood in different versions differently. According to the original definition, 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 $p \stackrel{1}{<} q$ 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

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

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 $\mathbf{p} \stackrel{1}{<} \mathbf{q}$ stands for the disjunction

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

and similarly for tuples of predicate variables.

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

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

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 $\text{PSM}[F]$. Clearly, every stable model is pointwise stable.

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

Proposition 4 *Formula $\text{PSM}[F]$ is equivalent to*

$$F \wedge \bigwedge_{1 \leq i \leq n} \neg \exists \mathbf{x}^i (p_i(\mathbf{x}^i) \wedge F^*(\mathbf{e}^i(\mathbf{x}^i))),$$

where $\mathbf{e}^i(\mathbf{x}^i)$ stands for the tuple

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

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

Proof: use the fact that for any predicate variable u , any predicate constant p of the same arity, and any formula $H(w)$,

$$\exists u((w \stackrel{1}{<} p) \wedge H(w))$$

is equivalent to

$$\exists \mathbf{x}(p(\mathbf{x}) \wedge H(\lambda \mathbf{y}(p(\mathbf{y}) \wedge \mathbf{y} \neq \mathbf{x}))).$$

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 $\text{PSM}[F]$ 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, $\text{PSM}[F]$ 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

We know from [25] that the process of completing a program can be extended in an obvious way to rules of a more general form than allowed in [1]. It is essential that the head of a rule be an atom, but the body can be an arbitrary first-order formula.

Theorem 5 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 that we consider in this section have the form

$$A_0 \leftarrow A_1 \wedge \cdots \wedge A_m \wedge F \tag{19}$$

where A_0, \dots, A_m ($m \geq 0$) are atomic formulas other than equalities, and F is a negative formula. For instance, every rule of form (16) has also form (19): take F to be $\neg A_{m+1} \wedge \cdots \wedge \neg A_n$.

A rule of form (19) is *acyclic* if for any predicate constant p and any tuples \mathbf{t}, \mathbf{t}' of terms such that $p(\mathbf{t})$ is the head A_0 of the rule and $p(\mathbf{t}')$ is one of the conjunctive terms A_1, \dots, A_m of its body, the formula

$$F \rightarrow \mathbf{t} \neq \mathbf{t}'$$

is logically valid. For instance, each of the rules (1) is obviously acyclic—its body does not contain the predicate constant occurring in the head. Any rule of form (19) 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 (15) 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.

Theorem 5 *For any finite set F of acyclic rules, the completion of F is equivalent to $\text{PSM}[F]$.*

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

5.3 Tight Formulas

If a logic program satisfies a certain syntactic condition, “tightness,” then its stable models can be characterized as the models of its completion [4]. This theorem and its generalizations (see [2]) play an important role in answer set programming.

Consider, for instance, logic programs consisting of rules of form (19). 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 (19) that the program consists of. The program is called tight if its predicate dependency graph is acyclic.

Theorem 6 below extends Fages’s theorem to the general framework introduced in this paper. 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 0.⁵ 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 $G \rightarrow H$ 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 $G \rightarrow H$ 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).

Here are two examples involving “nested implications.” The predicate dependency graph of the formula

$$((p(x) \rightarrow q(x)) \rightarrow r(x)) \rightarrow s(x)$$

has two edges: from s to p and from s to r . The predicate dependency graph of the formula

$$((p(x) \rightarrow x = a) \rightarrow r(x)) \rightarrow s(x)$$

has only one edge, from s to r , because the formula $p(x) \rightarrow x = a$ is negative.

⁵ 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.

A formula F is *tight* if its predicate dependency graph is acyclic. For instance, formulas (7)–(11) are tight; formula (14) is not tight, because its predicate dependency graph is a self-loop.

Theorem 6 *For any tight sentence F , $\text{PSM}[F]$ is equivalent to $\text{SM}[F]$.*

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 [27] and its generalization proposed in [19]. 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 in [24]. It makes use of 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 for future work.

Acknowledgements

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

Appendix: Proofs of Theorems

We will prove Theorem 2 first, and then derive Theorem 1 from it.

Proof of Theorem 2 (Section 3.1)

Recall that answer sets for a set Γ of propositional formulas are defined in [5] as follows. The *reduct* F^X of a formula F relative to a set X of atoms is obtained by replacing every maximal subformula of F that is not satisfied by X with \perp . The reduct Γ^X of a set Γ of formulas relative to X is the set of the reducts F^X of all formulas F in Γ . A set X of atoms is an *answer set* of Γ if X is minimal among the sets of atoms satisfying Γ^X .

Lemma 2 [8, Lemma 22] $X \models F^X$ iff $X \models F$.

Proof: F^X is obtained from F by replacing some subformulas that are not satisfied by X with \perp .

Lemma 3 [8, Lemma 23] (a) $(F \wedge G)^X$ is equivalent to $F^X \wedge G^X$; (b) $(F \vee G)^X$ is equivalent to $F^X \vee G^X$.

Proof. (a) If X satisfies $F \wedge G$ then the formulas $(F \wedge G)^X$ and $F^X \wedge G^X$ are equal to each other; otherwise, each of them is equivalent to \perp . (b) Similar.

The following lemma is a key to the proof of Theorem 2. It relates the reduct operator defined above to the operator $F \mapsto F^*(\mathbf{u})$ introduced in Section 2.3. In the statement of the lemma, $H(\mathbf{x})$ is a quantifier-free formula and \mathbf{x} is the list of all its free variables; \mathbf{p} is the list of all predicate constants occurring in $H(\mathbf{x})$; \mathbf{t} is a list of ground terms of the same length as \mathbf{x} ; X is a set of ground atoms. We identify an Herbrand interpretation with the set of ground atoms (other than equalities) to which it assigns the value *true*.

Lemma 4 *A subset Y of X satisfies $(H(\mathbf{t})^\neq)^X$ iff the Herbrand interpretation $X \cup Y_{\mathbf{q}}^{\mathbf{p}}$ satisfies $H^*(\mathbf{q}, \mathbf{t})$, where \mathbf{q} is a list of new, distinct predicate constants of the same length as \mathbf{p} .*

(Here $Y_{\mathbf{q}}^{\mathbf{p}}$ is the set of ground atoms obtained from Y by substituting the members of \mathbf{q} for the corresponding members of \mathbf{p} ; $H^*(\mathbf{q}, \mathbf{t})$ is the sentence obtained from $H^*(\mathbf{u}, \mathbf{x})$ by substituting the predicate constants \mathbf{q} for the predicate variables \mathbf{u} and the terms \mathbf{t} for the object variables \mathbf{x} .)

Proof by induction on H . *Case 1:* $H(\mathbf{x})$ has the form $t_1(\mathbf{x}) = t_2(\mathbf{x})$. Then $H^*(\mathbf{q}, \mathbf{t})$ is $t_1(\mathbf{t}) = t_2(\mathbf{t})$; $X \cup Y_{\mathbf{q}}^{\mathbf{p}}$ satisfies this sentence iff $t_1(\mathbf{t})$ equals $t_2(\mathbf{t})$. On the other hand, $(H(\mathbf{t})^\neq)^X$ is \top if $t_1(\mathbf{t})$ equals $t_2(\mathbf{t})$, and \perp otherwise. *Case 2:* $H(\mathbf{x})$ has the form $p(\mathbf{t}'(\mathbf{x}))$, where $\mathbf{t}'(\mathbf{x})$ is a tuple of terms. Then $H^*(\mathbf{q}, \mathbf{t})$ is $q(\mathbf{t}'(\mathbf{t}))$, where q is the member of \mathbf{q} corresponding to the member p of \mathbf{p} ; $X \cup Y_{\mathbf{q}}^{\mathbf{p}}$ satisfies this sentence iff $p(\mathbf{t}'(\mathbf{t}))$ belongs to Y . On the other hand, $(H(\mathbf{t})^\neq)^X$ is $p(\mathbf{t}'(\mathbf{t}))$ if this atom belongs to X , and \perp otherwise. Since $Y \subseteq X$, Y satisfies $(H(\mathbf{t})^\neq)^X$ iff $p(\mathbf{t}'(\mathbf{t}))$ belongs to Y . *Case 3:* $H(\mathbf{x})$ is a conjunction or

a disjunction; use Lemma 3. *Case 4:* $H(\mathbf{x})$ is $H_1(\mathbf{x}) \rightarrow H_2(\mathbf{x})$. Then $H^*(\mathbf{q}, \mathbf{t})$ is

$$H(\mathbf{t}) \wedge (H_1^*(\mathbf{q}, \mathbf{t}) \rightarrow H_2^*(\mathbf{q}, \mathbf{t})). \quad (20)$$

Case 4.1: $X \models H(\mathbf{t})^\neq$. Then the Herbrand interpretation $X \cup Y_{\mathbf{q}}^{\mathbf{p}}$ satisfies the conjunction (20) iff it satisfies its second term $H_1^*(\mathbf{q}, \mathbf{t}) \rightarrow H_2^*(\mathbf{q}, \mathbf{t})$. On the other hand, $(H(\mathbf{t})^\neq)^X$ is in this case $H_1(\mathbf{t})^\neq \rightarrow H_2(\mathbf{t})^\neq$, and it remains to apply the induction hypothesis. *Case 4.2:* $X \not\models H(\mathbf{t})^\neq$. Then $X \cup Y_{\mathbf{q}}^{\mathbf{p}}$ does not satisfy (20), and $(H(\mathbf{t})^\neq)^X$ is \perp .

Since we have agreed to identify an Herbrand interpretation with the set of ground atoms to which it assigns the value *true*, Theorem 2 (Section 3.1) can be stated as follows:

Let σ be a signature containing at least one object constant, and F a quantifier-free formula of σ . For any set X of ground atoms of σ , the following conditions are equivalent:

- X is an answer set of $\text{Ground}(F)^\neq$ in the sense of [5];
- X is a stable model of the universal closure of F .

Proof. In view of Lemma 2, X is an answer set of $\text{Ground}(F(\mathbf{x}))^\neq$ iff

- (i) X satisfies $\text{Ground}(F(\mathbf{x}))^\neq$, and
- (ii) no proper subset Y of X satisfies the reduct $(\text{Ground}(F(\mathbf{x}))^\neq)^X$.

On the other hand, X is a stable model of $\forall \mathbf{x}F(\mathbf{x})$ iff

- (i') X satisfies $\forall \mathbf{x}F(\mathbf{x})$, and
- (ii') X does not satisfy $\exists \mathbf{u}((\mathbf{u} < \mathbf{p}) \wedge \forall \mathbf{x}F^*(\mathbf{u}, \mathbf{x}))$.

It is clear that (i) is equivalent to (i'). Condition (ii) can be stated as follows: no proper subset Y of X satisfies all of the formulas $(F(\mathbf{t})^\neq)^X$ for arbitrary tuples \mathbf{t} of ground terms. Condition (ii') can be reformulated in terms of a tuple of new predicate constants \mathbf{q} : there is no proper subset Y of X such that, for every tuple \mathbf{t} of ground terms, $X \cup Y_{\mathbf{q}}^{\mathbf{p}}$ satisfies $F^*(\mathbf{q}, \mathbf{t})$. By Lemma 4, it follows that (ii) is equivalent to (ii').

Proof of Theorem 1 (Section 2.3)

The original definition [12] of a stable model for a set Π of ground rules of form (16) can be stated as follows. The *traditional reduct* of Π relative to a

set X of (ground) atoms is the set of rules

$$A_0 \leftarrow A_1, \dots, A_m \tag{21}$$

for all rules (16) of Π such that $A_{m+1}, \dots, A_n \notin X$. A stable model of Π is any set X of atoms such that the minimal model of the traditional reduct of Π relative to X is X .

The following lemma (essentially identical to Proposition 28 from [8]) shows that Theorem 1 is a special case of Theorem 2.

Lemma 5 *A set X of atoms is the minimal model of the traditional reduct of Π relative to X iff X is an answer set of Π in the sense of [5].*

Proof. Let Π^X denote the traditional reduct of Π relative to X .

Case 1: $X \not\models \Pi$. Set X is not an answer set of Π in the sense of [5]. On the other hand, Π contains a rule (16) such that $A_1, \dots, A_m \in X$ and $A_0, A_{m+1}, \dots, A_n \notin X$. The corresponding rule (21) in Π^X is not satisfied by X , so that X is different from the minimal model of Π^X .

Case 2: $X \models \Pi$. We will show that the reduct Π^X and the traditional reduct Π^X are satisfied by the same subsets of X . Since Π^X is the conjunction of the formulas R^X for all rules R of Π , and Π^X is the union of the programs $\{R\}^X$ for all rules R of Π , it is sufficient to verify this claim for the case when Π is a single rule (16). If X contains at least one of the atoms A_{m+1}, \dots, A_n then Π^X is empty and Π^X is the tautology $\perp \rightarrow A_0^X$. Otherwise Π^X is (21). If $A_1, \dots, A_m \in X$ then $A_0 \in X$, because $X \models \Pi$; consequently Π^X is the result of replacing A_{m+1}, \dots, A_n in (16) with \perp , which is equivalent to (21). It remains to consider the case when $A_{m+1}, \dots, A_n \notin X$ and at least one of the atoms A_1, \dots, A_m , say A_1 , does not belong to X . In this case Π^X is the tautology $\perp \rightarrow A_0^X$. On the other hand, Π^X is the rule (21) whose body contains A_1 and consequently is not satisfied by any subset of X . It follows that every subset of X satisfies Π^X .

Proof of Theorem 3 (Section 3.3)

As in Section 3.3, we represent here an interpretation of the underlying signature σ at the pair $\langle I, J \rangle$, where I interprets the function constants of σ , and J is a set of atomic formulas formed using predicate constants from σ and names ξ^* of elements ξ of the universe.

Take a tuple \mathbf{q} of new, pairwise distinct predicate constants of the same length as the tuple \mathbf{p} of the predicate constants in σ . The definition of a stable model

(Section 2.3) can be reformulated as follows: $\langle I, J \rangle$ is said to be a stable model of F if

- $\langle I, J \rangle \models F$, and
- for any proper subset J' of J , $\langle I, J \cup (J')^{\mathbf{p}} \rangle \not\models F^*(\mathbf{q})$.

(Recall that $(J')^{\mathbf{p}}$ stands for the set of atoms obtained from J' by replacing each predicate constant from \mathbf{p} with the corresponding predicate constant from \mathbf{q} .) In view of this fact, the assertion of Theorem 3 is immediate from the following lemma:

Lemma 6 *For any sentence F , possibly containing object constants ξ^* ,*

- (i) $\langle I, J, J \rangle \models F$ iff $\langle I, J \rangle \models F$,
- (ii) for any subset J' of J , $\langle I, J', J \rangle \models F$ iff $\langle I, J \cup (J')^{\mathbf{p}} \rangle \models F^*(\mathbf{q})$.

Proof: by induction on F .

Proof of Theorem 4 (Section 4)

Lemma 7 *Let $F(\mathbf{x})$ and $G(\mathbf{x})$ be first-order formulas, \mathbf{x} the list of variables that are free in at least one of them, w a predicate variable of arity equal to the length of \mathbf{x} , and $H(w)$ a formula, possibly of a larger signature, that has neither free second-order variables other than w nor bound second-order variables. The formula*

$$\forall \mathbf{x}((F(\mathbf{x}) \leftrightarrow G(\mathbf{x})) \wedge (F^*(\mathbf{u}, \mathbf{x}) \leftrightarrow G^*(\mathbf{u}, \mathbf{x}))) \rightarrow (H_F^*(\mathbf{u}) \leftrightarrow H_G^*(\mathbf{u})),$$

where H_F stands for $H(\lambda \mathbf{x}F(\mathbf{x}))$ and H_G for $H(\lambda \mathbf{x}G(\mathbf{x}))$, is logically valid.

Proof: by induction on $H(w)$.

The following lemma is equivalent to the “if” part of Theorem 4.

Lemma 8 *If the formula*

$$(\mathbf{u} \leq \mathbf{p}) \rightarrow (F^*(\mathbf{u}, \mathbf{x}) \leftrightarrow G^*(\mathbf{u}, \mathbf{x})) \tag{22}$$

is logically valid then $F(\mathbf{x})$ is strongly equivalent to $G(\mathbf{x})$.

Proof. We need to show, in notation on Lemma 7, that

$$H_F \wedge \neg \exists \mathbf{u}((\mathbf{u} < \mathbf{p}) \wedge H_F^*(\mathbf{u})) \tag{23}$$

is equivalent to

$$H_G \wedge \neg \exists \mathbf{u}((\mathbf{u} < \mathbf{p}) \wedge H_G^*(\mathbf{u})). \quad (24)$$

Since (22) is logically valid, $F^*(\mathbf{p}, \mathbf{x})$ is equivalent to $G^*(\mathbf{p}, \mathbf{x})$. By Lemma 1, it follows that $F(\mathbf{x})$ is equivalent to $G(\mathbf{x})$. Consequently the first conjunctive term of (23) is equivalent to the first conjunctive term of (24). By Lemma 7, it follows that the second conjunctive terms of (23) and (24) are equivalent to each other also.

In the proof of the other half of the theorem (Lemma 9) below, we refer to the conjunction of the “choice formulas”

$$\forall \mathbf{y}(p(\mathbf{y}) \vee \neg p(\mathbf{y})),$$

where \mathbf{y} is a tuple of distinct object variables, for all predicate constants p in \mathbf{p} . We will denote this conjunction by C . A straightforward calculation shows that $C^*(\mathbf{u})$ is equivalent to $\mathbf{p} \leq \mathbf{u}$.

Lemma 9 *If $F(\mathbf{x})$ is strongly equivalent to $G(\mathbf{x})$ then (22) is logically valid.*

Proof. Let E stand for $\forall \mathbf{x}(F(\mathbf{x}) \leftrightarrow G(\mathbf{x}))$, and let E' be $\forall \mathbf{x}(F(\mathbf{x}) \leftrightarrow F(\mathbf{x}))$. Since $F(\mathbf{x})$ is strongly equivalent to $G(\mathbf{x})$, the formula $\text{SM}[E \leftrightarrow C]$ is equivalent to $\text{SM}[E' \leftrightarrow C]$.⁶ It is easy to see that $(E \leftrightarrow C)^*$ can be rewritten as

$$E \wedge (E^*(\mathbf{u}) \leftrightarrow (\mathbf{p} \leq \mathbf{u})),$$

and that $E^*(\mathbf{u})$ is equivalent to

$$E \wedge \forall \mathbf{x}(F^*(\mathbf{u}, \mathbf{x}) \leftrightarrow G^*(\mathbf{u}, \mathbf{x})).$$

⁶ We understand an equivalence as shorthand for the conjunction of two implications.

Using these two facts and Lemma 1, we can simplify $\text{SM}[E \leftrightarrow C]$ as follows:

$$\begin{aligned}
& \text{SM}[E \leftrightarrow C] \\
& \Leftrightarrow (E \leftrightarrow C) \wedge \neg \exists \mathbf{u}((\mathbf{u} < \mathbf{p}) \wedge E \wedge (E^*(\mathbf{u}) \leftrightarrow (\mathbf{p} \leq \mathbf{u}))) \\
& \Leftrightarrow E \wedge \neg \exists \mathbf{u}((\mathbf{u} < \mathbf{p}) \wedge (E^*(\mathbf{u}) \leftrightarrow (\mathbf{p} \leq \mathbf{u}))) \\
& \Leftrightarrow E \wedge \neg \exists \mathbf{u}((\mathbf{u} < \mathbf{p}) \wedge \neg E^*(\mathbf{u})) \\
& \Leftrightarrow E \wedge \neg \exists \mathbf{u}((\mathbf{u} < \mathbf{p}) \wedge \neg \forall \mathbf{x}(F^*(\mathbf{u}, \mathbf{x}) \leftrightarrow G^*(\mathbf{u}, \mathbf{x}))) \\
& = \forall \mathbf{x}(F(\mathbf{x}) \leftrightarrow G(\mathbf{x})) \wedge \forall \mathbf{x}\mathbf{u}((\mathbf{u} < \mathbf{p}) \rightarrow (F^*(\mathbf{u}, \mathbf{x}) \leftrightarrow G^*(\mathbf{u}, \mathbf{x}))) \\
& \Leftrightarrow \forall \mathbf{x}((F^*(\mathbf{p}, \mathbf{x}) \leftrightarrow G^*(\mathbf{p}, \mathbf{x})) \wedge \forall \mathbf{x}\mathbf{u}((\mathbf{u} < \mathbf{p}) \rightarrow (F^*(\mathbf{u}, \mathbf{x}) \leftrightarrow G^*(\mathbf{u}, \mathbf{x})))) \\
& \Leftrightarrow \forall \mathbf{x}\mathbf{u}(\mathbf{u} \leq \mathbf{p} \rightarrow (F^*(\mathbf{u}, \mathbf{x}) \leftrightarrow G^*(\mathbf{u}, \mathbf{x}))).
\end{aligned}$$

The last formula entails (22). Similarly, $\text{SM}[E' \leftrightarrow C]$ is equivalent to

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

which is logically valid. Consequently, (22) is logically valid also.

Proof of Theorem 5 (Section 5.2)

Lemma 10 *The formula*

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

is logically valid.

Proof: by induction on F .

Theorem 5 *For any finite set F of acyclic rules, the completion of F is equivalent to $\text{PSM}[F]$.*

Proof. The completion of F is the conjunction of (the universal closures of the rules of) F with the formulas

$$\forall \mathbf{x}^i \left(p_i(\mathbf{x}^i) \rightarrow \bigvee_{(\mathbf{t}, B, \mathbf{y}) \in \Gamma_i} \exists \mathbf{y}(\mathbf{t} = \mathbf{x}^i \wedge B) \right) \quad (1 \leq i \leq n)$$

where Γ_i stands for the set of all triples $(\mathbf{t}, B, \mathbf{y})$ such that F contains the rule $p_i(\mathbf{t}) \leftarrow B$, and \mathbf{y} is the list of all free variables of this rule. On the other hand, $\text{PSM}[F]$ can be written as the conjunction of F with the formulas

$$\neg \exists \mathbf{x}^i (p_i(\mathbf{x}^i) \wedge F^*(\mathbf{e}^i(\mathbf{x}^i))) \quad (1 \leq i \leq n)$$

where $\mathbf{e}^i(\mathbf{x}^i)$ stands for the tuple

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

(Proposition 4). These formulas can be rewritten as

$$\forall \mathbf{x}^i(p_i(\mathbf{x}^i) \rightarrow \neg F^*(\mathbf{e}^i(\mathbf{x}^i))).$$

Consequently, we will prove the assertion of Theorem 5 if we derive the equivalence between the formulas

$$\bigvee_{(\mathbf{t}, B, \mathbf{y}) \in \Gamma_i} \exists \mathbf{y}(\mathbf{t} = \mathbf{x}^i \wedge B) \quad (25)$$

and

$$\neg F^*(\mathbf{e}^i(\mathbf{x}^i)) \quad (26)$$

from assumption F .

Formula $F^*(\mathbf{u})$ can be rewritten, under assumption F , as the conjunction of the formulas

$$\forall \mathbf{y}(B^*(\mathbf{u}) \rightarrow u_j(\mathbf{t}))$$

for all $j = 1, \dots, n$ and all $(\mathbf{t}, B, \mathbf{y}) \in \Gamma_j$. The j -th term of the tuple $\mathbf{e}^i(\mathbf{x}^i)$ is $\lambda \mathbf{y}^i(p_i(\mathbf{y}^i) \wedge \mathbf{y}^i \neq \mathbf{x}^i)$ if $j = i$, and p_j otherwise. Consequently, the conjunctive terms of $F^*(\mathbf{e}^i(\mathbf{x}^i))$ can be divided into two groups:

$$\forall \mathbf{y}(B^*(\mathbf{e}^i(\mathbf{x}^i)) \rightarrow (p_i(\mathbf{t}) \wedge \mathbf{t}^i \neq \mathbf{x}^i)) \quad (27)$$

for all $(\mathbf{t}, B, \mathbf{y}) \in \Gamma_i$, and

$$\forall \mathbf{y}(B^*(\mathbf{e}^i(\mathbf{x}^i)) \rightarrow p_j(\mathbf{t})) \quad (28)$$

for all $(\mathbf{t}, B, \mathbf{y}) \in \Gamma_j$ with $j \neq i$. By Lemma 10, $B^*(\mathbf{e}^i(\mathbf{x}^i))$ entails B . Consequently, in the presence of the conjunctive term $\forall \mathbf{y}(B \rightarrow p_j(\mathbf{t}))$ of F , formulas (27) can be rewritten as

$$\forall \mathbf{y}(B^*(\mathbf{e}^i(\mathbf{x}^i)) \rightarrow \mathbf{t}^i \neq \mathbf{x}^i), \quad (29)$$

and formulas (28) can be dropped altogether.

We conclude that formula (26) can be written as

$$\neg \bigwedge_{(\mathbf{t}, B, \mathbf{y}) \in \Gamma_i} \forall \mathbf{y}(B^*(\mathbf{e}^i(\mathbf{x}^i)) \rightarrow \mathbf{t}^i \neq \mathbf{x}^i),$$

or, equivalently,

$$\bigvee_{(\mathbf{t}, B, \mathbf{y}) \in \Gamma_i} \exists \mathbf{y} (B^*(\mathbf{e}^i(\mathbf{t}^i)) \wedge \mathbf{t}^i = \mathbf{x}^i). \quad (30)$$

Since the rule $B \rightarrow p_i(\mathbf{t})$ is acyclic, the formula $B^*(\mathbf{e}^i(\mathbf{t}^i))$ is equivalent to B . Consequently, (30) is equivalent to (25).

Proof of Theorem 6 (Section 5.3)

In the following lemma, F is a first-order formula, \mathbf{p} is the list of predicate constants occurring in F , and \mathbf{u} is a tuple of distinct predicate variables of the same length as \mathbf{p} .

Lemma 11 *Let S be the set of i 's such that p_i has a strictly positive occurrence in F . The formula*

$$\left((\mathbf{u} \leq \mathbf{p}) \wedge \bigwedge_{i \in S} (p_i = u_i) \right) \rightarrow (F \leftrightarrow F^*(\mathbf{u}))$$

is logically valid.

Proof: by induction on F , using Lemma 10.

In the following lemma, \mathbf{v} is a tuple of distinct predicate variables disjoint from \mathbf{u} .

Lemma 12 *Let S^+ be the set of i 's such that p_i has a positive occurrence in F that does not belong to a negative formula; let S^- be the set of i 's such that p_i has a nonpositive occurrence in F that does not belong to a negative formula. The formulas*

- (a) $((\mathbf{u} \leq \mathbf{v}) \wedge (\mathbf{v} \leq \mathbf{p}) \wedge \bigwedge_{i \in S^+} (p_i = u_i)) \rightarrow (F^*(\mathbf{v}) \rightarrow F^*(\mathbf{u}))$,
- (b) $((\mathbf{u} \leq \mathbf{v}) \wedge (\mathbf{v} \leq \mathbf{p}) \wedge \bigwedge_{i \in S^-} (p_i = u_i)) \rightarrow (F^*(\mathbf{u}) \rightarrow F^*(\mathbf{v}))$

are logically valid.

Proof. Both parts are proved simultaneously by induction on F . We will only consider the proof of (a) in the case when F is an implication $F_1 \rightarrow F_2$. Assume the antecedent

$$(\mathbf{u} \leq \mathbf{v}) \wedge (\mathbf{v} \leq \mathbf{p}) \wedge \bigwedge_{i \in S^+} (p_i = u_i).$$

If F is a negative formula then, by Proposition 1, each of the formulas $F^*(\mathbf{u})$, $F^*(\mathbf{v})$ is equivalent to F . Otherwise, the consequent

$$(F \wedge (F_1^*(\mathbf{v}) \rightarrow F_2^*(\mathbf{v}))) \rightarrow (F \wedge (F_1^*(\mathbf{u}) \rightarrow F_2^*(\mathbf{u})))$$

follows from

$$F_1^*(\mathbf{u}) \rightarrow F_1^*(\mathbf{v})$$

(part (b) of the induction hypothesis applied to F_1) and

$$F_2^*(\mathbf{v}) \rightarrow F_2^*(\mathbf{u})$$

(part (a) of the induction hypothesis applied to F_2).

Lemma 13 *Let D be the set of edges of the predicate dependency graph of F . The formula*

$$\left((\mathbf{u} \leq \mathbf{v}) \wedge (\mathbf{v} \leq \mathbf{p}) \wedge \bigwedge_{(p_i, p_j) \in D} (u_j = p_j \vee v_i = p_i) \right) \rightarrow (F^*(\mathbf{u}) \rightarrow F^*(\mathbf{v}))$$

is logically valid.

Proof. By induction on F . We will only consider the case when F is an implication $F_1 \rightarrow F_2$. Assume

$$(\mathbf{u} \leq \mathbf{v}) \wedge (\mathbf{v} \leq \mathbf{p}) \wedge \bigwedge_{(p_i, p_j) \in D} (u_j = p_j \vee v_i = p_i); \quad (31)$$

we need to derive

$$F^*(\mathbf{u}) \rightarrow F^*(\mathbf{v}). \quad (32)$$

Case 1: For each predicate constant p_i that has a strictly positive occurrence in F , $v_i = p_i$. Then, by Lemma 11, $F^*(\mathbf{v})$ is equivalent to F , which is one of the conjunctive terms of $F^*(\mathbf{u})$. *Case 2:* For some predicate p_i that has a strictly positive occurrence in F , $\neg(v_i = p_i)$. From the last conjunctive term of (31) we conclude then $u_j = p_j$ for every j such that $(p_i, p_j) \in D$. By Lemma 12(a), it follows that

$$F_1^*(\mathbf{v}) \rightarrow F_1^*(\mathbf{u}).$$

In combination with the assumption

$$F \wedge (F_1^*(\mathbf{u}) \rightarrow F_2^*(\mathbf{u}))$$

(the antecedent of (32)) and the formula

$$F_2^*(\mathbf{u}) \rightarrow F_2^*(\mathbf{v})$$

(the induction hypothesis applied to F_2), we conclude $F \wedge (F_1^*(\mathbf{v}) \rightarrow F_2^*(\mathbf{v}))$.
(the consequent of (32)).

Theorem 6 *For any tight sentence F , $\text{PSM}[F]$ is equivalent to $\text{SM}[F]$.*

Proof. We only need to prove the implication left-to-right. Since F is tight, we can assume without loss of generality that the members p_1, \dots, p_n of \mathbf{p} are ordered in such a way that $i < j$ for all edges (p_i, p_j) of the dependency graph of F . Assume $\text{PSM}[F]$ and $\mathbf{u} < \mathbf{p}$; we need to prove $\neg F^*(\mathbf{u})$. Let m be the largest i such that $u_i \neq p_i$. Take \mathbf{x} such that $p_m(\mathbf{x})$ but not $u_m(\mathbf{x})$. Define \mathbf{v} as follows: v_i is $\lambda \mathbf{y}(p_i(\mathbf{y}) \wedge \mathbf{x} \neq \mathbf{y})$ if $i = m$, and p_i otherwise. For this choice of \mathbf{v} , we can derive the antecedent

$$(\mathbf{u} \leq \mathbf{v}) \wedge (\mathbf{v} \leq \mathbf{p}) \wedge \bigwedge_{(p_i, p_j) \in D} (u_j = p_j \vee v_i = p_i)$$

of the formula from Lemma 13. Indeed, $\mathbf{u} \leq \mathbf{v}$ and $\mathbf{v} \leq \mathbf{p}$ are immediate, as well as the second disjunctive term of $u_j = p_j \vee v_i = p_i$ for any i different from m . Any j such that $(p_m, p_j) \in D$ is greater than m ; by the choice of m , we get the second disjunctive term $u_j = p_j$. Hence, by Lemma 13,

$$F^*(\mathbf{u}) \rightarrow F^*(\mathbf{v}).$$

On the other hand, $\mathbf{v} \stackrel{1}{<} \mathbf{p}$, so that, in view of $\text{PSM}[F]$, we can conclude that $\neg F^*(\mathbf{v})$. Consequently $\neg F^*(\mathbf{u})$.

References

- [1] Keith Clark. Negation as failure. In Herve Gallaire and Jack Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum Press, New York, 1978.
- [2] Esra Erdem and Vladimir Lifschitz. Tight logic programs. *Theory and Practice of Logic Programming*, 3:499–518, 2003.
- [3] 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>.
- [4] 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.
- [5] Paolo Ferraris. Answer sets for propositional theories. In *Proceedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, pages 119–131, 2005.

- [6] Paolo Ferraris, Joohyung Lee, and Vladimir Lifschitz. A generalization of the Lin-Zhao theorem. *Annals of Mathematics and Artificial Intelligence*, 2006. To appear.
- [7] Paolo Ferraris, Joohyung Lee, and Vladimir Lifschitz. A new perspective on stable models. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, 2007. To appear.
- [8] 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.
- [9] Paolo Ferraris and Vladimir Lifschitz. Weight constraints as nested expressions. *Theory and Practice of Logic Programming*, 5:45–74, 2005.
- [10] Michael Gelfond. On stratified autoepistemic theories. In *Proceedings of National Conference on Artificial Intelligence (AAAI)*, pages 207–211, 1987.
- [11] Michael Gelfond. Representing knowledge in A-Prolog. *Lecture Notes in Computer Science*, 2408:413–451, 2002.
- [12] 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.
- [13] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385, 1991.
- [14] Joohyung Lee and Fangzhen Lin. Loop formulas for circumscription. *Artificial Intelligence*, 170(2):160–185, 2006.
- [15] Vladimir Lifschitz. Computing circumscription. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pages 121–127, 1985.
- [16] Vladimir Lifschitz. Pointwise circumscription. In Matthew Ginsberg, editor, *Readings in nonmonotonic reasoning*, pages 179–193. Morgan Kaufmann, San Mateo, CA, 1987.
- [17] Vladimir Lifschitz. On open defaults. In John Lloyd, editor, *Computational Logic: Symposium Proceedings*, pages 80–95. Springer, 1990.
- [18] 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.
- [19] Vladimir Lifschitz. On the logic of causal explanation. *Artificial Intelligence*, 96:451–465, 1997.
- [20] Vladimir Lifschitz. Action languages, answer sets and planning. In *The Logic Programming Paradigm: a 25-Year Perspective*, pages 357–373. Springer Verlag, 1999.
- [21] Vladimir Lifschitz, David Pearce, and Agustin Valverde. Strongly equivalent logic programs. *ACM Transactions on Computational Logic*, 2:526–541, 2001.
- [22] Fangzhen Lin. *A Study of Nonmonotonic Reasoning*. PhD thesis, Stanford

- University, 1991.
- [23] 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.
 - [24] 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.
 - [25] John Lloyd and Rodney Topor. Making Prolog more expressive. *Journal of Logic Programming*, 3:225–240, 1984.
 - [26] 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.
 - [27] Norman McCain and Hudson Turner. Causal theories of action and change. In *Proceedings of National Conference on Artificial Intelligence (AAAI)*, pages 460–465, 1997.
 - [28] John McCarthy. Circumscription—a form of non-monotonic reasoning. *Artificial Intelligence*, 13:27–39,171–172, 1980.
 - [29] John McCarthy. Applications of circumscription to formalizing common sense knowledge. *Artificial Intelligence*, 26(3):89–116, 1986.
 - [30] Robert Moore. Semantical considerations on nonmonotonic logic. *Artificial Intelligence*, 25(1):75–94, 1985.
 - [31] Ilkka Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 25:241–273, 1999.
 - [32] David Pearce. A new logical characterization of stable models and answer sets. In Jürgen Dix, Luis Pereira, and Teodor Przymusiński, editors, *Non-Monotonic Extensions of Logic Programming (Lecture Notes in Artificial Intelligence 1216)*, pages 57–70. Springer-Verlag, 1997.
 - [33] 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.
 - [34] 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.
 - [35] David Pearce and Agustin Valverde. A first order nonmonotonic extension of constructive logic. *Studia Logica*, 80:323–348, 2005.
 - [36] Raymond Reiter. A logic for default reasoning. *Artificial Intelligence*, 13:81–132, 1980.
 - [37] Patrik Simons, Ilkka Niemelä, and Timo Soinen. Extending and implementing the stable model semantics. *Artificial Intelligence*, 138:181–234, 2002.