Approximation Fixpoint Theory and the Semantics of Logic and Answers Set Programs

Marc Denecker, Maurice Bruynooghe, and Joost Vennekens

Department of Computer Science, KULeuven

Abstract. Approximation Fixpoint Theory was developed as a fixpoint theory of lattice operators that provides a uniform formalization of four main semantics of three major nonmonotonic reasoning formalisms. The aim of this paper is, first, to clarify the application of this fixpoint theory to define the stable and well-founded semantics of logic programs, and second, to show the remarkable power of this theory for defining natural and elegant versions of these semantics for extensions of logic and answer set programs. In particular, we here consider extensions with general rule bodies, general interpretations (also non-Herbrand interpretations) and aggregates. We also investigate the relationship with the equilibrium semantics of nested answer set programs, on the formal and the informal level.

1 Introduction

In the nineties, Przymusiński [19], Vangelder [20] and Fitting [8] applied algebraic techniques to the semantics of Logic Programming. This culminated in a framework proposed by Fitting who demonstrated that the main four semantics of logic programs can be described in abstract algebraic terms of fixpoints of two operators [8]. Motivated by certain analogies between this work and techniques used elsewhere in nonmonotonic reasoning, Denecker, Marek and Truszczyński further generalized this work into an algebraic fixpoint theory defining four types of fixpoints for a bilattice operator [3]. They found that with this theory, the family of four semantics of logic programming can be generated from one operator, namely Fittings (three- or four-valued) immediate consequence operator. They also showed that the fixpoint theory induces isomorphic families including all main and some new semantics for default and autoepistemic logic [4], thus effectively unifying both logics. Thus, this work demonstrated common algebraic foundations in three major nonmonotonic logics.

The goal of this paper, which we dedicate to the 65'th birthday of Vladimir Lifschitz, is to give a survey and an explanation of the application of this approximation fixpoint theory to the semantics of logic programming and answer set programming and generalizations of these. We believe that it is fitting to write such a paper for several reasons.

First, [3] presented the fixpoint theory but not in detail how to apply it to logic programming. Here we fill in this gap and provide theorems and their proofs

to substantiate the claims in that paper, for the first time. As we will see, proofs are really straightforward. In fact, this is probably one of the main strenghts of this theory.

Second, we recall and analyse the stable semantics defined in [22]. There a version of stable semantics was defined through approximation fixpoint theory, for programs with arbitrary first order (FO) rule bodies and arbitrary structures (including non-Herbrand ones). It introduced these generalized versions of stable (and also well-founded) semantics in the context of a technical enquiry of *predicate introduction* but missed any motivation and analysis of the resulting logic from a semantic point of view.

Here in this paper, we first recall this formalization which is very close to the standard way of defining stable semantics, and then show how it can be derived using the four-valued immediate consequence operator. We compare this version of the stable semantics with equilibrium semantics of nested answer set programs [16, 17]. We prove a correspondence for a subclass of programs without nested negation. For programs outside this class, the semantics are quite different. We study the underlying notion of equivalence in our semantics (the equivalent of *strong equivalence* of equilibrium semantics) and show that it is much more “classic” than strong equivalence.

We end this paper with an illustration of the flexibility of approximation fixpoint theory to support language extension. Based on the work of [18], we show how to extend the formalism with recursion over aggregate expressions. Our aim here is to show the remarkable ability of approximation fixpoint theory to define, in a mathematically elegant way, how to build principled and natural semantics for extensions of logic programming and answer set semantics. Moreover, the work recalls how closely related stable and well-founded semantics are.

2 Approximation Fixpoint Theory of lattice operators

This algebraic fixpoint theory for arbitrary lattice operators was defined in [3]. This theory can be summarized as follows.

A complete lattice $\langle L, \leq \rangle$ is a partial order such that each subset has a least upperbound and greatest lowerbound. It induces a complete bilattice $\langle L^2, \leq_p \rangle$, where $\leq_p$ is the precision order on L^2 defined as follows: $(x, y) \leq_p (u, v)$ if $x \leq u$ and $v \leq y$. Tuples (x, y) such that $x \leq y$ are called *consistent*. Such tuples can be understood as approximating lattice elements u between x and y , i.e., for which $x \leq u \leq y$. Tuples (x, x) approximate only x ; they are called *exact*. Exact tuples are the maximally precise consistent pairs. Abusing notation, we will write $(x, y) \leq_p u$ to denote $(x, y) \leq_p (u, u)$ and say “ (x, y) approximates u ”.

Applications of approximation fixpoint theory are initiated by defining one of two sorts of operators. An *approximator* $A : L^2 \rightarrow L^2$ is a $\leq_p$ -monotone operator that is *symmetric*, i.e., $A(x, y) = (u, v)$ if and only if $A(y, x) = (v, u)$. Let us denote by $A_1, A_2 : L^2 \rightarrow L$ the projections of A on the first, respectively second argument; i.e., $A(x, y) = (A_1(x, y), A_2(x, y))$. By symmetry of A , we have that $A_2(x, y) = A_1(y, x)$. Thus, $A(x, y) = (A_1(x, y), A_1(y, x))$.

The second sort of operator is a *monotone-antimonotone* operator $Ap : L^2 \rightarrow L$ which is monotone in its first and anti-monotone in its second argument: when $x \leq u$ and $y \geq v$, it holds that $Ap(x, y) \leq Ap(u, v)$.

Approximators and monotone-antimonotone operators are two sides of the same coin. Indeed, for an arbitrary operator $Ap : L^2 \rightarrow L$, define the operator $A : L^2 \rightarrow L^2 : (x, y) \rightarrow A(x, y) = (Ap(x, y), Ap(y, x))$. Consequently, A_1 is identical to Ap . The following now holds:

Proposition 1. *Ap is monotone-antimonotone iff A is an approximator.*

As we shall see, sometimes it is easier to define Ap , sometimes it is more natural to define A .

Given an approximator A (or equivalently, a monotone-antimonotone Ap), we can define three derived operators $O_A : L \rightarrow L$, $S_A : L \rightarrow L$ and $\mathcal{S}_A : L^2 \rightarrow L^2$. The operators, their definition and the names of their (least) fixpoints are shown in Table 1. A and $\mathcal{S}_A$ are $\leq_p$ -monotone and have $\leq_p$ -least fixpoints in L^2 ; S_A is $\leq$ -anti-monotone and O_A is arbitrary; both may have 0 or more fixpoints in L .

Operator	Definition of the operator	(Least) Fixpoint
$A : L^2 \rightarrow L^2$	$A(x, y) = (A_1(x, y), A_1(y, x))$	Kripke-Kleene least fixpoint
$O_A : L \rightarrow L$	$O_A(x) = A_1(x, x)$	Supported fixpoints
$S_A : L \rightarrow L$	$S_A(x) = lfp(A_1(\cdot, x))$	Stable fixpoints
$\mathcal{S}_A : L^2 \rightarrow L^2$	$\mathcal{S}_A(x, y) = (S_A(y), S_A(x))$	Well-founded fixpoint

Table 1. Operators and their fixpoints.

The operator O_A is the lattice operator approximated by A . Indeed, when $(x, y) \leq_p u$ then $A(x, y) \leq_p O_A(u)$ and $A(x, x) = (O_A(x), O_A(x))$. Fixpoints of O_A are called *supported* and they correspond to exact fixpoints of A .

A crucial operator is the *stable operator* S_A , where $S_A(x)$ is defined as $lfp(A_1(\cdot, x))$ (or $lfp(Ap(\cdot, x))$). This operator is antimonotone. It inherits this property from the anti-monotonicity of A_1 in the second argument. Indeed, consider $u = S_A(x)$ and $v = S_A(y)$ for $x \leq y$. We have that $u = lfp(A_1(\cdot, x))$, hence $u = A_1(u, x)$. By anti-monotonicity of A_1 in its second argument, $u = A_1(u, x) \geq A_1(u, y)$. It follows that u is a *pre-fixpoint* of the monotone operator $A_1(\cdot, y)$. Since the least fixpoint of a monotone operator is also its least pre-fixpoint, it follows that $v = lfp(A_1(\cdot, y)) \leq u$.

Anti-monotone operators have interesting properties. In particular, they have a *maximal oscillation pair* (u, v) : (u, v) is an oscillation pair (i.e., $S_A(u) = v$, $S_A(v) = u$) and for each oscillation pair (x, y) of S_A , we have that $u \leq x$ and $v \geq y$. It follows that $(u, v) \leq_p (x, y)$. Since stable fixpoints w correspond to exact oscillation pairs (w, w) , it follows that (u, v) is less precise than any stable fixpoint.

The operator $\mathcal{S}_A$ is called the well-founded operator. It is defined by $\mathcal{S}_A(x, y) = (S_A(y), S_A(x))$. It follows from the anti-monotonicity of S_A that $\mathcal{S}_A$ is $\leq_p$ -monotone in L^2 . Its fixpoints correspond exactly to the oscillation pairs of S_A ; its least fixpoint is (u, v) , the maximal oscillation pair of S_A ; this is called the well-founded fixpoint of A . Fixpoints of $\mathcal{S}_A$ are also fixpoints of A ; indeed, if $x = lfp(A_1(\cdot, y)), y = lfp(A_1(\cdot, x))$ it holds that $x = A_1(x, y)$ and $y = A_1(y, x) = A_2(x, y)$, hence $A(x, y) = (x, y)$.

The names of these fixpoints reflect the well-known semantics of logic programming. Indeed, if we take the four-valued versions of Fitting's three immediate consequence operator for A [7], it was claimed in [3] that the four different types of fixpoints of this operator correspond to four well-known semantics of logic programming: Kripke-Kleene semantics [7], supported model semantics, the restriction of the completion semantics [2] to Herbrand models, stable semantics [10] and well-founded semantics [21]. Remarkably enough, formal proofs of these results so far did not appear in print. Proofs will be included here for the first time.

3 Relationship to the stable model semantics of Logic Programs

In honour of Vladimir Lifschitz and Michael Gelfond, the first step of our enquiry of the applications of approximation fixpoint theory in LP and ASP is its relationship with the stable model semantics [10]. We show it here for the propositional case; in later sections we consider predicate logic programs.

A (propositional) vocabulary Σ is a set of propositional symbols. A Σ -interpretation is a subset of Σ ; i.e., it is an element of 2^Σ . A normal rule (over Σ) is of the form:

$$p \leftarrow p_1, \dots, p_n, \text{not } q_1, \dots, \text{not } q_m$$

with $p, p_1, \dots, p_n, q_1, \dots, q_m \in \Sigma$. A normal logic program Π over Σ is a collection of normal rules over Σ . With a normal logic program Π we associate its immediate consequence operator $\Gamma^\Pi : 2^\Sigma \rightarrow 2^\Sigma$ which maps interpretations M to interpretations I consisting of all p such that for some rule

$$p \leftarrow p_1, \dots, p_n, \text{not } q_1, \dots, \text{not } q_m \in \Pi,$$

it holds that $p_1, \dots, p_n \in M$ and $q_1, \dots, q_m \notin M$.

The Gelfond-Lifschitz reduct Π_M of Π under Σ -interpretation M uses the interpretation M to reduce the normal program Π to a definite program (without negation in bodies). Formally, it is defined as the program consisting of rules:

$$p \leftarrow p_1, \dots, p_n$$

for which there exists a rule $p \leftarrow p_1, \dots, p_n, \text{not } q_1, \dots, \text{not } q_m \in \Pi$ such that $q_1, \dots, q_m \notin M$.

Π_M is a definite logic program. As a consequence, Γ^{Π_M} is monotone and has a least fixpoint $lfp(\Gamma^{\Pi_M})$. The operator that maps M to $lfp(\Gamma^{\Pi_M})$ is called

the Gelfond-Lifschitz operator GL_Π of Π . M is called a stable model of Π if $M = GL_\Pi(M)$, i.e., if $M = lfp(\Gamma^{\Pi_M})$.

The link with the Approximation fixpoint theory is direct.

Definition 1. Define $T_1^\Pi : 2^\Sigma \times 2^\Sigma \rightarrow 2^\Sigma$ as the operator that maps (I, M) to $\Gamma^{\Pi_M}(I)$.

We observe that this operator evaluates positive body literals in its first argument, and, due to the construction of the reduct, its negative body literals in its second argument. $T_1^\Pi(I, M)$ is the set K of atoms p such that for some rule $p \leftarrow p_1, \dots, p_n, \text{not } q_1, \dots, \text{not } q_m \in \Pi$, $p_1, \dots, p_n \in I$ and $q_1, \dots, q_m \notin M$.

An obvious but crucial property is the following.

Proposition 2. T_1^Π is a monotone-antimonotone operator; i.e., it is monotone in its first and antimonotone in its second argument.

Definition 2. Define $T^\Pi : 2^\Sigma \times 2^\Sigma \rightarrow 2^\Sigma \times 2^\Sigma : (I, J) \rightarrow (T_1^\Pi(I, J), T_1^\Pi(J, I))$.

Proposition 3. T^Π is an approximator in the bilattice of $2^\Sigma, \subseteq$. Its first component is T_1^Π . The stable operator S_{T^Π} of T^Π is nothing else than the Gelfond-Lifschitz operator of Π . Stable fixpoints of T^Π are exactly the stable models of Π .

Proof. By Proposition 1, T^Π is an approximator with T_1^Π as first component. We have that $GL_\Pi(M) = lfp(\Gamma^{\Pi_M}) = lfp(T_1^\Pi(\cdot, M)) = S_{T^\Pi}(M)$. Hence, stable models of Π are exactly the stable fixpoints of T^Π .

This proposition was informally mentioned in [3], but was never explicitly stated or proven.

Relationship with well-founded semantics As T^Π is an approximator, the well-founded least fixpoint of T^Π is the maximal oscillating pair of S_{T^Π} . In [20], this maximal oscillating pair was used to characterize the well-founded model of Π . In particular, this pair was proven to be structurally isomorphic with the three-valued well-founded model of Π as defined in [21].

In Section 5, we explain this isomorphism. There we will also show a property that was obvious to the authors of [3] but was never proven and never appeared in print: that T^Π is isomorphic with the three and four-valued immediate consequence operator of Π as defined by Fitting [7, 8]. This latter result shows that the four types of semantics of LP (Kripke-Kleene, supported, stable, well-founded) are indeed generated from the four-valued immediate consequence operator through approximation fixpoint theory.

But first, we extend the stable semantics to arbitrary FO rule bodies and arbitrary non-Herbrand structures.

4 Extending the stable semantics to predicate rule sets and non-Herbrand structures

In this section, we present a version of the stable semantics for rule sets with arbitrary first order logic (FO) formulas in the body. Moreover, this version of the semantics allows stable models to be arbitrary structures, including non-Herbrand ones. The semantic theory defined below was presented for the first time in [22] in the context of an enquiry of *predicate introduction*. However, the resulting semantics was also defined through another semantic theory in [18] as will be shown in the next section.

To define our semantics, we need some standard machinery from first order (FO) logic syntax and semantics.

4.1 Preliminaries: FO syntax and semantics

A predicate *vocabulary* Σ is now a finite set of predicate symbols Σ^P and function symbols Σ^F , each such predicate or function symbol with an associated arity. Constants are function symbols with arity 0. We often denote a symbol S with arity n by S/n . Terms and first order FO formulas over a vocabulary Σ are defined in the standard way.

We say that (an occurrence of) a subformula ψ of ϕ is *positive* in ϕ if it occurs in the scope of an even number of negations; it is *negative* otherwise.

An occurrence of x in ψ is *bound* if it occurs in the scope of a quantifier, otherwise the occurrence is *free*. A *free variable* of ψ is one with a free occurrence.

As usual, we denote a formula φ by $\varphi[\mathbf{x}]$ to indicate that the set of free variables of φ is a subset of $\mathbf{x}$.

A FO formula without free variables is called a (*FO*) *sentence*.

A Σ -*interpretation* or *structure* I consists of a domain D and an assignment of a relation $P^I \subseteq D^n$ to each predicate symbol $P/n \in \Sigma$ and an assignment of a function $F^I : D^n \rightarrow D$ to each function symbol $F/n \in \Sigma$.

A *pre-interpretation* of Σ consists of a domain and an interpretation of the function symbols.

A (variable) *assignment* θ in D is a mapping from variables to domain elements of D .

If I is a Σ -interpretation and $\Sigma' \subseteq \Sigma$, we denote by $I|_{\Sigma'}$ the restriction of I to the symbols of Σ' .

If Σ_1 and Σ_2 are two disjoint vocabularies, I a Σ_1 -interpretation with domain D and J a Σ_2 -interpretation with the same domain, then $I + J$ denotes the unique $(\Sigma_1 \cup \Sigma_2)$ -interpretation with domain D such that $(I + J)|_{\Sigma_1} = I$ and $(I + J)|_{\Sigma_2} = J$.

We define the FO satisfaction relation $I, \theta \models \psi$ between structures I , variable assignments θ and FO formulas ψ by the standard recursion. By slight abuse of notation, we often will write $I \models \psi[\vec{d}]$ as a shorthand for $I, \theta \models \psi[\vec{x}]$, where $\theta[x] = \vec{d}$.

Finally, we introduce the two truth values **t** and **f**, and define the strict truth order $\mathbf{f} <_t \mathbf{t}$ and the inverses $(\mathbf{f})^{-1} = \mathbf{t}$ and $(\mathbf{t})^{-1} = \mathbf{f}$. The truth evaluation

function $\psi^{I,\theta}$ is defined as follows: $\varphi^{I,\theta} = \mathbf{t}$ if $I, \theta \models \varphi$ and $\varphi^{I,\theta} = \mathbf{f}$ otherwise. For a formula $\psi[\bar{x}]$ and assignment θ such that $\theta(\bar{x}) = \bar{d}$, we often write $\psi[\bar{d}]^I$ instead of $\psi[\bar{x}]^{I,\theta}$.

The truth order point-wise extends to interpretations. If I and J are two Σ -interpretations with the same pre-interpretation, then we say that $I \leq_t J$ if for every predicate symbol P and tuple of domain elements $\mathbf{d}$ it holds that $P^I(\mathbf{d}) \leq_t P^J(\mathbf{d})$.

4.2 Generalized programs and stable and well-founded semantics

Definition 3. A rule is a formula of the form $\forall \bar{x}(P(\bar{t}) \leftarrow \varphi[\bar{x}])$ with $P/n \in \Sigma^P$, $\bar{t} = (t_1, \dots, t_n)$ n terms, $\bar{x}$ the set of all free variables in $P(\bar{t})$ and φ , and φ a FO formula. A program Π is a set of rules.

We will often omit the outer universal quantifiers.

We now extend the stable semantics to predicate programs Π and, moreover, also define non-Herbrand stable models. The idea in [22] is to extend the operator $T_1^\Pi(I, M)$ to this general case. The crucial property that needs to be preserved, if we want to maintain the relationship with approximation fixpoint theory, is that the extended operator should be monotone-antimonotone.

In the case for normal programs, these properties were a result of the fact that I was only used to evaluate positive rule literals, and M only negative rule literals. It is fairly straightforward to extend this to arbitrary FO bodies, namely if we use I to interpret atoms that occur positively in rule bodies and M to interpret atoms that occur negatively.

To this aim, we define a satisfaction function that performs such a dual evaluation of atoms in positive and negative contexts.

Definition 4 (Pos-neg evaluation relation φ^{+I-J}). Let I_p be a pre-interpretation of Σ . Let φ be a Σ -formula, I and J be Σ -interpretations extending I_p and θ an assignment. We define the pos-neg evaluation of φ in I and J and θ , denoted by $\varphi^{+I-J,\theta}$, by induction over the size of φ :

- for an atom $\varphi = P(\mathbf{t})$, $\varphi^{+I-J,\theta} = \varphi^{I,\theta}$;
- for $\varphi = \neg\psi$, $\varphi^{+I-J,\theta} = (\psi^{+J-I,\theta})^{-1}$;
- the normal recursive rules for $\wedge, \vee, \exists, \forall$.
E.g., $(\psi \wedge \phi)^{+I-J,\theta} = \text{Min}(\psi^{+I-J,\theta}, \phi^{+I-J,\theta})$.

Note that this definition evaluates atoms in I but switches the role of I and J when entering negation. Thus, indeed, positively occurring atoms are evaluated in I , negatively occurring atoms in J .

Proposition 4. The pos-neg evaluation $\psi^{+I-J,\theta}$ is monotone in I and anti-monotone in J .

Proof. By a routine inductive argument.

Definition 5. Define the operator T_1^Π on all pairs of interpretations (I, J) sharing the same pre-interpretation I_p as $T_1^\Pi(I, J) = K$ if K has pre-interpretation I_p (the same as I and J), and moreover, for each $P/n \in \Sigma^P$ and tuple of domain elements $\bar{d}$, $P^K(\bar{d}) = \mathbf{t}$ iff for some rule $\forall \bar{x}(P(\bar{t}) \leftarrow \varphi[\bar{x}])$ and variable assignment θ , $\bar{t}^{K,\theta} = \bar{d}$ and $\varphi[\bar{x}]^{+I-J,\theta} = \mathbf{t}$.

Proposition 5. T_1^Π is monotone-antimonotone.

This immediately follows from the monotonicity properties of pos-neg truth evaluation.

Thus, we can again define the operator $T^\Pi(I, J) = (T_1^\Pi(I, J), T_1^\Pi(J, I))$. For each pre-interpretation I_p , the set of Σ -interpretations extending I_p under the truth order $\leq_t$ is a complete lattice.

Definition 6. A Σ -interpretation M is a stable model of Π if $M = \text{lf}p(T^\Pi(\cdot, M))$.

Or equivalently, M is a stable model if it is a fixpoint of the stable operator S_{T^Π} which is defined as always, $S_{T^\Pi}(M) = \text{lf}p(T_1^\Pi(\cdot, M))$.

The above definition evidently generalizes the standard definition.

Well-founded semantics For each pre-interpretation I_p , the operator S_{T^Π} is an anti-monotone operator that defines a maximal oscillation pair (I, J) . This pair approximates all stable models (since these correspond to exact oscillating pairs). As we shall see in the next section, this pair is structurally isomorphic to a generalized notion of three-valued well-founded model.

Parametrized stable semantics It has become quite common to consider programs with input and output predicates. Initially in answer set programming, one particular form of this idea was found in idea of LP-functions [9]. A different form was introduced in LP modules [11].

Our semantics can be extended to cover parametrized stable semantics by generalizing the notion of a pre-interpretation. Let us assume that a pre-interpretation now not only provides the domain and interpretation of function symbols but also of certain input or parameter predicate symbols that occur only in rule bodies, not in heads. All above definitions generalize.

Relationship to Equilibrium semantics. In [16, 17], a semantics called *equilibrium semantics* was defined for logic programs and nested answer set programs, based on the notion of the logic of Here and There (HT). We present this semantics and study its relationship to ours.

A formula of HT is defined as in FO. For simplicity, this discussion is limited to the propositional case. The structures of HT are pairs (I, J) such that $I \subseteq J$. The truth function $v_{(I,J)}(\varphi)$ is defined by induction:

- $v_{(I,J)}(p) = p^I$, for $p \in \Sigma$
- the standard rules for $\wedge, \vee$
- $v_{(I,J)}(\neg\psi) = \mathbf{t}$ if $v_{(I,J)}(\psi) = \mathbf{f}$ and $v_{(J,J)}(\psi) = \mathbf{f}$

- $v_{(I,J)}(\psi \rightarrow \phi) = \mathbf{t}$ if $v_{(I,J)}(\psi) = \mathbf{t}$ implies $v_{(I,J)}(\phi) = \mathbf{t}$ and $v_{(J,J)}(\psi) = \mathbf{t}$ implies $v_{(J,J)}(\phi) = \mathbf{t}$.

An interpretation M is an *equilibrium model* of a HT theory T if (i) $v_{(M,M)}(T) = \mathbf{t}$ and (ii) if, for $I \leq M$, it holds that $v_{(I,M)}(T) = \mathbf{t}$ then $I = M$.

For program Π , define $HT(\Pi)$ as the set of formulas $\{\varphi \rightarrow A \mid A \leftarrow \varphi \in \Pi\}$. It was shown in [16] that for normal programs Π , the standard stable models correspond to the equilibrium models of $HT(\Pi)$. This correspondence extends to the case of nested programs [13].

Let us investigate the relationship to our semantics. First, they disagree for simple programs.

Example 1. Let $\Pi = \{p \leftarrow \neg\neg p\}$. The corresponding HT theory has two equilibrium models $\{\}$ and $\{p\}$. On the other hand, it has only one stable model, namely $\{\}$. This follows from the fact that $(\neg\neg p)^{+I-J} = p^{+I-J} = p^I$. As a consequence, this program induces the same stable operator as $\{p \leftarrow p\}$ whose only fixpoint is $\{\}$.

A striking similarity of the two semantics is the use of truth functions based on pairs of interpretations. We can show that in some cases these truth functions coincide. That will allow us to prove that under certain circumstances both semantics coincide.

Proposition 6. *Let φ be a formula without nested negations (and no $\rightarrow, \leftrightarrow$). For each $I \subseteq J$, it holds that $v_{I,J}(\varphi) = \varphi^{+I-J}$.*

Proof. The proof is by induction on the structure of φ . In both definitions, the rules for the truth value of atoms, $\wedge$ and $\vee$ are identical. So, we need to consider only the case of formulas $\neg\psi$. Since by assumption φ contains no nested negation, ψ is a positive formula.

We have $v_{(I,J)}(\neg\psi) = \mathbf{t}$ if $v_{(I,J)}(\psi) = \mathbf{f}$ and $v_{(J,J)}(\psi) = \mathbf{f}$ iff (by the induction hypothesis) $\psi^{+I-J} = \mathbf{f}$ and $\psi^J = \mathbf{f}$.

Let us consider this last condition a bit closer. The formula ψ is a positive formula, therefore $\psi^{+J-I} = \psi^J$. Moreover, it always holds that $\psi^{+I-J} \leq \psi^J$. Hence, the condition $\psi^{+I-J} = \mathbf{f}$ and $\psi^J = \mathbf{f}$ is equivalent to $\psi^{+J-I} = \mathbf{f}$ which in turn is equivalent to $\neg\psi^{+I-J} = \mathbf{t}$.

Theorem 1. *If no rule body of a logic program Π contains nested negation, then M is a stable model of Π (as defined in this paper) iff M is an equilibrium model of $HT(\Pi)$.*

Proof. M is an equilibrium model of $HT(\Pi)$ iff (i) $v_{(M,M)}(HT(\Pi)) = \mathbf{t}$ and (ii) $v_{(I,M)}(HT(\Pi)) = \mathbf{t}$ for $I \leq M$ implies $I = M$.

Let us look at the first condition. With φ a rule body, the truth function $v_{M,M}(\varphi)$ coincides with the standard truth function. Hence (i) expresses that M is a model of $PC(\Pi)$, the set of propositional material implications syntactically identical to $HT(\Pi)$. In turn, this condition is equivalent to M being a prefixpoint of the immediate consequence operator of Π . And this is equivalent to $T_1^\Pi(M, M) \leq M$.

Now we consider the second condition. When $v_{(I,M)}(HT(\Pi)) = \mathbf{t}$, this means that for each $A \leftarrow \varphi \in \Pi$, $v_{(I,M)}(\varphi) = \varphi^{+I-M} = \mathbf{t}$ implies $v_{(I,M)}(A) = A^I = \mathbf{t}$ and $v_{(M,M)}(\varphi) = \varphi^M = \mathbf{t}$ implies $v_{(M,M)}(A) = A^M = \mathbf{t}$. It follows that $v_{(I,M)}(HT(\Pi)) = \mathbf{t}$ holds iff $T_1^\Pi(I, M) \leq I$ and $T_1^\Pi(M, M) \leq M$. Thus, the second condition is that if $T_1^\Pi(I, M) \leq I$ and $T_1^\Pi(M, M) \leq M$ then $I = M$.

Given that $T_1^\Pi(M, M) \leq M$ because of (i), we obtain that M is an equilibrium model iff M is a fixpoint of $T_1^\Pi(\cdot, M)$ and this operator has no smaller pre-fixpoints. This is equivalent to saying that $M = lfp(T_1^\Pi(\cdot, M))$.

Discussion Equilibrium semantics is defined for a larger class of programs than stable semantics defined here, namely for all of HT, i.e., for all sentences in FO syntax. Whether this is an advantage or not depends on whether it formalizes a natural generalization of the concept of a rule to the full FO formalism. So far no natural applications exist of nested answer set programs beyond the standard rules that we have considered here. To a great extent, the question of whether the above two semantics makes sense depends on the *informal semantics* formalized by each. This issue, the intuitive meaning of language constructs such as negation and the rule operator, and of rules induced by the semantics has not been addressed for either of the semantics. In the next section, we will briefly comment on this issue for the stable semantics defined here.

Part of the problem of making intuitive sense of both logics is due to the used truth evaluation function. The evaluation of rules in equilibrium semantics is based on the logic of Here and There. While this logic has a long history, it is nevertheless the case that little or nothing is known about its informal semantics¹. Questions such as: what is the intuitive interpretation of negation and implication operator in HT, remain unclear to this day. But also the definition of the pos-neg evaluation may seem awkward and its motivation unclear. In the following section, we provide some explanation by laying a simple and tight link between this truth function and the standard Kleene three-valued truth function and Belnap's four-valued extension. At the same time, this will clarify the link between T^Π and the three- and four-valued immediate consequence operator of Π .

5 Generating stable and well-founded semantics from three- and four-valued semantics

5.1 The bilattice view on three- and four-valued logic

There is an intimate connection between the approximation of an interpretation by a pair of interpretations and Belnap's four-valued logic [1]. We now present this.

We denote the truth values *true*, *false*, *unknown* and *inconsistent* of four-valued logic by respectively $\mathbf{t}$, $\mathbf{f}$, $\mathbf{u}$ and $\mathbf{i}$. Define $\mathbf{i}^{-1} = \mathbf{i}$, $\mathbf{u}^{-1} = \mathbf{u}$. On these truth values, the *truth order* $\leq_t$ and *precision order* $\leq_p$ are defined as shown in Figure 1.

¹ David Pearce, private communication, ASP'03 Workshop, Messina.

Fig. 1. The truth and precision order

A four-valued relation of arity n on some domain D is a function from D^n to $\{\mathbf{t}, \mathbf{f}, \mathbf{u}, \mathbf{i}\}$. It is three-valued if $\mathbf{i}$ is not in the image. A four-valued (three-valued) interpretation $\mathcal{I}$ of vocabulary Σ consists of a pre-interpretation and, for each predicate symbol $P/n \in \Sigma$ a four-valued (three-valued) relation $P^{\mathcal{I}}$ of arity n in domain $D(\mathcal{I})$.

The truth and precision orders pointwise extend to interpretations. E.g. if $\mathcal{I}$ and $\mathcal{J}$ are two Σ -interpretations sharing the pre-interpretation, then we say that $\mathcal{I} \leq_p \mathcal{J}$ if for every predicate symbol P and tuple of domain elements $\bar{d}$ it holds that $P^{\mathcal{I}}(\bar{d}) \leq_p P^{\mathcal{J}}(\bar{d})$. Similarly, also the truth order is extended to interpretations.

Definition 7. Let $\mathcal{I}$ be a four-valued Σ -structure, θ a variable assignment. For each formula φ over Σ , we define its truth value $\varphi^{\mathcal{I}, \theta}$ by essentially the same induction as for 2-valued logic:

- $P(\bar{t})^{\mathcal{I}, \theta} = P^{\mathcal{I}}(\bar{t}^{\mathcal{I}})$;
- The standard rules for $\neg, \wedge, \neg\forall, \exists$.
E.g. $(\exists x\psi)^{\mathcal{I}, \theta} = \text{Max}_{\leq_t} \{\psi^{\mathcal{I}, \theta[x:d]} \mid d \in D_{\mathcal{I}}\}$.

This truth evaluation was originally proposed by Belnap as an extension of the standard Kleene evaluation [12] for three-valued structures.

A key property of this truth evaluation is that when $\mathcal{I}$ approximates I , that is, $\mathcal{I} \leq_p I$, then $\varphi^{\mathcal{I}, \theta} \leq_p \varphi^{I, \theta}$. This shows that three-valued interpretations can be used as approximations of two-valued ones. Thus, we see that three-valued interpretations could play the same role as consistent pairs in the bilattice, as approximations of exact lattice elements. In fact, both are isomorphic views.

There is a natural isomorphism between Belnap's four truth values and pairs of the two standard truth values:

$$\begin{aligned} \tau(\mathbf{t}, \mathbf{t}) &= \mathbf{t}; \\ \tau(\mathbf{f}, \mathbf{t}) &= \mathbf{u}; \\ \tau(\mathbf{t}, \mathbf{f}) &= \mathbf{i}; \\ \tau(\mathbf{f}, \mathbf{f}) &= \mathbf{f}. \end{aligned}$$

Intuitively, this mapping τ interprets its first argument as an underestimate to the “real” truth value, and its second argument as an overestimate: if the

underestimate is $\mathbf{f}$ and the overestimate is $\mathbf{t}$, then the real truth value is indeed unknown; whereas, if the underestimate is $\mathbf{t}$ and the overestimate is $\mathbf{f}$, then there cannot exist a real truth value, since $\mathbf{t} \not\leq \mathbf{f}$, so we end up with inconsistency. This isomorphism τ extends in an obvious way to an isomorphism between pairs (I, J) of two-valued interpretations and four-valued interpretations $\mathcal{I} = \tau(I, J)$ which all share the same pre-interpretations. I.e., for each predicate P/n and tuple $\bar{d} \in D^n$, $P^{\mathcal{I}}(\bar{d}) = \tau(P^I(\bar{d}), P^J(\bar{d}))$.

The tight link between our pos-neg evaluation function ϕ^{+I-J} and Belnap evaluation $\phi^{\mathcal{I}}$ is now as follows.

Proposition 7. *For each four-valued Σ -interpretation $\mathcal{I}$, assignment θ and formula φ over Σ :*

$$\phi^{\mathcal{I}, \theta} = \tau(\phi^{+I-J, \theta}, \phi^{+J-I, \theta}), \text{ where } \tau(I, J) = \mathcal{I}.$$

This proposition seems to be folk theorem in the area of three- and four-valued logic. The proof is by a routine induction. We did not find a proof but a related result is found in [6]. In the rest of this paper, we will often omit the isomorphism τ , and simply denote the four-valued truth value of a formula ϕ in a pair of interpretations (I, J) as $\phi^{(I, J)}$. An important property, that we already stated above in different notation, is that $\varphi^{(I, J)} \leq_p \varphi^K$ for all $K \in [I, J]$.

What this says about the intuition underlying the pos-neg evaluation, is that this function serves to compute *approximate* truth values for formulas, *both from below and from above*. Indeed, when (I, J) approximates K , then compared to φ^K , φ^{+I-J} underestimates φ^K and φ^{+J-I} overestimates φ^K . To see this, observe that φ^{+I-J} underestimates positive occurring atoms and overestimates negatively occurring atoms. The inverse is true for φ^{+J-I} , leading to an overestimation of φ^K . The inverse happens with φ^{+J-I} . The link with the four-valued truth evaluation is that the latter performs the same computations (under- and overestimation) simultaneously.

5.2 Semantics of Programs through the four-valued consequence operator

The (two-valued) immediate consequence operator Γ^Π of a program Π extends to three- and four-valued interpretations in an obvious way. The three-valued version was defined first in [7], the four-valued one in [8].

Let Π be a program over Σ . The operator Ψ^Π is a mapping of four-valued Σ -interpretations, that maps $\mathcal{I}$ to $\mathcal{J}$ if the following condition holds:

- the pre-interpretation of $\mathcal{I}$ and $\mathcal{J}$ are the same;
- for each predicate P and tuple $\bar{d} \in D^n$, it holds that

$$P^{\mathcal{J}}(\bar{d}) = \text{Max}_{\leq_t} \{ \varphi^{\mathcal{I}, \theta} \mid \text{there exists a rule } p(\bar{t}) \leftarrow \varphi \in \Pi \\ \text{and assignment } \theta \text{ such that } \bar{t}^{\mathcal{I}, \theta} = \bar{d} \}.$$

We can simplify its definition if we first normalize the program using a well-known procedure akin to predicate completion. In particular, we replace the set of rules $\forall \bar{x}_i (P(\bar{t}_i) \leftarrow \varphi_i)$ with the same predicate P in the head by one rule:

$$\forall \bar{y} (P(\bar{y}) \leftarrow \dots \vee \exists \bar{x}_i (\bar{t}_i = \bar{y} \wedge \varphi_i) \vee \dots)$$

We obtain a program in which each predicate is defined by one rule $\forall \bar{y} (P(\bar{y}) \leftarrow \varphi_P[\bar{y}])$. For such programs, the definition of the immediate consequence operator simplifies. For $\mathcal{J} = \Psi^\Pi(\mathcal{I})$, $P^\mathcal{J}$ is defined for all tuples $\bar{d} \in D^n$ as:

$$P^\mathcal{J}(\bar{d}) = \varphi[\bar{d}]^\mathcal{I}$$

Proposition 8. *For each program Π and four-valued interpretation $\mathcal{I}$, we have that $\Psi^\Pi(\mathcal{I}) = \tau(T^\Pi(\tau(\mathcal{I})))$.*

Proof. Follows from the isomorphism between pairs of interpretations four-valued interpretations and the correspondence between pos-neg and Belnap truth evaluation. In particular, if $\tau(I, J) = \mathcal{I}$, $T^\Pi(I, J) = (K, L)$, $\Psi^\Pi(\mathcal{I}) = \mathcal{K}$ then $\tau(K, L) = \mathcal{K}$.

Historical remark. In the late nineties, through the work of Van Gelder, Fitting and Przymusiński, it had become clear that considerable sections of Logic Programming semantics could be “algebraized”. For example, Fitting had defined his three- and four-valued immediate consequence operator Ψ^Π to characterize Kripke-Kleene fixpoints. He had also introduced the operator T_1^Π which had been extended to an operator mapping pairs of 4-valued interpretations to four-valued interpretations to define four-valued stable models. However, the missing link was the relationship between Ψ^Π and T_1^Π as expressed by Proposition 8. This result is the cornerstone that allows to push all semantic constructions completely to the algebraic setting with the four-valued immediate consequence operator in the centre. This is the theory that was elaborated in [3].

To the authors of [3], the existence of this algebraic relationship between Ψ^Π and T_1^Π came as a big surprise. Indeed, such a relationship seemed impossible given that the definitions of T_1^Π and the stable operator explicitly depend on syntactic details such as the distinction between positive and negative literals; moreover the stable operator treats both in a completely asymmetrical way. On the other hand, the immediate consequence operator seem to treat positive and negative literals in a symmetrical way. It therefore seemed impossible that there could be an algebraic relationship between both operators. That it was possible after all, is due to the fact that the four-valued truth function can be decomposed in two two-valued ones using pos-neg evaluation. Pos-neg evaluation incorporates the asymmetric treatment of positive and negative literals w.r.t. different structures.

5.3 Some pragmatic consequences regarding the semantics

Equivalence. From a pragmatical viewpoint, equivalence is an important aspect in a logic. When human experts express properties of a domain in a theory,

there are typically several ways to express them. Different experts, or the same expert at different times, may express the same properties in different ways. This should lead to equivalent theories. For this to work well, it is necessary that the logic comes with a set of natural well-understood equivalence preserving laws. To illustrate the issue, assume that a condition in a program rule is that symbols P and Q are equivalent. There are two ways to represent this:

$$(P \rightarrow Q) \wedge (Q \rightarrow P) \quad \text{or} \quad (P \wedge Q) \vee (\neg P \wedge \neg Q)$$

In FO, these formulas are equivalent, so it does not matter which one is used. But in other logics, these two formulas are not equivalent and the human expert should select the right one to insert in his formula. Knowledge representation without a good understanding of the meaning of both would be seriously hampered.

In the context of the equilibrium semantics, equivalence of programs is based on equivalence in the logic of HT and is called *strong equivalence* [13]: substituting a formula in a body with one that is equivalent (i.e., has the same HT truth value in each pair (I, J)) is equivalence preserving. Most of the standard equivalence preserving transformations in FO do not hold in HT. E.g., the above two formulas for expressing $P \leftrightarrow Q$ are not strongly equivalent. Strong equivalence preserving transformations were investigated in [5, 14].

In the context of the semantics presented here, the issue of equivalence has not yet so thoroughly been investigated. Yet, several things are obvious. The underlying equivalence in each of the family of semantics considered here is four-valued equivalence: substituting a formula in a rule body by one that has the same truth value in all four-valued structures preserves the four-valued immediate consequence operator and hence, the four types of semantics. The good news is that almost all standard basic equivalence preserving transformations in FO also hold in four-valued logic: e.g., removing double negation, commutativity of $\wedge, \vee, \leftrightarrow$, distribution laws of $\wedge, \vee, \exists, \forall$ over other connectives, laws of De Morgan. It is easy to check that the left and righthand side of these formulas have the same truth value in all four-valued interpretations. This is important because these principles capture some of the most fundamental “laws of thought” of human thinking. In contrast, few of these preserve strong equivalence in HT. Standard transformations towards negation normal form or prenex normal form are all equivalence preserving in four-valued logic and hence, can be applied to rule bodies.

Transformations that are not equivalence preserving in four-valued logic are those that introduce reasoning by cases. E.g., the following FO tautologies that include some form of case-based reasoning do not yield equivalence preserving transformations in four-valued logic: e.g., $\varphi \leftrightarrow \varphi \wedge (\psi \vee \neg\psi)$, $\varphi \leftrightarrow \varphi \vee (\psi \wedge \neg\psi)$ or $\varphi \leftrightarrow (\psi \rightarrow \varphi) \wedge (\neg\psi \rightarrow \varphi)$. This explains why the following programs are not equivalent:

$$\Pi_1 = \{p \leftarrow \mathbf{t}\} \text{ and } \Pi_2 = \{p \leftarrow p \vee \neg p\}$$

Proposition 9. *If Π is a program containing rule $\forall \bar{x}(P(\bar{t}) \leftarrow \varphi)$ and φ' is a formula that has the same truth value in each four-valued interpretation as φ ,*

then $\Pi \setminus \{\forall \bar{x}(P(\bar{t}) \leftarrow \varphi)\} \cup \{\forall \bar{x}(P(\bar{t}) \leftarrow \varphi')\}$ are equivalent in any of the four semantics.

All in all, the form of equivalence underlying the stable semantics defined here, which is the same for the three other semantics of the family, is much more “standard” than in HT. We believe that this is a considerably advantage from a pragmatic point of view.

Language extensions and abstract semantics. A challenge for a semantics is the ease with which it can be extended to handle new language constructs. A good example where such an extension proved to be quite difficult is the case of logic programming extended with recursion over aggregate expressions.

To accomodate for language extension, one solution is the use of an abstract, syntax independent semantics. This strategy was followed, e.g., in the context of answer set programming with abstract constraint atoms [15]. In this semantics, any new language construct can be added to rule bodies provided that its semantics can be expressed abstractly as “abstract constraint atoms”.

The semantics presented here in this paper goes even further in this abstract direction. All that is needed is an approximator on the bilattice, or equivalently, a monotone-antimonotone lattice operator. This is why the Approximation Fix-point Theory is not only useful to describe LP semantics but also the semantics of Default logic and Autoepistemic reasoning [4].

What needs to be done to add a new language expression in rule bodies of programs (e.g., an aggregate expression)? To extend the family of semantics defined here, it suffices to define a four-valued immediate consequence operator Ψ^{Π} for the extended formalism. The extended operator should satisfy two conditions: it should be $\leq_p$ -monotone and it should be symmetric. This will be the case if we can extend the four-valued truth function $\varphi^{\mathcal{I},\theta}$ for the extended formalism of rule bodies such that the following conditions are satisfied:

- $\leq_p$ -monotonicity: if $\mathcal{I} \leq_p \mathcal{J}$, then $\varphi^{\mathcal{I},\theta} \leq_p \varphi^{\mathcal{J},\theta}$.
- symmetry: if $\mathcal{J}$ can be obtained from $\mathcal{I}$ by switching **i** and **u**, then $\varphi^{\mathcal{J},\theta}$ can be obtained from $\varphi^{\mathcal{I},\theta}$ by switching **i** and **u**. I.e., both are true, or both are false, or one is **u** and the other **i**.

Proposition 10. *If the four-valued truth function is $\leq_p$ -monotone and symmetric, so will be Ψ^{Π} .*

The proof is straightforward.

It is sometimes even simpler to explain the extension of the semantics in terms of the pos-neg truth evaluation. For the extended formalism of rule bodies, we define $\varphi^{\tau(I,J),\theta} = \tau(\varphi^{+I-J}, \varphi^{+J-I})$ as before and extend the pos-neg evaluation so that the following condition is satisfied:

- monotone-antimonotone: if $I \leq_t K$ and $J \geq_t L$, then $\varphi^{+I-J,\theta} \leq \varphi^{+K-L,\theta}$.

It follows from monotone-antimonotonicity that if $I \leq_t J$ then for every $I \leq_t K \leq_t J$, it holds that $\varphi^{+I-J,\theta} \leq \varphi^{+K,\theta} \leq \varphi^{+J-I,\theta}$. As a special case, the truth value $\varphi^{+I-I,\theta}$ denotes the truth value of φ under I, θ and will be denoted $\varphi^{I,\theta}$.

Proposition 11. *A monotone and symmetric four-valued truth function induces a monotone-antimonotone pos-neg truth valuation and vice versa.*

The proof is straightforward.

In practice, the pos-neg evaluation is defined by induction on the structure of the formula φ . What needs to be done is to insert a new case in this inductive definition, defining the truth value for the new language construct. Below we illustrate this for the cardinality aggregate. The definitions below stem from [18] where approximation fixpoint theory was used to define the stable and well-founded semantics for logic programs with arbitrary aggregates.

The new language constructs considered here are of the form

$$\#\{\bar{x} : \varphi\} \circ t$$

where $\circ \in \{=, <, >, \leq, \geq\}$, t a numerical term and $\{\bar{x} : \varphi\}$ a set expression.

We need to define $(\#\{\bar{x} : \varphi\} \circ t)^{+I-J, \theta}$. This is done in two steps.

- With the set expression $\{\bar{x} : \varphi\}$, we can associate two sets of tuples of domain elements of D :

$$\{\bar{d} \mid \varphi^{+I-J, \theta[\bar{x}:\bar{d}]} = \mathbf{t}\} \quad \text{and} \quad \{\bar{d} \mid \varphi^{+J-I, \theta[\bar{x}:\bar{d}]} = \mathbf{t}\}$$

In case $I \leq J$, the first will be a subset of the second, due to monotone-antimonotonicity. The first one is an underestimation of the approximated set, the second an overestimation.

- Depending on the operator $\circ$, we use one of these sets to compute the truth value of the expression. We illustrate this here for the operator $<$ and $>$. The truth function for the other operators can be defined in terms of these.

- Define $(\#\{\bar{x} : \varphi\} < t)^{+I-J, \theta} = \mathbf{t}$ iff the number of elements in the set

$$\{\bar{d} \mid \varphi^{+J-I, \theta[\bar{x}:\bar{d}]} = \mathbf{t}\}$$

is strictly smaller than t^I . Thus, for $I \leq_t J$, this value is true if the cardinality of the overestimated set is less than t^I .

- Define $(\#\{\bar{x} : \varphi\} > t)^{+I-J, \theta} = \mathbf{t}$ iff the number of elements in the set

$$\{\bar{d} \mid \varphi^{+I-J, \theta[\bar{x}:\bar{d}]} = \mathbf{t}\}$$

is strictly larger than t^I . Now for $I \leq_t J$, this value is true if the cardinality of the underestimated set is already larger than t^I .

Proposition 12 ([18]). $\varphi^{+I-J, \theta}$ is monotone-antimonotone.

Thus, this pos-neg truth function induces a monotone-antimonotone operator T_1'' , a symmetric, $\leq_p$ -monotone four-valued truth function, an approximator and four types of semantics.

References

1. N. D. Belnap. A useful four-valued logic. In J. M. Dunn and G. Epstein, editors, *Modern Uses of Multiple-Valued Logic*, pages 8–37. Reidel, Dordrecht, 1977. Invited papers from the Fifth International Symposium on Multiple-Valued Logic, held at Indiana University, Bloomington, Indiana, May 13–16, 1975.
2. K. L. Clark. Negation as failure. In *Logic and Data Bases*, pages 293–322. Plenum Press, 1978.
3. M. Denecker, V. W. Marek, and M. Truszczyński. Approximating operators, stable operators, well-founded fixpoints and applications in non-monotonic reasoning. In *Logic-based Artificial Intelligence*, The Kluwer International Series in Engineering and Computer Science, pages 127–144. Kluwer Academic Publishers, Boston, 2000.
4. M. Denecker, V. W. Marek, and M. Truszczyński. Uniform semantic treatment of default and autoepistemic logics. *Artif. Intell.*, 143(1):79–122, 2003.
5. T. Eiter, M. Fink, H. Tompits, P. Traxler, and S. Woltran. Replacements in non-ground answer-set programming. In P. Doherty, J. Mylopoulos, and C. A. Welty, editors, *KR*, pages 340–351. AAAI Press, 2006.
6. S. Feferman. Toward useful type-free theories. *Journal of Symbolic Logic*, 49(1):75–111, 1984.
7. M. Fitting. A Kripke-Kleene semantics for logic programs. *Journal of Logic Programming*, 2(4):295–312, 1985.
8. M. Fitting. Fixpoint semantics for logic programming a survey. *Theoretical Computer Science*, 278(1-2):25–51, 2002.
9. M. Gelfond. Representing knowledge in A-Prolog. In A. C. Kakas and F. Sadri, editors, *Computational Logic: Logic Programming and Beyond*, volume 2408 of *LNCS*, pages 413–451. Springer, 2002.
10. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R. A. Kowalski and K. A. Bowen, editors, *ICLP/SLP*, pages 1070–1080. MIT Press, 1988.
11. T. Janhunen, E. Oikarinen, H. Tompits, and S. Woltran. Modularity aspects of disjunctive stable models. *J. Artif. Intell. Res. (JAIR)*, 35:813–857, 2009.
12. S. C. Kleene. *Introduction to Metamathematics*. Van Nostrand, 1952.
13. V. Lifschitz, D. Pearce, and A. Valverde. Strongly equivalent logic programs. *ACM Trans. Comput. Log.*, 2(4):526–541, 2001.
14. F. Lin and Y. Chen. Discovering classes of strongly equivalent logic programs. *J. Artif. Intell. Res. (JAIR)*, 28:431–451, 2007.
15. V. W. Marek and M. Truszczyński. Logic programs with abstract constraint atoms. In *In Proceedings of the 19th National Conference on Artificial Intelligence (AAAI-04)*, pages 86–91. AAAI Press, 2004.
16. D. Pearce. A new logical characterisation of stable models and answer sets. In J. Dix, L. M. Pereira, and T. C. Przymusiński, editors, *NMELP*, volume 1216 of *Lecture Notes in Computer Science*, pages 57–70. Springer, 1997.
17. D. Pearce. Equilibrium logic. *Ann. Math. Artif. Intell.*, 47(1-2):3–41, 2006.
18. N. Pelov, M. Denecker, and M. Bruynooghe. Well-founded and stable semantics of logic programs with aggregates. *Theory and Practice of Logic Programming (TPLP)*, 7(3):301–353, 2007.
19. T. C. Przymusiński. The well-founded semantics coincides with the three-valued stable semantics. *Fundamenta Informaticae*, 13(4):445–463, 1990.
20. A. Van Gelder. The alternating fixpoint of logic programs with negation. *Journal of Computer and System Sciences*, 47(1):185–221, 1993.

21. A. Van Gelder, K. A. Ross, and J. S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, 1991.
22. J. Vennekens, M. Mariën, J. Wittocx, and M. Denecker. Predicate introduction for logics with a fixpoint semantics. Part I: Logic programming. *Fundamenta Informaticae*, 79(1-2):187–208, September 2007.