Well-Founded Semantics Coincides With Three-Valued Stable Semantics

Teodor Przymusinski*
Department of Mathematical Sciences
The University of Texas at El Paso
teodor%utep.uucp@cs.utexas.edu

September 5, 1989

Abstract

We introduce *3-valued stable models* which are a natural generalization of standard (2-valued) *stable models*. We show that every logic program P has at least one 3-valued stable model and that the *well-founded model* of any program P [VGRS90] coincides with the *smallest 3-valued stable model* of P. We conclude that the well-founded semantics of an arbitrary logic program *coincides* with the 3-valued stable model semantics.

The 3-valued stable semantics is closely related to non-monotonic formalisms in AI. Namely, every program P can be translated into a suitable autoepistemic (resp. default) theory $\hat{P}$ so that the 3-valued stable semantics of P coincides with the (3-valued) autoepistemic (resp. default) semantics of $\hat{P}$. Similar results hold for circumscription and CWA. Moreover, it can be shown that the 3-valued stable semantics has a natural extension to the class of *all* disjunctive logic programs and deductive databases.

*The author acknowledges support from the National Science Foundation under grant #IRI-89-10729 and from the Army Research Office under grant #27079-EL-SAH.

Finally, following upon the recent approach developed by Gelfond and Lifschitz, we extend all of our results to more general logic programs which, in addition to the use of *negation as failure*, permit the use of *classical negation*.

1 Introduction

The *well-founded semantics* has been introduced in [VGRS90]. It is a *3-valued* semantics which seems to be the *most adequate* extension of the *perfect model semantics* [ABW88, VG89b, Prz88a] from the class of *stratified* logic programs to the class of *all* logic programs, avoiding various drawbacks of the other proposed approaches (see [PP90] for an overview). The well-founded semantics has been proven to share many of the natural properties of the perfect model semantics [Prz89a] and it has been shown to be equivalent to suitable (3-valued) forms of all four major *non-monotonic formalisms* [Prz91, Prz89b]. Recently, D. S. Warren introduced the Extended Warren Abstract Machine (XWAM) for this semantics [War89] and developed an elegant interpreter in Prolog.

The *stable model semantics* has been introduced in [GL88] (see also [BF88]). It is a 2-valued semantics, which also extends the perfect model semantics and has an elegant and simple fixed point definition. It is closely related to autoepistemic and default approaches to non-monotonic reasoning. However, the (2-valued) stable semantics also has some important drawbacks. First of all, it is defined only for a restricted class of logic programs; secondly, it is computationally very expensive, and, thirdly, it does not always lead to the expected (intended) meaning of the program [VGRS90, PP90].

Well-founded and stable models are closely related. It is known that if a logic program P has a 2-valued well-founded model than this model is the unique stable model of P [VGRS90]. On the other hand, however, there are programs with unique stable models, whose well-founded models are 3-valued and thus are not (2-valued) stable.

In this paper we introduce *3-valued stable models* of logic programs, which constitute a natural extension of (2-valued) stable models. We show that all such models are *minimal* and that *every* logic program P has at least one 3-valued stable model. Moreover, we prove that the *well-founded model* of any logic program P is, in fact, the *smallest 3-valued stable model* of P. As

a result, we conclude that the well-founded semantics of an arbitrary logic program coincides with the *3-valued stable model semantics*.

One of the important features of well-founded models, and a strong indication of their naturality, is the fact that they can be described in many different, but equivalent, ways (see [VGRS90, Prz89a, Prz91, VG89a, Bry89]). Our results provide a new and simple characterization of well-founded models as smallest 3-valued stable models. They also seem to point out the naturality and importance of stable models, at the same time indicating that the proper definition of stable models should be 3-valued.

The 3-valued stable model semantics not only provides a natural extension of the well-founded semantics, but it also naturally corresponds to *nonmonotonic formalisms* in AI (cf. [Prz89b, Prz88b]). Namely, every program P can be translated into an autoepistemic (resp. default) theory $\hat{P}$, so that the 3-valued stable semantics of P coincides with the (3-valued) autoepistemic (resp. default) semantics of $\hat{P}$ [Prz91]. Similar results hold for circumscription and CWA.

In [Prz90a, Prz90b] we prove that the 3-valued stable semantics has a natural extension to the class of *all disjunctive logic programs*, thus providing a natural and very general semantics for all disjunctive logic programs and deductive databases. The fact that the 3-valued stable semantics is well-defined for any normal program and can be extended to the class of all disjunctive logic programs [Prz90b] is very important. A logic program may contain predicates whose truth or falsity is *not fully determined* by the program (and thus is *undefined*), in addition to predicates whose truth value is *completely determined* by the program. Semantics which are well-defined only for limited classes of programs usually fail to assign any semantics to such programs.

To illustrate this point, let us consider the following program:

$$\begin{aligned} work &\leftarrow \sim tired \\ sleep &\leftarrow \sim work \\ tired &\leftarrow \sim sleep \\ angry &\leftarrow \sim paid, work \\ paid &\leftarrow \end{aligned}$$

It appears that the first three rules describe only mutual relationships between propositions $tired, work$ and $sleep$, without providing sufficient in-

formation to determine their truth or falsity. Depending on the point of view, we could describe our knowledge about propositions $tired, work$ and $sleep$ as either incomplete or perhaps even confusing. On the other hand, regardless of the status of propositions $tired, work$ and $sleep$, the proposition $paid$ must be true and thus $angry$, by negation as failure, must be false.

This leads to the unique 3-valued stable model $M =< paid;\ angry >$ of the program, in which $paid$ is true, $angry$ is false and $tired,\ work$ and $sleep$ are undefined. If we later learn, e.g., that $work$ is actually true then we will conclude that $sleep$ is false and $tired$ is true, but our beliefs about $paid$ and $angry$ will remain unchanged.

It is worth noting, that Prolog would return the same answers. Similarly, Fitting and Kunen's 3-valued extension of Clark's semantics [Fit85, Kun87], which should really be viewed as the "true Clark's semantics", leads precisely to the same result, namely, M is the *only* (3- or 2-valued) model of Clark's completion of the program. On the other hand, it is easy to see that the *2-valued* stable semantics, applied to this program, is undefined not only making it impossible for us to find out that we don't have a complete information about propositions $tired,\ work$ and $sleep$, but also, more importantly, denying us the ability to establish well-defined truth values of predicates $paid$ and $angry$.

The need to consider 3-valued models (possible worlds) to describe our knowledge stems also from the fact that our knowledge about the world is almost always incomplete and therefore we need the ability to describe possible worlds (models) in which some facts are neither true nor false and thus their status is undefined. Three-valued semantics have a realistic, computationally based proof theory and can be naturally implemented in various inference engines. Namely, the *SLS-resolution* [Prz89a] provides a sound and complete procedural mechanism for the 3-valued stable (or well-founded) semantics. The Extended Warren Abstract Machine (XWAM), introduced in [War89], also provides a procedural semantics for the 3-valued stable (or well-founded) semantics. Moreover, recently D. S. Warren developed an elegant interpreter for the well-founded semantics written in Prolog.

Clark's predicate completion semantics and its 3-valued extensions [Cla78, Llo84, Fit85, Kun87], which are considered by many researchers to be *too weak* [PP90], can be viewed as natural and computationally less expensive *approximations* to the intended 3-valued stable (or well-founded) semantics, in the sense that any answers given by the (3-valued extensions of) Clark

semantics are correct with respect to the (3-valued) stable semantics but not vice versa.

The negation operator $\sim$ that is used in standard logic programs does not represent the *classical negation*, but rather the so called *negation as failure*. For example, all major semantics applied to the logic program $a \leftarrow \sim b$ imply $\sim b$ (and thus also a), based on the *lack of evidence* that b holds. This is much weaker than the requirement of *positive evidence* that the negation $\neg b$ of b holds, which is needed to assert classical negation of b.

Gelfond and Lifschitz pointed out [GL89] that in logic programming it is often useful to use the negation as failure operator ($\sim$) together with a different negation operator ($\neg$), which is supposed to constitute a rough counterpart of classical negation[1]. They developed a semantics for such programs, based on their (2-valued) stable models, which is defined for some but not all such programs. Following upon their approach, we define the *well-founded* and the *3-valued stable* semantics for *all* such extended programs and we show that both semantics coincide. This allows us to extend all results obtained in this paper to the class of *programs permitting both types of negation*.

The paper is organized as follows. In the next Section 2 we introduce 3-valued interpretations and models. In Section 3 we define and discuss 3-valued stable models. In Section 4 we prove some of their properties, including the equivalence of the well-founded and 3-valued stable semantics. In Section 5 we discuss the relationship of the 3-valued stable semantics to non-monotonic formalisms. In Section 6 we extend our results to programs permitting the use of both negations $\sim$ and $\neg$.

The results contained in this paper were announced in [Prz90a]. The extension of the 3-valued stable semantics to all disjunctive programs is described in [Prz90a, Prz90b]. For an overview of semantic issues in logic programming and theory of deductive databases, the reader is referred to [PP90].

2 Model Theory

Before defining 3-valued stable models we need to define 3-valued interpretations and models of logic programs. We closely follow the approach developed

[1]Gelfond and Lifschitz use the symbol *not*, instead of $\sim$, to denote the negation as failure.

in [Prz89a, PP90].

By an *alphabet* $\mathcal{A}$ of a first order language $\mathcal{L}$ we mean a (finite or countably infinite) set of *constant*, *predicate* and *function* symbols. In addition, any alphabet is assumed to contain a countably infinite set of *variable* symbols, connectives $(\wedge, \vee, \sim, \leftarrow)$, quantifiers $(\exists, \forall)$ and the usual punctuation symbols. Moreover, we assume that our language also contains propositions **t**, **u** and **f**, denoting the properties of being *true* (resp. *undefined* or *partially true*; resp. *false*). The *first order language* $\mathcal{L}$ over the alphabet $\mathcal{A}$ is defined as the set of all well-formed first order formulae that can be built starting from the atoms and using connectives, quantifiers and punctuation symbols in a standard way. An expression is called *ground* if it does not contain any variables. The set of all ground atoms of $\mathcal{A}$ is called the *Herbrand base* $\mathcal{H}$ of $\mathcal{A}$. The set of all ground terms of $\mathcal{A}$ is called the *Herbrand universe* $\mathcal{U}$ of $\mathcal{A}$. If G is a quantifier-free formula, then by its *ground instance* we mean any ground formula obtained from G by substituting ground terms from $\mathcal{U}$ for all variables. For a given formula G of $\mathcal{L}$ its *universal closure* or just *closure* $(\forall)G$ is obtained by universally quantifying all variables in G which are not bound by any quantifier.

If P is a program then, unless stated otherwise, we will assume that the alphabet $\mathcal{A}$ used to write P consists precisely of all the constant, predicate and function symbols that explicitly *appear* in P and thus $\mathcal{A} = \mathcal{A}_P$ is completely determined[2] by the program P. We can then talk about the first order *language* $\mathcal{L} = \mathcal{L}_P$ *of the program* P and the *Herbrand base* $\mathcal{H} = \mathcal{H}_P$ *of the program.*

Definition 2.1 *By a* 3-valued Herbrand interpretation *I of the language $\mathcal{L}$ we mean any pair $< T; F >$, where T and F are disjoint subsets of the Herbrand base $\mathcal{H}$. The set T contains all ground atoms true in I, the set F contains all ground atoms false in I and the truth value of the remaining atoms in $U = \mathcal{H} - (T \cup F)$ is undefined (or undefined). We assume that in every interpretation I the proposition* **t** *is true, the proposition* **f** *is false and the proposition* **u** *is undefined. A 3-valued interpretation I is 2-valued if all ground atoms (except for the proposition* **u***) are either true or false in I.*

Throughout the paper, we consider only *Herbrand* interpretations and models, although our results can be easily extended to non-Herbrand models.

[2]If there are no constants in P then one is added to the alphabet.

Any interpretation $I =< T; F >$ can be equivalently viewed as a function $I : \mathcal{H} \to \{0, \frac{1}{2}, 1\}$, from the Herbrand base $\mathcal{H}$ to the 3-element set $\mathcal{V} = \{0, \frac{1}{2}, 1\}$, defined by:

$$I(A) = \begin{cases} 0, & \text{if } A \in F \\ \frac{1}{2}, & \text{if } A \in U \\ 1, & \text{if } A \in T. \end{cases}$$

We now extend the function (interpretation) $I : \mathcal{H} \to \mathcal{V}$ recursively to the truth valuation $\hat{I} : \mathcal{C} \to \mathcal{V}$ defined on the set $\mathcal{C}$ of all closed formulae of the language.

Definition 2.2 *[Prz89a] If I is an interpretation, then the* truth valuation *$\hat{I}$ corresponding to I is a function $\hat{I} : \mathcal{C} \to \mathcal{V}$ from the set $\mathcal{C}$ of all (closed) formulae of the language to $\mathcal{V}$ recursively defined as follows:*

- *If A is a ground atom, then $\hat{I}(A) = I(A)$.*
- *If S is a closed formula then $\hat{I}(\sim S) = 1 - \hat{I}(S)$.*
- *If S and V are closed formulae, then*

$$\begin{aligned} \hat{I}(S \wedge V) &= min(\hat{I}(S), \hat{I}(V)); \\ \hat{I}(S \vee V) &= max\{\hat{I}(S), \hat{I}(V)\}; \\ \hat{I}(V \leftarrow S) &= \begin{cases} 1, & \textit{if } \hat{I}(V) \geq \hat{I}(S) \\ 0, & \textit{otherwise.} \end{cases} \end{aligned}$$

- *For any formula $S(x)$ with one unbounded variable x:*

$$\begin{aligned} \hat{I}(\forall x\ S(x)) &= min\{\hat{I}(S(A)) : A \in \mathcal{U}\}; \\ \hat{I}(\exists x\ S(x)) &= max\{\hat{I}(S(A)) : A \in \mathcal{U}\}; \end{aligned}$$

  where the maximum (resp. minimum) of an empty set is defined as 0 (resp. 1).

Remark 2.1 Truth valuations assign to every formula F a number $0, \frac{1}{2}$ or 1, which reflects the *degree of truth* of F, ranging from the lowest, namely *false* (0), through *undefined* ($\frac{1}{2}$), to the highest, namely *true* (1). Here, the intuitive meaning of the *undefined* truth value is *partially true* or *possible*, rather than *either true or false*. Therefore, the undefined status of an atom A in a given model M of a theory T indicates that M assigns some, but only limited, truth to A.

By a *logic program* we mean a set of universally closed *clauses* of the form

$$A \leftarrow L_1 \wedge \ldots \wedge L_m$$

where $m \geq 0$, A's is an atom and L_i's are literals. For consistency reasons, we will not allow propositions **t**, **u** and **f** to appear in heads of clauses[3]. Conforming to a standard convention, conjunctions are replaced by commas and therefore clauses are simply written in the form

$$A \leftarrow L_1, \ldots, L_m.$$

A program P is *positive* if all of its clauses contain only positive premises.

Definition 2.3 *An (Herbrand) interpretation M is a model of a program P if all of its clauses are true in M, i.e., if for every ground instance*

$$A \leftarrow L_1, \ldots, L_m$$

of a program clause we have

$$\hat{M}(A) \geq min\{\hat{M}(L_i) : i \leq m\}.$$

Thus, M is a model of a program if and only if the degree of truth of the head of every clause is at least as high as the degree of truth of its body (i.e., the conjunction of its premises).

By a *ground instantiation* of a logic program P we mean the (possibly infinite) theory consisting of all ground instances of clauses from P. It is easy to see, that an Herbrand interpretation M is a model of a program P if and only if it is a model of its ground instantiation. Therefore, as long as only Herbrand interpretations are considered, one can identify any program P with its ground instantiation. Whenever convenient, we will assume, without further mention, that the program P has already been instantiated.

There are *two natural orderings* between interpretations, one of them, $\preceq$, is called the *standard ordering* and the other, $\preceq_F$, is called the *F-ordering* (or Fitting-ordering).

[3]This assumption is not essential and can be removed, but for the sake of simplicity this issue will not be discussed here.

Definition 2.4 *[Prz89a] Suppose that* $I =< T; F >$ *and* $J =< T'; F' >$ *are two interpretations. We say that* $I \preceq J$ *if*

$$T \subseteq T' \quad and \quad F \supseteq F'.$$

We say that $I \preceq_F J$ *if*

$$T \subseteq T' \quad and \quad F \subseteq F'.$$

Models which are $\preceq$*-minimal or* $\preceq$*-least will be just called minimal or least models, respectively. On the other hand, models which are* $\preceq_F$*-minimal or* $\preceq_F$*-least will be called F-minimal or F-least, respectively. F-least models will also be called* smallest *models.*

It is easy to verify that $I \preceq J$ if and only if for all atoms A:

$$I(A) \leq J(A) \quad \text{(or, equivalently, } \hat{I}(A) \leq \hat{J}(A)\text{)}.$$

The notions of *F-minimal* and *F-least* models are different from the notions of *minimal* and *least* models. While minimal and least models of a program P minimize the *degree of truth* of atoms, by minimizing the set T of true atoms and maximizing the set F of false atoms F, F-minimal and F-least models minimize the *degree of information* of their atoms, by jointly minimizing the sets T and F of atoms which are either true or false and thus maximizing the set U of unknown atoms. For example, the F-least (or smallest) model of the program $p \leftarrow p$ is obtained when p is undefined, while the least model of P is obtained when p is false.

3 Three-Valued Stable Models

In this section we define *3-valued stable models*. Our definition is similar to the definition of (2-valued) stable models given in [GL88]. Every 2-valued stable model is a 3-valued stable model, so our definition extends the notion of stable models.

First, we need the following generalization of the Kowalski-Van Emden Theorem [VEK76]:

Theorem 3.1 *Every positive logic program P has a unique least 3-valued model.*

The above theorem is a *strict* generalization of the Kowalski-Van Emden theorem in view of the fact that our positive logic programs are allowed to contain atoms **t**, **u** and **f** among their premises (only the occurence of atoms **u** is essential). In particular, least models of such programs may not be 2-valued.

Example 3.1 Suppose that P is given by:

$$\begin{aligned} c &\leftarrow \\ a &\leftarrow c, \mathbf{u} \\ b &\leftarrow b, \mathbf{u} \end{aligned}$$

The least model of P is $M =< c; b >$, i.e., in M the atom c is true, b is false and a is unknown.

In order to prove Theorem 3.1 we will first introduce a natural operator Ψ, which acts on the set of all 3-valued interpretations of a program and generalizes the Van Emden-Kowalski *immediate consequence operator* [VEK76]. We will then show that the least model of a positive program can be obtained as the least fixed point of this operator.

Definition 3.1 *Suppose that P is a logic program, I is a 3-valued interpretation of P and A is a ground atom. Define $\Psi(I)$ to be the interpretation given by:*

(i) $\Psi(I)(A) = 1$ *if there is a clause* $A \leftarrow A_1, \ldots, A_n$ *in* P *such that* $I(A_i) = 1$, *for all* $i \leq n$;

(ii) $\Psi(I)(A) = \frac{1}{2}$ *if* $\Psi(I)(A) \neq 1$ *and there is a clause* $A \leftarrow A_1, \ldots, A_n$ *in* P *such that* $I(A_i) \geq \frac{1}{2}$, *for all* $i \leq n$;

(iii) $\Psi(I)(A) = 0$, *otherwise.*

Theorem 3.1 is an immediate consequence of the following Theorem generalizing the well-known result from [VEK76].

Theorem 3.2 *If P is a positive program, then the operator Ψ has the least fixed point M_P, i.e., there exists the least interpretation M_P such that $\Psi(M_P) = M_P$. The interpretation M_P is the least model of P.*

Moreover, the model M_P can be obtained by iterating ω times the operator Ψ. More precisely, the sequence $\Psi^{\uparrow n}$, $n = 0, 1, 2, \ldots, \omega$, of iterations[4] of Ψ is monotonically increasing and it has a fixed point $\Psi^{\uparrow\omega} = M_P$.

Proof: The proof is completely analogous to the proof given in [VEK76] and therefore it is skipped. □

We now introduce the operator Γ^* which assigns to every 3-valued interpretation I a new 3-valued interpretation $\Gamma^*(I)$. This operator extends the Gelfond-Lifschitz transformation [GL88] to 3-valued logic programs.

Definition 3.2 *Let P be a logic program and let I be any 3-valued interpretation. By the* extended GL-transformation of P modulo I *we mean the new program $\frac{P}{I}$ obtained from P by replacing in every clause of P all negative premises $L = \sim C$ which are true (resp. unknown; resp. false) in I by atoms* **t** *(resp.* **u***; resp.* **f***). Since the resulting program $\frac{P}{I}$ is positive thus, by Theorem 3.1, it has a unique least 3-valued model J. We define $\Gamma^*(I) = J$.*

It is easy to see that if **t** appears among the premises then it can be simply erased and if **f** appears among the premises of a given clause, then the whole clause can be erased without changing anything. On the other hand, **u**'s in general cannot be removed. It turns out that *fixed points* of the operator Γ for a program P are always minimal models of P.

Proposition 3.1 *Fixed points of the operator Γ for a program P are minimal models of P.*

Proof: Let M be a fixed point of the operator Γ, i.e., let $\Gamma(M) = M$. Then M is the least model of the reduced program $\frac{P}{M}$. Let $\mathcal{C}$:

$$A \leftarrow B_1 \wedge \ldots \wedge B_m, \sim C_1 \wedge \ldots \wedge \sim C_n$$

be an arbitrary clause from P. The corresponding clause $\mathcal{C}'$ in $\frac{P}{M}$ is obtained by replacing all negative premises $L_i = \sim C_i$ which are true (resp. unknown; resp. false) in M by atoms **t** (resp. **u**; resp. **f**) and clearly $\mathcal{C}'$ must be satisfied in M. This immediately implies that also $\mathcal{C}$ must be satisified in M, thus showing that M is a model of P.

[4]With respect to the standard ordering $\preceq$ of interpretations and beginning from the least interpretation $< \emptyset, \mathcal{H} >$. Here ω denotes the first infinite ordinal.

To show that M is a minimal model of P it suffices to show that any model N of P, which is less than or equal to M (i.e., $N \preceq M$) is also a model of $\frac{P}{M}$. Since M is the least model of $\frac{P}{M}$, this will imply that $M = N$. Let $\mathcal{C}$, as above, be an arbitrary clause in P and let $\mathcal{C}'$ be the reduced clause in $\frac{P}{M}$ and C'' the reduced clause in $\frac{P}{N}$. Since N is a model of P it must clearly satisfy the clause $\mathcal{C}''$. Moreover, since $N \preceq M$ and the premises $\sim C_i$ are negative N must also satisfy $\mathcal{C}'$. This shows that N must be a model of $\frac{P}{M}$. $\square$

Fixed points of the operator Γ^* are defined as 3-valued stable models of P.

Definition 3.3 *A 3-valued interpretation M of a logic program P is called a* 3-valued stable model *of P if $\Gamma^*(M) = M$. Thus M is a 3-valued stable model of P if and only if it is the least model of $\frac{P}{M}$.*

The 3-valued stable model semantics *$STABLE3(P)$ of a program P is determined by the set $STBMOD(P)$ of all 3-valued stable models of P: a sentence F is true in $STABLE3(P)$ if and only if it is true in all models from $STBMOD(P)$.*

The above definition of 3-valued stable models is a strict extension of the original definition of stable models. In fact, standard stable models coincide with 2-valued stable models introduced above.

Proposition 3.2 *For any logic program P, stable models, as introduced in [GL88], coincide with 2-valued stable models, introduced above.*

Proof: As we observed before, if $\mathbf{t}$ appears among the premises of a given clause then it can be simply erased and if $\mathbf{f}$ appears among the premises then the whole clause can be erased without changing anything. This immediately shows that if the stable model is 2-valued then the definition given above coincides with the definition given in [GL88]. $\square$

Let us consider the example discussed in the introduction:

Example 3.2 Suppose that P is:

$$\begin{aligned} work &\leftarrow \sim tired \\ sleep &\leftarrow \sim work \\ tired &\leftarrow \sim sleep \\ angry &\leftarrow work, \sim paid \\ paid &\leftarrow \end{aligned}$$

and let $M =< paid;\ angry >$. Then the transformed program $\frac{P}{M}$ is:

$$\begin{aligned} work &\leftarrow \mathbf{u} \\ sleep &\leftarrow \mathbf{u} \\ tired &\leftarrow \mathbf{u} \\ angry &\leftarrow work, \mathbf{f} \\ paid &\leftarrow \end{aligned}$$

and its least model coincides with M which shows that M is a 3-valued stable model of P. As we mentioned before, the above program does not have any 2-valued stable models. It also illustrates differences between the 3-valued stable (well-founded) semantics and the semantics recently introduced in [BLM90], which implies, e.g., $work \vee sleep$ and generally treats clauses of the form $A \leftarrow \sim B$ as disjunctions $A \vee B$, which our semantics does not do.

In general, a logic program may have more than one 3-valued stable model (we will see in the next section that it always has at least one). For example the program:

$$\begin{aligned} a &\leftarrow \sim b \\ b &\leftarrow \sim a \end{aligned}$$

has three stable models, two of which are 2-valued. In one of them a is true and b false, in the other b is true and a false and in the third both a and b are undefined. When originally defining the stable model semantics Gelfond and Lifschitz considered only those programs which have a unique (2-valued) stable model. We do not make any such assumption.

Remark 3.1 It seems that most researchers agree that the truth value of the proposition A in the logic program:

$$A \leftarrow \sim A$$

should *not* be defined as *true* and stress the fact that neither the connective $\leftarrow$ nor the negation symbol $\sim$ represent classical implication and negation. Consequently, those researchers either propose not to assign any semantics to programs with this kind of negative recursion or to use 3-valued semantics and assign to the proposition A the value *undefined*. The last solution

seems superior, because it allows us to assign reasonable semantics to more programs, and thus prevents us from loosing useful information contained in the program (see the example discussed in the introduction). This is the solution adopted by the well-founded semantics and also by Fitting and Kunen [Fit85, Kun87] in their 3-valued extension of Clark's semantics.

At the same time a number of researchers (e.g., [BLM90, BS90]) argue that a program P of the form:

$$A \leftarrow \sim B$$

$$B \leftarrow \sim A$$

should somehow be viewed as representing a disjunction $A \vee B$ and thus, when augmented with clauses:

$$C \leftarrow A$$

$$C \leftarrow B$$

it should imply C.

The well-founded (or 3-valued stable) semantics, as well as Fitting-Kunen's semantics assign undefined value to C and we believe that this is a correct thing to do. We explain this claim below:

- First of all, if we really wanted to express disjunction, we should do so explicitely, rather than use this conspicuous way.

- Secondly, the only reason why we should view the above clauses as representing a disjunction $A \vee B$ is that we view the world as "black or white" (2-valued), i.e., we assume that everything, is always known to be either true or false. We argue then that since A must be either true or false, B must be, respectively, false or true and, consequently, either A or B must always be true. However, the assumption that our knowledge about the world is always "black or white" (2-valued) is not only unintuitive, but it also prevents us, as pointed out above, from assigning any sensible meaning to large classes of programs and thus should be rejected. Once we agree that portions of our knowledge might be undefined, there is no longer any reason to view the program P above as somehow representing a disjuction $A \vee B$, because it is quite possible that both A and B might be undefined (and so can be C).

- Finally, if for any propositions A and B the program

$$A \leftarrow \sim B$$
$$B \leftarrow \sim A$$
$$C \leftarrow A$$
$$C \leftarrow B$$

  is somehow to be viewed as representing a disjunction $A \vee B$ and imply C then, in particular, this should be true for A equal to B. However, this immediately leads to the conclusion that the program:

$$A \leftarrow \sim A$$
$$C \leftarrow A$$

  should imply both C and A, which contradicts the view that the clause $A \leftarrow \sim A$ is not equivalent to A.

Although the class of 3-valued stable models extends the class of 2-valued stable models, the 3-valued stable model semantics *does not* extend the (2-valued) stable model semantics. In fact, as we will see in the next section, the 3-valued stable model semantics coincides with the well-founded semantics, which is known to differ from the 2-valued stable semantics.

Example 3.3 [[VGRS90]] Let P be given by:

$$\begin{aligned} b &\leftarrow \sim a \\ a &\leftarrow \sim b \\ p &\leftarrow \sim p \\ p &\leftarrow \sim a. \end{aligned}$$

The above program has a unique (2-valued) stable model $M = \{p, b\}$. Therefore, in spite of the fact that the information encoded in the above program is rather confusing or incomplete, the (2-valued) stable semantics categorically implies that p and b are true and a is false, which is considered by many researchers to be unintuitive [VGRS90].

But P also has two 3-valued stable models $< a;\ b >$ and $< \emptyset; \emptyset >$. The latter model $< \emptyset; \emptyset >$ coincides with the well-founded model of P. The 3-valued stable semantics, which implies only those formulae which are satisfied in *all* 3-valued stable models, coincides with the well-founded semantics and correctly assigns undefined values to all a, b and p.

4 Well-Founded Semantics Coincides With Three-Valued Stable Semantics

The main result of this section shows that the well-founded model M_P of any program P is the *smallest (i.e., F-least) stable model* of P. This clearly implies that every logic program has at least one 3-valued stable model, namely M_P. Moreover, it also implies that the well-founded semantics of any program P coincides with the 3-valued stable model semantics of P.

Theorem 4.1 *Every normal logic program P has the smallest (i.e., F-least) 3-valued stable model. Moreover, this model always coincides with the well-founded model M_P of P.*

In other words, the well-founded model M_P is the smallest stable model of P, in the sense that, of all stable models, M_P contains the least number of true or false facts, and, thus, the largest set of undefined facts. We can say, borrowing from Horty and Thomasson's inheritance network terminology, that the well founded model is the most *skeptical* 3-valued stable model or possible world for P. For example, if P is given by $a \leftarrow \sim b,\ b \leftarrow \sim a$, then, as we have seen before, P has three stable models. One, in which a is true and b is false, the second, exactly opposite and the third in which both a and b are undefined. The last model, the most 'skeptical' one, is the well-founded model of P. Similarly, if P is the program from Example 3.3, then the 'most undefined' stable model of P is well-founded. This can be explained by saying that the well-founded semantics 'believes' only in those things which hold in all possible worlds (stable models) of the program.

Observe that although the above characterization of well founded models as F-least 3-valued stable models is mathematically elegant it does not provide any *constructive* way of finding such models. Constructive definitions of well-founded models were given in [Prz89a, VG89a].

Proof of Theorem 4.1: In order to prove Theorem 4.1 we first need to recall the definition of well-founded models. We combine here the definitions given in [VGRS90, Prz89a]. Suppose that P is a logic program and I is a 3-valued interpretation. We define two subsets T_I and F_I of the Herbrand base as follows:

- T_I is the smallest set of atoms A with the property that A is in T_I if there is a clause $A \leftarrow L_1, \ldots, L_n$ in P such that, for all $i \leq n$, either

$\hat{I}(L_i) = 1$ or $L_i = B_i$ is an atom and B_i is in T_I.

- F_I is the largest set of atoms A with the property that A is in F_I if for every clause $A \leftarrow L_1, \ldots, L_n$ in P there is an $i \leq n$ such that either $\hat{I}(L_i) = 0$ or $L_i = B_i$ is an atom and B_i is in F_I.

The sets T_I and F_I always exist. The well-founded model is defined recursively as follows. Let $M_0 =< \emptyset; \emptyset >$ and suppose that for every $\alpha < \beta$ interpretations $M_\alpha =< T_\alpha; F_\alpha >$ are already defined. Define:

$$M_\beta =< T_\alpha \cup T_{M_\alpha}; F_\alpha \cup F_{M_\alpha} >$$

if $\beta = \alpha + 1$ is a successor ordinal and

$$M_\beta =< \bigcup_{\alpha<\beta} T_\alpha; \bigcup_{\alpha<\beta} F_\alpha >$$

if β is a limit ordinal. The transfinite sequence M_α is clearly increasing and therefore there must exist the first ordinal λ such that $M_{\lambda+1} = M_\lambda$. Then $M_P = M_\lambda =< T_\lambda; F_\lambda >$ is defined as the well-founded model of P.

We first show that M_P is stable. Let $P' = \frac{P}{M_P}$ and suppose that $N =< T; F >$ is also a model of P' and that $N \preceq M$, i.e., $T \subseteq T_\lambda$ and $F \supseteq F_\lambda$. We will show that $M = N$. Suppose first that $T \neq T_\lambda$ and let β be the first ordinal such that there is an atom A such that $A \in T_\beta - T$. By the definition of T_β it follows immediately that β must be a successor ordinal, $\beta = \alpha + 1$. Moreover, we also have $T_\alpha \subseteq T$ and naturally $F_\alpha \subseteq F_\lambda \subseteq F$.

By definition, $T_\beta = T_\alpha \cup T_{M_\alpha}$. But T_{M_α} is the smallest set of atoms A with the property that A is in T_{M_α} if there is a clause $A \leftarrow L_1, \ldots, L_n$ in P such that, for all $i \leq n$, either $\hat{M}_\alpha(L_i) = 1$ or $L_i = B_i$ is an atom and B_i is in T_{M_α}. Therefore, T_{M_α} contains all those atoms that are logically implied by the program $P'' = \frac{P}{M_\alpha}$ (cf. [Prz89a]). Since the model N extends the interpretation M_α, i.e., $T_\alpha \subseteq T$ and $F_\alpha \subseteq F$, we conclude that they must also be logically implied by the program $\frac{P}{N}$ of which N is a model. This implies that $T_{M_\alpha} \subseteq T$ and thus $T_\beta \subseteq T$, which is a contradiction and proves that $T = T_\lambda$.

Suppose now that $F \supset F_\lambda$. Since N is a model of the positive program P' that means that for any $A \in F$ and for any clause in P' with A in its head there must exist a (positive) premise B_i which is false in N, i.e., such that $B_i \in F$. Consequently, for any $A \in F$ and for any clause in P with A in its

head there must either exist a positive premise $B_i \in F$ or a negative premise $\sim C_i$ such that C_i is true in N and thus $C_i \in T_\lambda$. From the definition of F_{M_λ} it follows that $F \subseteq F_{M_\lambda} \subseteq F_{\lambda+1} = F_\lambda$, which again is a contradiction.

It remains to show that the well-founded model M_P is the F-least stable model of P. Let $N =< T; F >$ be an arbitrary stable model of P. Therefore N is the least model of $\frac{P}{N}$. We will show by transfinite induction that:

$$(*) \quad T_\alpha \subseteq T \text{ and } F_\alpha \subseteq F$$

for every $\alpha \leq \lambda$. Clearly (*) is true for $\alpha = 0$. Assume that (*) is true for all $\alpha < \beta$. If β is a limit ordinal then clearly (*) holds for β. Assume therefore that $\beta = \alpha + 1$.

We first prove that $T_\beta \subseteq T$. By definition, $T_\beta = T_\alpha \cup T_{M_\alpha}$. But T_{M_α} is the smallest set of atoms A with the property that A is in T_{M_α} if there is a clause $A \leftarrow L_1, \ldots, L_n$ in P such that, for all $i \leq n$, either $\hat{M}_\alpha(L_i) = 1$ or $L_i = B_i$ is an atom and B_i is in T_{M_α}. Therefore, T_{M_α} contains all those atoms that are logically implied by the program $P'' = \frac{P}{M_\alpha}$. Since by the inductive assumption the model N extends the interpretation M_α, i.e., $T_\alpha \subseteq T$ and $F_\alpha \subseteq F$, we conclude that they must also be logically implied by the program $\frac{P}{N}$ of which N is a model. This implies that $T_{M_\alpha} \subseteq T$ and thus $T_\beta \subseteq T$.

We now prove that $F_\beta \subseteq F$. It suffices to show that $F_{M_\alpha} \subseteq F$. By definition, F_{M_α} is the largest set of atoms A with the property that A is in F_{M_α} if for every clause $A \leftarrow L_1, \ldots, L_n$ in P there is an $i \leq n$ such that either $\hat{M}_\alpha(L_i) = 0$ or $L_i = B_i$ is an atom and B_i is in F_{M_α}. Since N extends the interpretation M_α it easily follows that all of the atoms in F_{M_α} must be false in the least model of $\frac{P}{N}$. But N itself is the least model of $\frac{P}{N}$ and therefore all of the atoms in F_{M_α} must be false in N. This shows that $F_\beta \subseteq F$ and completes the proof of the theorem. □

Corollary 4.2 *Every logic program has at least one 3-valued stable model, namely, the well-founded model M_P.*

Consequently, as opposed to the (2-valued) stable semantics, the 3-valued stable semantics is *well-defined for any logic program.* As we stressed in the introduction, this is a very important property of the semantics.

Corollary 4.3 *The well-founded semantics of an arbitrary logic program P coincides with the 3-valued stable model semantics of P in the sense that for*

any sentence F (not containing the connective "$\leftarrow$"):

$$M_P \models F \quad \equiv \quad STABLE3(P) \models F.$$

Proof: Since the well-founded model is stable, if K is a literal and $STABLE3(P) \models K$ then $M_P \models K$. Conversely, if $M_P \models K$ then, since M_P is the F-least stable model of P and therefore it contains the least number of true or false facts of all stable models of P, K must also hold in all stable models of P. This shows that $STABLE3(P) \models K$. $\Box$

The above results stress the naturality and importance of stable and well-founded models, at the same time indicating that the proper definition of stable models should be 3-valued.

As a byproduct of our considerations we immediately obtain the important result from [VGRS90] relating 2-valued well-founded and stable models.

Corollary 4.4 *[VGRS90] If the well founded model of a program P is 2-valued then it coincides with the unique stable model of P.*

Proof: Well-founded models are F-least stable models of a program and therefore, if the well founded model of a program P is 2-valued, then it must be the only stable model of P. $\Box$

5 Relationship to Non-Monotonic Formalisms

It follows immediately from the results proved in [Prz91] that the 3-valued stable semantics of logic programs closely corresponds to non-monotonic formalisms. Namely, after a suitable translation of an arbitrary program P into an autoepistemic (resp. default) theory $\hat{P}$, the 3-valued stable semantics of P coincides with the (3-valued) autoepistemic (resp. default) semantics of $\hat{P}$ [Prz91].

We first need to translate logic programs into autoepistemic theories. We use the translation proposed in [Gel87].

Definition 5.1 *[Gel87] Let P be a logic program. The autoepistemic theory $\hat{P}$, which we call the* autoepistemic translation of P, *consists of all clauses of the form:*

$$A \leftarrow B_1, \ldots, B_m, \neg \mathbf{L}C_1, \ldots, \neg \mathbf{L}C_n,$$

for all possible (ground instances of) clauses:

$$A \leftarrow B_1, \ldots, B_m, \sim C_1, \ldots, \sim C_n$$

from P, where $\mathbf{L}$ *represents the autoepistemic belief operator.*

The following theorem is an immediate consequence of the results obtained in [Prz91]:

Theorem 5.1 (Equivalence of 3-valued stable and autoepistemic semantics) *There is a one-to-one correspondence between 3-valued stable models of P and 3-valued stable autoepistemic expansions of $\hat{P}$. In particular, the 3-valued stable semantics of P coincides with the (3-valued) autoepistemic semantics of $\hat{P}$.*

To obtain the correspondence with default logic we first have to translate a logic program P into a default theory $\hat{P}$. We use the approach proposed originally by Bidoit and Froidevaux [BF88]. First of all, the default translation $\hat{P}$ of P contains all clauses

$$A \leftarrow B_1, \ldots, B_m$$

from P, which do *not* have negative premises. Next, for every clause:

$$A \leftarrow B_1, \ldots, B_m, \sim \mathbf{L}C_1, \ldots, \sim \mathbf{L}C_n,$$

in P with some negative premises, the translation $\hat{P}$ contains a default rule:

$$\frac{B_1 \wedge \ldots \wedge B_n : \neg C_1, \ldots, \neg C_k}{A}$$

Finally, $\hat{P}$ contains defaults

$$\frac{: \neg A}{\neg A}$$

for every ground atom A in the language.

The following theorem also follows from the results obtained in [Prz91]:

Theorem 5.2 (Equivalence of 3-valued stable and default semantics) *There is a one-to-one correspondence between 3-valued stable models of P and 3-valued default extensions of $\hat{P}$. In particular, the 3-valued stable semantics of P coincides with the (3-valued) default semantics of $\hat{P}$.*

Similar relationships are valid between 3-valued stable semantics and circumscription and CWA [Prz91].

6 Programs with "Classical" Negation

As we mentioned in the introduction, Gelfond and Lifschitz pointed out [GL89] that in logic programming it is often useful to use the *negation as failure* operator ($\sim$) together with a different negation operator ($\neg$), which is supposed to constitute a rough counterpart of *classical negation*. They developed a semantics for such programs based on their (2-valued) stable models. For standard logic programs their semantics coincides with the (2-valued) stable model semantics and, consequently, it is not defined for all logic programs.

In this section, following upon Gelfond and Lifschitz's approach, we will define the *well-founded* and the *3-valued stable* semantics for *all* such extended programs with "classical" negation and we will show that both semantics coincide. This will allow us to extend the results obtained in this paper to the class of all programs permitting both types of negation.

As in [GL89], by an extended program with classical negation we will mean any (finite or infinite) set of clauses of the form:

$$L \leftarrow L_1, \ldots, L_m, \sim L_{m+1}, \ldots, \sim L_n$$

where $0 \leq m \leq n$ and L_i's are $\neg$-*literals*, i.e., L_i's are either atoms A or negated atoms $\neg A$, where the "classical" negation $\neg$ is used instead of the negation as failure $\sim$. Since we are only concerned with Herbrand models, as explained in Section 2, we can assume that all clauses are ground.

The only, yet essential, difference between standard logic programs and extended logic programs defined above is the fact that L_i's are allowed to be ("classically") negated literals $\neg A$ rather than just atoms. The class of extended programs is clearly broader than the class of standard programs.

Example 6.1 The following program P is an example of such an extended program:

$$\begin{aligned} innocent &\leftarrow charged, \neg guilty \\ \neg convicted &\leftarrow charged, \sim guilty \\ charged &\leftarrow \end{aligned}$$

The program states that if someone is charged then to know for sure that he is innocent one needs to know *for sure* that the person is not guilty (classical

negation). However, to deduce that someone is not convicted it is enough to know that he has not been proven guilty (negation as failure). Observe the use of classically negated atom "convicted" in the head of the second clause.

Now we extend the well-founded and 3-valued stable semantics to such programs with classical negation by using the following program transformation (cf. [GL89]):

- First, we rename all "classically" negated (ground) literals $\neg A$ by new atomic symbols, say, A' and make a suitable substitution everywhere in the program P. As a result we obtain a standard program P^* without classical negation.
- Then for any 3-valued stable model M^* of of the transformed program P^* (in particular, for the well-founded model) we define the corresponding 3-valued stable (or well-founded) model M of the extended program P as follows:
  - If A (resp. A') is true in M^*, then we we take A (resp. $\neg A$) to be true in M.
  - If A (resp. A') is false in M^*, then we we take $\sim A$ (resp. $\sim(\neg A)$) to be true in M, i.e., we view A (resp. $\neg A$) as false by default.
  - Otherwise, if A (resp. A') is undefined in M^*, then we consider the status of A (resp. $\neg A$) in M as also undefined.
- We throw out all *inconsistent* models M, i.e., those which contain both A and $\neg A$, for some atom A.

The transformed program P^* of the program P given in Example 6.1 looks as follows:

$$\begin{aligned} innocent &\leftarrow charged, guilty' \\ convicted' &\leftarrow charged, \sim guilty \\ charged &\leftarrow \end{aligned}$$

The transformed program P^* has exactly one stable (and well-founded) model M^* in which $convicted'$ and $charged$ are true and all the remaining atoms $innocent$, $innocent'$, $guilty$, $guilty'$, $charged'$ and $convicted$ are false.

This leads to the unique stable (and well-founded) model M of the extended program P described by the following table:

Atom A	guilty	innocent	charged	convicted
A	no	no	yes	no
$\neg A$	no	no	no	yes

In the above table, *yes* (resp. *no*) for a literal L means that L (resp. $\sim L$) is true in M. Thus, in M the individual is known to be charged and not convicted, but neither *innocent* nor *guilty* nor their classical negations, $\neg innocent$ or $\neg guilty$, can be proven and thus their negation (as failure) can be assumed.

Notice, that, L is true (resp. false) if and only if $\sim(L)$ is false (resp. true), for any $\neg$-literal L. This is a consequence of the fact that we require $\hat{I}(L) = 1 - \hat{I}(\sim L)$, for any $\neg$-literal L. On the other hand, observe that it is possible that neither A nor $\neg A$ holds in M_P and consequently, $\sim A$ and $\sim(\neg A)$ both are true. This simply means that neither A nor $\neg A$ are "classically" derivable, and therefore both can be assumed false by default. It also shows that the condition $\hat{I}(A) = 1 - \hat{I}(\neg A)$ is, in general, not satisfied for classical negation. The ability to distinguish between facts "classically" true and those which are true by virtue of negation as failure is the greatest strength of Gelfond and Lifschitz's approach.

Remark 6.1 Observe that the above described method applies to any semantics that we wish to choose for standard programs (i.e., programs without classical negation). This means that no matter what semantics we choose for standard programs we can immediately extend it to programs with classical negation. Thus the inclusion of classical negation does not require any new semantic considerations.

The resulting model M derived from M^* may turn out to be *inconsistent*, namely, it may contain both A and $\neg A$, for some atom A. This is caused by the fact that the atoms A and A' in the transformed program P^* are treated as independent atoms, even though one of them is really meant to represent the classical negation of the other. In this case we discard it. The following is an example of an extended program with resulting inconsistent model:

Example 6.2 Let P be as follows:

$$\begin{aligned} \neg a &\leftarrow \sim b \\ a &\leftarrow \neg a \end{aligned}$$

The transformed program P^* is:

$$\begin{aligned} a' &\leftarrow \sim b \\ a &\leftarrow a' \end{aligned}$$

and it has a unique well-founded (in fact, perfect) model $M^* =< \{a, a'\}; \{b, b'\} >$. Therefore, the resulting model M contains both A and $\neg A$ and thus it is inconsistent. In other words, M is inconsistent, because both a and a' are true in M^*.

Once we defined the well-founded and 3-valued stable models models for any extended program P we define the well-founded (or 3-valued stable) semantics of P as the set of all sentences satisfied all (consistent) models, where the satisfaction of sentences involving both $\sim$ and $\neg$, in an interpretation I, is defined similarly as in Section 2. We require, as before, that $\hat{I}(L) = 1 - \hat{I}(\sim L)$, for any $\neg$-literal L, but, for reasons explained above, we *do not* require $\hat{I}(A) = 1 - \hat{I}(\neg A)$, but instead treat A and $\neg A$ as separate literals which, as in the case of b in Example 6.1, both might be false or even, if I is inconsistent, both can be true.

Observe, that the 3-valued stable semantics may be well-defined, even though some of the 3-valued stable models of P turn out to be inconsistent:

Example 6.3 Suppose that P is given by:

$$\begin{aligned} \neg a &\leftarrow \\ a &\leftarrow \sim b \\ b &\leftarrow \sim a \end{aligned}$$

The transformed program P^* is:

$$\begin{aligned} a' &\leftarrow \\ a &\leftarrow \sim b \\ b &\leftarrow \sim a \end{aligned}$$

and it has three 3-valued stable models: $M_1^* =< \{a'\}; \{b'\} >$, $M_2^* =< \{a', a\}; \{b', b\} >$ and $M_3^* =< \{a', b\}; \{b', a\} >$. They give rise to three 3-valued stable models M_1, M_2 and M_3 of P. In the model M_2 both a and $\neg a$ hold and therefore M_2 is inconsistent, yet the 3-valued stable semantics of P is consistent, because a is not true in the remaining two models. The smallest (F-least) stable model M_1 of P again coincides with the well-founded model of P.

The following result extends the results obtained in the previous sections of the paper to the broader class of programs allowing "classical" negation. It is an almost immediate consequence of the above given definitions and the results proved before and therefore its proof will be omitted.

Theorem 6.1 *The well-founded (resp. 3-valued stable) models and semantics defined above extend the concepts of well-founded (resp. 3-valued stable) models and semantics from standard programs to the class of* all *programs with "classical" negation. The well-founded model always coincides with the smallest (F-least) 3-valued stable model. Consequently, the well-founded semantics always coincides with the 3-valued stable semantics.*

The above result immediately implies the following generalization of the result previously obtained in [VGRS90] establishing a relationship between 2-valued stable models of extended programs introduced in [GL89] and well-founded models of such programs introduced here.

Corollary 6.2 *If the well founded model of an extended program P is 2-valued then it coincides with the unique 2-valued stable model of P.*

Remark 6.2 It should be pointed out that the negation operator $\neg$ does not *truly* represent classical negation, but only constitutes its rough counterpart. Classically, the negation $\neg A$ of an atom A is implied by a given theory T if and only if A is false in all models of T. This certainly does not apply to the negation operator $\neg$ considered here. For example, the program

$$\begin{aligned} \neg a &\leftarrow \sim b \\ a &\leftarrow \neg a \end{aligned}$$

from Example 6.2 clearly has a (consistent) model in which b and a are true and $\neg a$ is false. Therefore, from the standpoint of classical logic, the

program neither is inconsistent nor implies $\neg b$. This illustrates the fact that the "classical" negation operator $\neg$ and the negation as failure operator $\sim$ are mutually dependent upon one another and as a result both of them are *non-monotonic* operators and thus neither one of them can be truly viewed as a classical, monotonic negation.

Moreover, even in programs which do *not* contain negation as failure the operator $\neg$ does not behave in a "classical" way (due to the special interpretation of the implication symbol $\leftarrow$). For example, the program:

$$\begin{array}{rcl} a & \leftarrow & \\ \neg a & \leftarrow & b \end{array}$$

classically implies $\neg b$ and yet neither b nor $\neg b$ is true in the unique stable or well-founded model of P.

Consequently, it is best to view $\neg$ as a non-monotonic negation operator different from the negation as failure operator $\sim$, which is supposed to constitute a rough counterpart of classical negation in non-monotonic theories.

References

[ABW88] K. Apt, H. Blair, and A. Walker. Towards a theory of declarative knowledge. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 89–142. Morgan Kaufmann, Los Altos, CA., 1988.

[BF88] N. Bidoit and C. Froidevaux. General logical databases and programs: Default logic semantics and stratification. *Journal of Information and Computation*, 1988. In print.

[BLM90] C. Baral, J. Lobo, and J. Minker. Generalized well-founded semantics for logic programs. In *10th International Conference on Automated Deduction, West Germany*, 1990.

[Bry89] F. Bry. Logic programming as constructivism: A formalization and its application to databases. In *Proceedings of the Symposium on Principles of Database Systems*, pages 34–50. ACM SIGACT-SIGMOD, 1989.

[BS90] C. Baral and V.S. Subrahmanian. Stable and extension class theory for logic programs and default logics. Research report, University of Maryland, 1990.

[Cla78] K.L. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum Press, New York, 1978.

[Fit85] M. Fitting. A Kripke-Kleene semantics for logic programs. *Journal of Logic Programming*, 2(4):295–312, 1985.

[Gel87] M. Gelfond. On stratified autoepistemic theories. In *Proceedings AAAI-87*, pages 207–211, Los Altos, CA, 1987. American Association for Artificial Intelligence, Morgan Kaufmann.

[GL88] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R. Kowalski and K. Bowen, editors, *Proceedings of the Fifth Logic Programming Symposium*, pages 1070–1080, Cambridge, Mass., 1988. Association for Logic Programming, MIT Press.

[GL89] M. Gelfond and V. Lifschitz. Logic programs with classical negation. Research report, UT El Paso and Stanford University, 1989.

[Kun87] K. Kunen. Negation in logic programming. *Journal of Logic Programming*, 4(4):289–308, 1987.

[Llo84] J.W. Lloyd. *Foundations of Logic Programming*. Springer Verlag, New York, N.Y., first edition, 1984.

[PP90] H. Przymusinska and T. C. Przymusinski. Semantic issues in deductive databases and logic programs. In R. Banerji, editor, *Formal Techniques in Artificial Intelligence*, pages 321–367. North-Holland, Amsterdam, 1990.

[Prz88a] T. C. Przymusinski. On the declarative semantics of stratified deductive databases and logic programs. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 193–216. Morgan Kaufmann, Los Altos, CA., 1988.

[Prz88b] T. C. Przymusinski. On the relationship between non-monotonic reasoning and logic programming. In *Proceedings AAAI-88*, pages 444–448, Los Altos, CA, 1988. American Association for Artificial Intelligence, Morgan Kaufmann. [The full version appeared in: T. C. Przymusinski. Non-Monotonic Reasoning vs. Logic Programming: A New Perspective. In *The Foundations of Artificial Intelligence. A Sourcebook*, D. Partridge and Y. Wilks, editors, Cambridge University Press, London, 1990, 49-71.].

[Prz89a] T. C. Przymusinski. Every logic program has a natural stratification and an iterated fixed point model. In *Proceedings of the Eighth Symposium on Principles of Database Systems*, pages 11–21. ACM SIGACT-SIGMOD, 1989.

[Prz89b] T. C. Przymusinski. Non-monotonic formalisms and logic programming. In G. Levi and M. Martelli, editors, *Proceedings of the Sixth International Logic Programming Programming Conference, Lisbon, Portugal*, pages 655–674, Cambridge, Mass., 1989. Association for Logic Programming, MIT Press.

[Prz90a] T. C. Przymusinski. Extended stable semantics for normal and disjunctive logic programs. In *Proceedings of the Seventh International Logic Programming Programming Conference, Jerusalem, Israel*, pages 459–477, Cambridge, Mass., 1990. Association for Logic Programming, MIT Press.

[Prz90b] T. C. Przymusinski. Stationary semantics for disjunctive logic programs and deductive databases. In *Proceedings of the North American Logic Programming Conference, Austin, Texas, October 1990*, pages 40–59, Cambridge, Mass., 1990. Association for Logic Programming, MIT Press.

[Prz91] T. C. Przymusinski. Three-valued non-monotonic formalisms and semantics of logic programs. *Journal of Artificial Intelligence*, 1991. (In print. Extended abstract appeared in: T. C. Przymusinski. Three-valued non-monotonic formalisms and logic programming. In R. Brachman, H. Leveque, and R. Reiter, editors, *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning (KR'89), Toronto, Canada*, pages 341–348, Morgan Kaufmann, 1989.).

[VEK76] M. Van Emden and R. Kowalski. The semantics of predicate logic as a programming language. *Journal of the ACM*, 23(4):733–742, 1976.

[VG89a] A. Van Gelder. The alternating fixpoint of logic programs with negation. In *Proceedings of the Symposium on Principles of Database Systems*, pages 1–10. ACM SIGACT-SIGMOD, 1989.

[VG89b] A. Van Gelder. Negation as failure using tight derivations for general logic programs. *Journal of Logic Programming*, 6(1):109–133, 1989.

Preliminary versions appeared in *Third IEEE Symp. on Logic Programming* (1986), and *Foundations of Deductive Databases and Logic Programming*, J. Minker, ed., Morgan Kaufmann, 1988.

[VGRS90] A. Van Gelder, K. A. Ross, and J. S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 1990. (to appear). Preliminary abstract appeared in Seventh ACM Symposium on Principles of Database Systems, March 1988, pp. 221–230.

[War89] David S. Warren. The XWAM: A machine that integrates Prolog and deductive database query evaluation. Technical report #25, SUNY at Stony Brook, 1989.