# Combining explicit negation and negation by failure via Belnap's logic ⋆

Paul Ruet, François Fages [1]

*Thomson - LCR, Domaine de Corbeville, 91404 Orsay, France,*
*LIENS, Ecole Normale Supérieure, 45 rue d'Ulm, 75005 Paris, France*

**Abstract**

This paper deals with logic programs containing two kinds of negation: negation as failure and explicit negation. This allows two different forms of reasoning in the presence of incomplete information. Such programs have been introduced by Gelfond and Lifschitz and called extended programs. We provide them with a logical semantics in the style of Kunen, based on Belnap's four-valued logic, and an answer sets' semantics that is shown to be equivalent to that of Gelfond and Lifschitz.

The proofs rely on a translation into normal programs, and on a variant of Fitting's extension of logic programming to bilattices.

## 1 INTRODUCTION

One of the striking features of logic programming is that it naturally supports non-monotonic reasoning by means of negative literals. Simply inferring negative information from a positive program is already a form of non-monotonic inference that shows essential differences between the two main approaches to the model-theoretic semantics of logic programs: namely the *standard model* approach and the *program's completion* approach.

In the standard model approach, the semantics of a positive program is identified to the least Herbrand model of the program. Then $\neg A$ must be inferred if $A$ is false in the least Herbrand model of the program (i.e. Closed World Assumption). In the program's completion approach, the clauses defining the

⋆ A preliminary version was presented at ILPS'94, Workshop on Uncertainty in Databases and Deductive Systems, Ithaca NY (Nov. 94).
[1] *Correspondence to*: Laboratoire d'Informatique de l'Ecole Normale Supérieure, 45 rue d'Ulm, 75005 Paris, France. E-mail: {ruet,fages}@dmi.ens.fr.

same predicate are read as a definition of the predicate using an equivalence connective in place of implications. Then $\neg A$ must be inferred if $\neg A$ is a logical consequence of the completion of the program.

From a programming language point of view, the standard model approach is not viable because it is untractable, namely the set of false atoms is not recursively enumerable. From a knowledge representation point of view however, standard models correspond naturally to the intended semantics of programs. Therefore the challenge is to provide constructs that capture the essential aspects of standard models, in a recursively enumerable setting.
In the framework of normal programs which allow negation inside program clause bodies, the stable models of [12] provide a general notion of standard model. Stable models however may not exist or may not be unique. Stratified and perfect models [3], are particular cases of stable models uniquely defined for restricted classes of normal programs. Three-valued standard models have also been defined to resolve the difficulty of existence and uniqueness of a standard model for normal programs. None of these notions of standard model for normal programs however is computable so any concrete operational semantics is necessarily incomplete.
On the other hand, the completion of a normal program may be inconsistent, e.g. with $P = \{p :- \neg p\}$, $P^* = \{p \leftrightarrow \neg p\}$, in which case any literal should be inferred. In order to resolve these difficulties, Kunen proposed to take the set of the consequences in three-valued logic of the program's completion as the declarative semantics of the program. In the previous example, taking the third truth value $u$ for $p$ provides a model of $P^*$ as $u \leftrightarrow \neg u$. Kunen proved a completeness result [18] for the negation as failure rule w.r.t. the three-valued completion of the program, followed by stronger completeness results for the constructive negation rule [19,7].

In this paper we study extended logic programs as introduced by Gelfond and Lifschitz [13,14] (see also [21,1]) to deal with two kinds of negation: explicit negation allowed in clause heads and bodies and negation by failure allowed in clause bodies only. These two negations allow two different forms of reasoning in the presence of incomplete information: to infer *not* $A$, you may want to know that $A$ cannot be inferred (it is the case of negation by failure $/A$), or you may require an explicit inference process for *not* $A$, when e.g. the closed world assumption cannot be made on $A$ (it is the case of explicit negation $\neg A$).
We study the existence of 4-valued Belnap's models for extended programs and develop a 9-valued Kunen-style semantics for extended programs. Because the negation as failure connective is not monotonic w.r.t. the knowledge ordering, our construction is not an instance of the bilattice extension of logic programming proposed by Fitting in [10], it corresponds rather to an extension of this framework to incorporate negation as failure: our $3\times3$ construction extends Fitting's programs on the $2\times2$ bilattice of Belnap's logic, in the same way as

programs with negation as failure (provided with 3-valued semantics) extend positive (2-valued) programs.
Furthermore we show that the answer sets of [14] correspond to a notion of standard 4-valued Belnap's models, and we suggest with examples that our computable semantics captures essential aspects of the answer set semantics for extended programs.

## 2 PRELIMINARIES ON EXTENDED LOGIC PROGRAMS

**Closed World Assumption.**

When the closed world assumption is not applicable, i.e. when dealing with possibly incomplete or inconsistent information, the deduction process of the falsity of a sentence $A$ should be independent from that of the truth of $A$. Such a deduction process should then be able to infer negative information in another way than does the usual negation as failure, namely it should be able to infer "explicitly" negated information as well as positive one. In order to do this, one has to distinguish between two kinds of negation: *explicit negation* (denoted $\neg$) and *negation as failure* (denoted $/$). Not unexpectedly, $\neg$ will be allowed to occur in the head of clauses, but not $/$.

Here is a short example borrowed from [14].
A College uses the following rules for awarding scholarships to its students:

(i) Every student with a GPA of at least 3.8 is eligible.
(ii) Every minority student with a GPA of at least 3.6 is eligible.
(iii) No student with the GPA under 3.6 is eligible.
(iv) The students whose eligibility is not determined by these rules are interviewed by the scholarship committee.

Assume this program is used in conjunction with a database containing the following fact about one of the students: Ann's GPA is 3.7.
The database contains no information about $minority(ann)$, whereas Ann is a minority student, but declined to state this fact on her application, as a matter of principle. Representing such a knowledge in a logic programming language requires thus two kinds of negation: an explicit negation, which may occur in the head of clauses (rule 3) and negation by failure (rule 4).

Such programs have been introduced by Gelfond and Lifschitz in [13] and called extended programs.

Note that the two kinds of negation allow to distinguish between *temporary* and *definitive* lack of information. For this reason, extended programs have been early devoted to temporal reasoning about actions. In [15] Gelfond and Lifschitz use them as a language for deriving properties of actions in an open context.

**Syntax of Extended Programs.**

We assume our language $\mathcal{L}$ to be fixed, and contain, for each $n \geq 0$, a countable set of $n$-ary function symbols and a countable set of $n$-ary predicate symbols; in addition, $\mathcal{L}$ has a symbol $=$ for equality, that never occurs in a program, but is used in forming the completed program. The set $\mathcal{V}$ of variables is fixed as well.

*Atomic formulas* are defined as usual from $\mathcal{L}$ and $\mathcal{V}$. A *classical literal* is an atomic formula or the *explicit negation* $\neg A$ of an atomic formula $A$. A (general) *literal* is a classical literal or the *negation by failure*, $/L$, of a classical literal $L$. A literal of the form $/L$ is called a *slashed literal.* A *clause* is of the form:

$$L_0 :\!- L_1, \ldots, L_m, /L_{m+1}, \ldots, /L_n,$$

where the $L_i$'s are classical literals, $0 \leq m \leq n$, and commas stand as usual for conjunctions $\wedge$. If $n = 0$, we just write $L_0$. $L_0$ is called the *head* of the clause, and $(L_1, \ldots, L_m, /L_{m+1}, \ldots, /L_n)$ its *body.* Note that a clause admits explicitly negated literals in its head. An *(extended) program* is a finite set of clauses. A *query* is of the form $L_1, \ldots, L_m, /L_{m+1}, \ldots, /L_n$.

**Call-consistent, stratified, p.o.c. and strict extended programs.**

The following definitions are not essential for our semantics, they will be used in Section 5 to establish connections with the answer sets' semantics of Gelfond and Lifschitz.

Let PRED be the set of all predicate symbols or $\neg$-negated predicate symbols. Let P be a given program. If $p, q \in$ PRED, we define (as in [18]) $p \sqsupseteq_{+1} q$ iff P contains a clause in which $p$ occurs in the head and $q$ occurs in a classical (not-slashed) literal of the body. We say $p \sqsupseteq_{-1} q$ iff P contains a clause in which $p$ occurs in the head and $q$ occurs in a slashed literal of the body. Let $\geq_{+1}$ and $\geq_{-1}$ be the least pair of relations on PRED satisfying: $p \geq_{+1} p$ and $p \sqsupseteq_i q \;\&\; q \geq_j r \Rightarrow p \geq_{i \cdot j} r$, $i, j \in \{+1, -1\}$. Intuitively $\geq_{+1}$ (resp. $\geq_{-1}$) denotes the relation of dependance among predicates through an even number (including 0) of / (resp. an odd number of /).

These relations could have been defined similarly on ground atoms instead of predicates. In the following definitions, we use both forms (on predicates, on atoms), so for sake of clarity we explicitly mention which one is intended.

We say that P is *call-consistent* iff we never have $p \geq_{-1} p$ for any predicate symbol $p$, i.e. no predicate symbol or $\neg$-negated predicate symbol $p$ is defined negatively from itself.

Following [3] we say that an extended program $P$ is *stratified* iff no predicate symbol or $\neg$-negated predicate symbol depends on itself through at least one / negation.

It is *locally stratified* iff no ground atom or ground $\neg$-negated atom depends on itself through at least one / negation.

Following [6] we say that an extended program $P$ is *positive order consistent (p.o.c.)* iff the relation $\sqsupseteq_{+1}$ on atoms has no infinite decreasing chain (it is in particular the case if all recursions are through one or more /).

If $\phi$ is a query, we say $\phi \geq_i p$ iff either $a \geq_i p$ for some atom $a$ occurring positively in $\phi$ or $a \geq_{-i} p$ for some $a$ occurring negatively in $\phi$. An extended program $P$ is said *strict w.r.t. the query* $\phi$ iff for no predicate letter $p$ do we have both $\phi \geq_{+1} p$ and $\phi \geq_{-1} p$.

## 3 BELNAP'S LOGIC

In [4] Belnap introduced a four-valued logic intended to deal in a useful way with inconsistent or incomplete information (see also [2]).

A way to interpret Belnap's truth values is to think of them as sets of classical truth values: we write t for $\{true\}$, f for $\{false\}$, $\perp$ for $\emptyset$ (indicating a lack of information) and $\top$ for $\{true, false\}$ (indicating inconsistency).

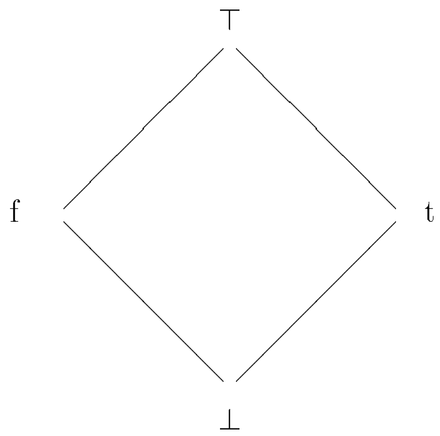


Fig. 1. The bilattice of Belnap's logic.

This set $\mathcal{T}$ of truth values has two natural orderings: one is the subset relation, a *knowledge* ordering $\lhd_k$ (the *vertical* ordering in Figure 1), and a *truth* ordering $\lhd_t$ (the *horizontal* one). In this way, inconsistency ($\top$) and lack of information ($\bot$) cannot be distinguished according to the truth ordering. Each of these orderings provides the set of truth values with the structure of a lattice, so that the whole structure can be considered as the simplest non-trivial *bilattice* [16,9].

Meet and join under $\lhd_t$ are denoted $\wedge$ and $\vee$; they are generalizations of the usual *conjunction* and *disjunction*. Meet and join under $\lhd_k$ are denoted $\otimes$ and $\oplus$, respectively *consensus* and *gullability* operators; but we shall not need them in our extended logic programs. On the other hand, there is a natural notion of *negation* $\neg$, which flips the diagram from left to right, switching f and t, leaving $\bot$ and $\top$ alone.

In [10] Fitting proposes an extension of logic programming to bilattices: to execute a bilattice logic program, you just compute the actual (truth) value $v$ of the body of a clause and replace the value of the head by $v$. Since all connectives considered by Fitting are monotone w.r.t. $\lhd_k$, this mechanism amounts to adding information to the fact base: your knowledge about the situation increases (but not necessarily following the truth ordering).

In this paper, we shall consider also connectives that are non-monotonic w.r.t. the knowledge ordering in order to model negation as failure. We shall consider the *slash* /, which flips the diagram from bottom-left to top-right, switching f and $\top$, and t and $\bot$ [2]. In a four-valued logic it is very natural to introduce such a connective, moreover there is a need for it, to get a fully expressive set of connectives:

**Complete Sets of Connectives.**

In classical logic, the connectives $\neg$, $\wedge$ and $\vee$ form a complete set, i.e for every integer $n$, all the mappings from $\{f,t\}^n$ to $\{f,t\}$ can be expressed by composition of some of the connectives. In fact, $\neg$ and $\wedge$ suffice to form a complete set for classical logic.

When moving to Belnap's logic, the connectives $\neg$ and $\wedge$ extending the corresponding classical ones do not form a complete set. Precisely: if $K_n$ are sets of mappings from $\{f,t,\bot,\top\}^n$ to $\{f,t,\bot,\top\}$, and $K = \bigcup K_n$, let us call $\overline{K}$ the intersection of all the sets $S$ such that:

(i) $S$ contains $K$, the 0-ary f, t, $\bot$, $\top$, and the $k^{th}$ $n$-ary projection $\pi_n^k$ for every integers $n \geq k \geq 1$,

(ii) if $f_1 \ldots f_p$ are in $S$ and all $n$-ary, and if $\kappa$ is in $K$ and $p$-ary, then $\kappa \circ <$

[2] We will give some intuition behind the choice of / as the right connective for modeling negation as failure in Section 4.

$f_1 ... f_p$> is in $S$ and is $n$-ary.

We say that the set $K$ is *complete* for $\mathcal{T}$ if $\overline{K}$ contains all the mappings from $\{f, t, \perp, \top\}^n$ to $\{f, t, \perp, \top\}$, for every integer $n$.

**Proposition 1** *The connectives $\neg$ and $\wedge$ do not form a complete set for $\mathcal{T}$.*

**Proof:** let $K$ be this set of connectives. One verifies easily that $\neg$ and $\wedge$ are *monotone* w.r.t. $\lhd_k$, i.e. $x \lhd_k x'$ implies $\neg(x) \lhd_k \neg(x')$, and $x \lhd_k x'$ and $y \lhd_k y'$ imply $x \wedge y \lhd_k x' \wedge y'$. (The property that the lub and glb under one ordering of a bilattice are monotone in the other ordering holds in any distributive bilattice [16,11]; this is a corollary of Ginsberg's representation theorem for distributive bilattices [16].)
By induction, all connectives in $\overline{K}$ are monotone w.r.t. $\lhd_k$ as well.
But / is not monotone w.r.t. the knowledge ordering: for instance, $\perp \lhd_k$ t, but $/\perp =$ t $\not\lhd_k$ /t $= \perp$. Hence $/ \notin \overline{K}$. □

Hence, to get a complete set of connectives, we need to add at least a non-monotonic connective to $\{\neg,\wedge\}$. The addition of / suffices:

**Proposition 2** *The connectives $\neg$, $\wedge$ and / form a complete set for $\mathcal{T}$.*

**Proof :** let $K'$ be this set of connectives. We prove, by induction on $n$, that every function from $\{f, t, \perp, \top\}^n$ to $\{f, t, \perp, \top\}$ is in $\overline{K'}$.

• <u>$n = 0$ :</u> because of condition (i), all truth values are in $\overline{K'}$.

• <u>$n + 1$ :</u> define $\backslash x = \neg/\neg x$ (exchanges f and $\perp$, t and $\top$), and $-x = /\neg/x$. Hence the connectives $\backslash$ and $-$ belong to $\overline{K'}$.

Now let $f$ be a $(n + 1)$-ary function. We define the $n$-ary functions $g$, $h$, $k$, $l$ by:

$$\begin{cases} f(x_1...x_n, f) = g(x_1...x_n) \\ f(x_1...x_n, t) = h(x_1...x_n) \\ f(x_1...x_n, \perp) = k(x_1...x_n) \\ f(x_1...x_n, \top) = l(x_1...x_n) \end{cases}$$

If (?t) is the unary function that lets t invariant and maps the other truth values to f, then we have: $f(x_1...x_{n+1}) = [(?t)(\neg x_{n+1}) \wedge g(x_1...x_n)] \vee [(?t)x_{n+1} \wedge h(x_1...x_n)] \vee [(?t)(/x_{n+1}) \wedge k(x_1...x_n)] \vee [(?t)(\backslash x_{n+1}) \wedge l(x_1...x_n)]$.

But (?t)$x = x \wedge -x$. Hence $f$ can be expressed in $K'$, and the result is proved for any integer $n$. □

## 4 9-VALUED KUNEN-STYLE SEMANTICS VIA BELNAP'S LOGIC

In the usual case (programs without explicit negation $\neg$), the semantics is 3-valued, and this corresponds to the three possible situations for a ground query: 'yes' answer (true), finite failure (false) and looping (undefined).

In the case of programs with both negations, the answers concerning the truth and falsity of a query are completely independant. So the truth value assigned to a formula $A$ will be a couple of classical truth values (true, false, undefined), the $1^{st}$ element of this couple corresponding to the knowledge about the truth of $A$, and the $2^{nd}$ one corresponding to the knowledge about its falsity. Hence logic programs with both negations will be provided with a 9-valued Kunen-style semantics.

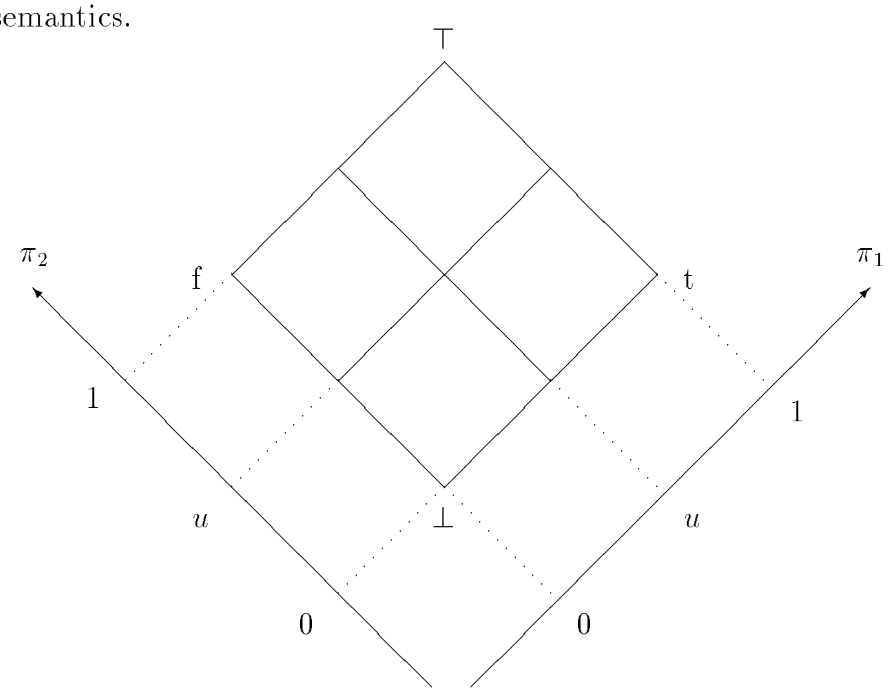


Fig. 2. The 9-valued logic of extended programs.

Truth values are then handled as points in a square (see Figure 2). For instance, $t = (1,0)$, $\perp = (0,0)\ldots$ We can define two projections on truth values: $\pi_1(x,y) = x$ and $\pi_2(x,y) = y$, where $x, y \in \{0, u, 1\}$. Now define an order $<_t$ on $\{0,u,1\}$ by $0 <_t u <_t 1$, and the order-reversing function $v \mapsto \overline{v}$ by $\overline{0} = 1$, $\overline{1} = 0$ and $\overline{u} = u$: then we may extend the connectives defined in Section 3 by $(v_1, v_2) \wedge (w_1, w_2) = (\min_{<_t}(v_1, w_1), \max_{<_t}(v_2, w_2))$, $(v_1, v_2) \vee (w_1, w_2) = (\max_{<_t}(v_1, w_1), \min_{<_t}(v_2, w_2))$, $\neg(v_1, v_2) = (v_2, v_1)$ and $/(v_1, v_2) = (\overline{v_1}, v_2)$.

The choice of the connective / for modeling negation as failure becomes clear now. Since the positive and negative informations about a formula $A$ are separated in extended programs, such a connective has to negate (the truth value representing) the positive information in $A$, keeping unchanged the negative information in $A$. This is indeed what the connective / does.

The definition of the completed program (see below) uses a (2-valued) Lukasiewicz equivalence $\leftrightarrow$: $v \leftrightarrow w$ is t iff $\pi_1(v) = \pi_1(w)$ and f otherwise. Note that $\leftrightarrow$ is not monotone w.r.t. the knowledge ordering of Belnap's logic, just like /. Of course $v \leftrightarrow w$ can be defined here in terms of $\wedge$, / and $\neg$, thanks to Proposition 2.

### *4.1 COMPLETED PROGRAM AND 9-VALUED MODELS*

**Completed program.**

Let $L(\tau_1, ..., \tau_n) :- \phi$ be a clause, with variables $Y_1 \ldots Y_j$. Its *normalization* is

$$L(X_1, ..., X_n) :- \exists Y_1 \cdots \exists Y_j (X_1 = \tau_1 \wedge ... \wedge X_n = \tau_n \wedge \phi)$$

where $X_1 \ldots X_n$ are new variables.

Let $P$ be an extended program and $L(X_1, ..., X_n) :- \psi_i$ $(1 \leq i \leq m)$ be the $m$ normalizations of the clauses in $P$ where $L$ occurs in the head. Then the *completed definition* of the $n$-ary classical literal $L$ is $\forall X_1 \cdots \forall X_n\ (L(X_1, ..., X_n) \leftrightarrow \psi_1 \vee \cdots \vee \psi_m)$. If $m = 0$, we just write $/L(X_1, ..., X_n)$.

Now the *completed program* $P^\star$ is the set of the completed definitions of all classical literals, together with the axioms of Clark's equational theory CET (see [5]).

**9-Valued structures.**

A 9-*valued structure* $\mathcal{A}$ for the fixed language $\mathcal{L}$ consists of a nonempty set $A$ (the *domain* of interpretation), and:

(i) for every $n$-ary function symbol $f$, $\mathcal{A}(f) : A^n \rightarrow A$ is a $n$-ary function,

(ii) for every $n$-ary predicate symbol $p$ other than $=$, $\mathcal{A}(p)$ is a mapping from $A^n$ to the set of 9 truth values; $\mathcal{A}(=)$ is always true identity, i.e., $\mathcal{A}(=)(a, b)$ is $t$ iff $a$ and $b$ are the same object and $f$ otherwise.

A 4-*valued structure* is simply a 9-valued structure in which, for every predicate $p$, neither $\pi_1(\mathcal{A}(p))$ nor $\pi_2(\mathcal{A}(p))$ takes the value $u$. As usual, the interpretation is extended to formulas according to the 9-valued truth tables (defined above componentwise); for the quantifiers, we define obviously $\mathcal{A}(\exists X\phi) = \bigvee_{a\in A}\mathcal{A}(\phi(a))$ and $\mathcal{A}(\forall X\phi) = \bigwedge_{a\in A}\mathcal{A}(\phi(a))$.

We say that the 9-valued structure $\mathcal{A}$ is a *model* of the completed program $P^\star$, denoted $\mathcal{A} \models_9 P^\star$, iff all formulas in $P^\star$ have truth value t in $\mathcal{A}$. If $\mathcal{A}$ is in fact a 4-valued structure, then we write $\mathcal{A} \models_4 P^\star$.

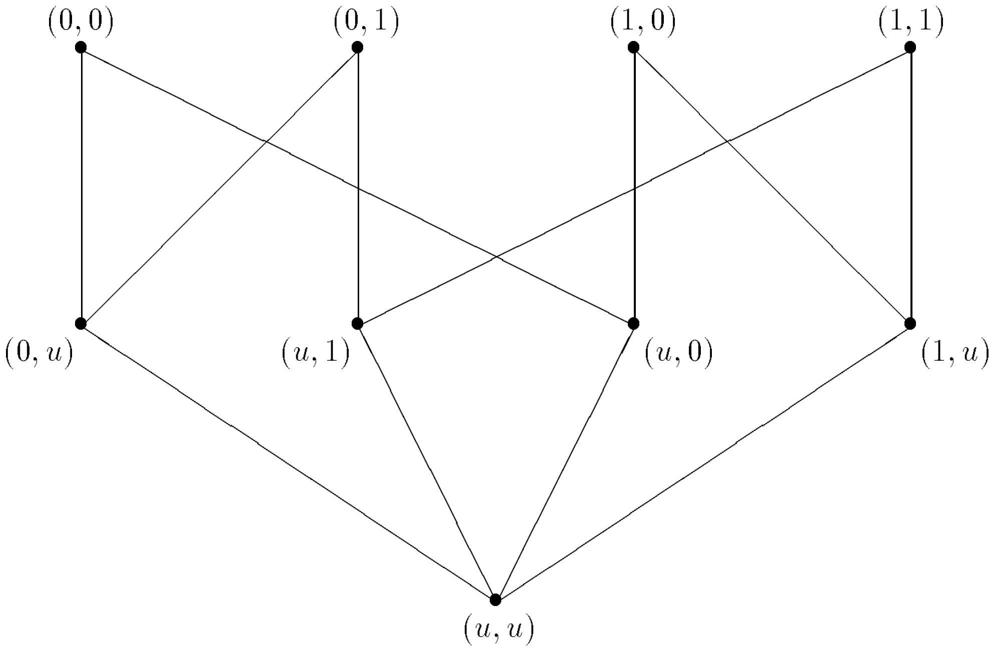


Fig. 3. The 9-valued bilattice based on Belnap's logic.

**Extensions.**

Let $<_k$ be the ordering on $\{0, u, 1\}$ such that $u <_k 0$ and $u <_k 1$. If $\mathcal{A}$ and $\mathcal{B}$ are two 9-valued structures, we shall say that $\mathcal{B}$ is an *extension* of $\mathcal{A}$ iff $\mathcal{A}$ and $\mathcal{B}$ have the same domain of interpretation and agree on the interpretations of all function symbols, and for each ground atomic formula $\phi$, $\pi_1(\mathcal{A}(\phi)) \leq_k \pi_1(\mathcal{B}(\phi))$ and $\pi_2(\mathcal{A}(\phi)) \leq_k \pi_2(\mathcal{B}(\phi))$. The natural ordering between extensions is induced by the ordering $\preceq_k$ defined component-wise on the 9 truth values from $\leq_k$ (see Figure 3).

Intuitively, an extension of $\mathcal{A}$ is "less undefined" than $\mathcal{A}$. It is a concept different from that of "expansion" (see [18]) and more natural in our context, but Kunen's proofs of interest for us can be easily adapted to the notion of extension.

To see this, let us return temporarily to the classical setting and recall the definition of an expansion: if $\mathcal{P}$ and $\mathcal{Q}$ are sets of predicate symbols, $\mathcal{P} \subseteq \mathcal{Q}$, $\mathcal{M}$ is a 3-valued $\mathcal{P}$-structure (i.e. a structure that interprets only predicate symbols in $\mathcal{P}$) and $\mathcal{N}$ a 3-valued $\mathcal{Q}$-structure, then $\mathcal{N}$ is called an *expansion* of $\mathcal{M}$ if $\mathcal{M}$ and $\mathcal{N}$ have the same domain and agree on the interpretations of all function symbols and predicate symbols in $\mathcal{P}$. Let us define an *extension* of a classical 3-valued structure $\mathcal{M}$ to be a 3-valued structure $\mathcal{N}$ such that for every formula $\phi$, $\mathcal{M}(\phi) \leq_k \mathcal{N}(\phi)$. If $\mathcal{P}$ is a set of predicate symbols and $\mathcal{M}$ is a 3-valued $\mathcal{P}$-structure, let $\mathcal{M}_\mathcal{P}$ denote the structure such that $\mathcal{M}_\mathcal{P}(p) = \mathcal{M}(p)$ if $p \in \mathcal{P}$ else $\mathcal{M}_\mathcal{P}(p) = u$. Then the following (trivial) proposition establishes the connection between expansions and extensions.

**Proposition 3** *Let $\mathcal{M}$ be a 3-valued $\mathcal{P}$-structure and $\mathcal{N}$ a 3-valued $\mathcal{Q}$-structure, with $\mathcal{P} \subseteq \mathcal{Q}$. $\mathcal{N}_\mathcal{Q}$ is an extension of $\mathcal{M}_\mathcal{P}$ iff $\mathcal{N}$ is an expansion of $\mathcal{M}$.*

This shows that expansions and extensions are about the same notion (for instance: Kunen's immediate consequence operator $\Psi$ maps each 3-valued structure to an "extension" of it; besides if $\mathcal{M}$ is a 3-valued $\mathcal{Q}$-structure and $S$ a signing for $\mathcal{P} \subseteq \mathcal{Q}$, then $2val(\mathcal{M}, S)_\mathcal{Q}$ is an "extension" of $\mathcal{M}_\mathcal{Q}$; etc.).

**Immediate consequence operator.**

Given an extended program $P$, we define an operator $T_P$ which maps each 9-valued structure to an extension of it. Let $\mathcal{A}$ be a 9-valued structure, $p$ a $n$-ary predicate and $a_1 \ldots a_n \in A$. The domain of $T_P(\mathcal{A})$ equals that of $\mathcal{A}$; $T_P(\mathcal{A})$ and $\mathcal{A}$ agree on the interpretations of all function symbols. For predicate symbols, let $v = T_P(\mathcal{A})(p)(a_1...a_n)$ be defined by:

(i) – $\pi_1(v) = 1$ iff there is a clause in $P$ of the form $p(\tau_1...\tau_n) :\!-\ \phi$, with variables $Y_1 \ldots Y_j$, and some $b_1 \ldots b_j \in A$, such that $\pi_1(\mathcal{A}(\phi)(b_1...b_j)) = 1$ and $\forall i$, $\mathcal{A}(\tau_i)(b_1...b_j) = a_i$;
– $\pi_1(v) = 0$ iff for each clause in $P$ of the form $p(\tau_1...\tau_n) :\!-\ \phi$, with variables $Y_1 \ldots Y_j$ and every $b_1 \ldots b_j \in A$, we have either $\pi_1(\mathcal{A}(\phi)(b_1...b_j)) = 0$ or some $\mathcal{A}(\tau_i)(b_1...b_j) \neq a_i$;
– $\pi_1(v) = u$ otherwise.

(ii) – $\pi_2(v) = 1$ iff there is a clause in $P$ of the form $\neg p(\tau_1...\tau_n) :\!-\ \phi$, with variables $Y_1 \ldots Y_j$, and some $b_1 \ldots b_j \in A$, such that $\pi_1(\mathcal{A}(\phi)(b_1...b_j)) = 1$ and $\forall i$, $\mathcal{A}(\tau_i)(b_1...b_j) = a_i$;
– $\pi_2(v) = 0$ iff for each clause in $P$ of the form $\neg p(\tau_1...\tau_n) :\!-\ \phi$, with variables $Y_1 \ldots Y_j$ and every $b_1 \ldots b_j \in A$, we have either $\pi_1(\mathcal{A}(\phi)(b_1...b_j)) = 0$ or some $\mathcal{A}(\tau_i)(b_1...b_j) \neq a_i$;
– $\pi_2(v) = u$ otherwise.

One verifies easily that $T_P(\mathcal{A})$ is indeed an extension of $\mathcal{A}$.

**Theorem 4** *Let $\mathcal{A}$ be a 9-valued structure. $T_P(\mathcal{A}) = \mathcal{A}$ iff $\mathcal{A} \models_9 P^\star$: the fixed points of $T_P$ are exactly the 9-valued models of $P^\star$.*

**Conversion to 4-valued structures.**

Since $T_P$ is monotone (w.r.t. the well-founded ordering $\preceq_k$ induced on 9-valued structures), it has a fixed point (see [8]), hence $P^\star$ always has a 9-valued model. More specifically, as in the classical case, we would like to know when $P^\star$ has in fact a 4-valued model. This is given by the condition of call-consistency introduced in Section 2:

**Theorem 5** *If $P$ is call-consistent and $\mathcal{A} \models_9 P^\star$, then $\mathcal{A}$ has a 4-valued extension $\mathcal{B}$ such that $\mathcal{B} \models_4 P^\star$. As a consequence, if $P$ is call-consistent, then $P^\star$ has a 4-valued model.*

**Theorem 6** *Suppose $P$ is call-consistent and strict w.r.t. a query $\phi$. Then $\phi$ is a 4-valued consequence of $P^\star$ iff it is a 9-valued consequence of $P^\star$.*

This means that our extended programs are to Fitting's programs on Belnap's logic, what programs with negation as failure are to positive programs.

In the next Section, we give proofs of these theorems through a "faithful" translation from extended programs to normal programs (such a translation has already been used by Gelfond and Lifschitz in [14].

### *4.2 REDUCTION TO NORMAL PROGRAMS*

Let $\mathcal{L}$ be a fixed first-order language. We build a new first-order language $\mathcal{L}^\neg$ by adding to $\mathcal{L}$, for each predicate symbol $p$, a new predicate symbol $p'$.

Let $L$ be a classical literal built on the language $\mathcal{L}$: if $L$ is an atomic formula, then let $L^\neg$ be $L$; if $L = \neg p(a_1 ... a_n)$, then $L^\neg = p'(a_1 ... a_n)$. (Note that in any case, $L^\neg$ is an atomic formula built on $\mathcal{L}^\neg$.)

Let $P$ be an extended program. $P^\neg$ is the classical program obtained by replacing each clause $L_0 :- L_1, \dots, L_m, /L_{m+1}, \dots, /L_n$ by $L_0^\neg :- L_1^\neg, \dots, L_m^\neg, /L_{m+1}^\neg, \dots, /L_n^\neg$ . In $P^\neg$, / stands for negation as failure. Again, the notation / is unusual, but using $\neg$ could have lent to confusion. Similarly we can define $\Sigma^\neg$ for any set $\Sigma$ of $\mathcal{L}$-formulas such that negation $\neg$ occurs only in front of atomic formulas.

Let $\mathcal{A}$ be a 9-valued ($\mathcal{L}$-)structure. The 3-valued ($\mathcal{L}^\neg$-)structure $\mathcal{A}^\neg$ is defined as follows:

(i) the domain of $\mathcal{A}^\neg$ is $A$, the domain of $\mathcal{A}$;

(ii) for every $n$-ary function symbol $f$, $\mathcal{A}^\neg(f) = \mathcal{A}(f)$;

(iii) for $n$-ary predicate symbols other than $=$, we have to distinguish between two cases: $\mathcal{A}^\neg(p) = \pi_1(\mathcal{A}(p))$ and $\mathcal{A}^\neg(p') = \pi_2(\mathcal{A}(p))$; $\mathcal{A}^\neg(=)$ is always true

(2-valued) identity, i.e. $\mathcal{A}^{\neg}(=)(a,b)$ is 1 iff $a$ and $b$ are the same object and 0 otherwise.

Finally, if $\Pi$ is a normal program, $\Psi_{\Pi}$ denotes the immediate consequence operator of Kunen [18] on 3-valued structures, and $\Pi^{\star}$ denotes Clark's completed program [5].

**Proposition 7** *Let $\mathcal{A}$ and $\mathcal{B}$ be 9-valued ($\mathcal{L}$-)structures and $P$ an extended program. The following statements hold:*

*(i)* $\mathcal{A}$ *is 4-valued iff* $\mathcal{A}^{\neg}$ *is 2-valued;*

*(ii)* $\mathcal{B}$ *is an extension of* $\mathcal{A}$ *iff* $\mathcal{B}^{\neg}$ *is an extension of* $\mathcal{A}^{\neg}$*;*

*(iii)* $P^{\star\neg} = P^{\neg\star}$*;*

*(iv)* $T_P(\mathcal{A}) = \mathcal{A}$ *iff* $\Psi_{P^{\neg}}(\mathcal{A}^{\neg}) = \mathcal{A}^{\neg}$*;*

*(v)* $\mathcal{A} \models_9 P^{\star}$ *iff* $\mathcal{A}^{\neg} \models_3 P^{\star\neg}$.

**Proof:**
(i) and (iii) are clear;

(ii): the main observation is that $\wedge$, $\neg$ and $/$ are monotone w.r.t. the ordering $\preceq_k$, and that $\wedge$ and $\neg$ are monotone w.r.t. $\leq_k$. Now: $\mathcal{B}$ is an extension of $\mathcal{A} \iff$ for every ground formula $\phi$, $\pi_1(\mathcal{A}(\phi)) \leq_k \pi_1(\mathcal{B}(\phi))$ and $\pi_2(\mathcal{A}(\phi)) \leq_k \pi_2(\mathcal{B}(\phi)) \iff$ for every $n$-ary predicate symbol $p$ and each $a_1 \ldots a_n \in A = B$, $\pi_1(\mathcal{A}(p(a_1...a_n))) \leq_k \pi_1(\mathcal{B}(p(a_1...a_n)))$ and $\pi_2(\mathcal{A}(p(a_1...a_n))) \leq_k \pi_2(\mathcal{B}(p(a_1...a_n)))$ (for $\wedge$, $\neg$ and $/$ are monotone w.r.t. $\preceq_k$) $\iff$ for every $n$-ary predicate symbol $p$ and each $a_1 \ldots a_n \in A = B$, $\mathcal{A}^{\neg}(p(a_1...a_n)) \leq_k \mathcal{B}^{\neg}(p(a_1...a_n))$ and $\mathcal{A}^{\neg}(p'(a_1...a_n)) \leq_k \mathcal{B}^{\neg}(p'(a_1...a_n)) \iff$ for every ground formula $\phi$, $\mathcal{A}^{\neg}(\phi) \leq_k \mathcal{B}^{\neg}(\phi)$ (for $\wedge$ and $\neg$ are monotone w.r.t. $\leq_k$) $\iff \mathcal{B}^{\neg}$ is an extension of $\mathcal{A}^{\neg}$;

(iv) follows from the definition of $T_P$ and the remark that $\pi_2(T_P(\mathcal{A})(p)(a_1...a_n)) = \pi_1(T_P(\mathcal{A})(\neg p)(a_1...a_n))$;

(v): one can prove easily by induction that for every ground formula $\phi$ such that $\neg$ occurs only in front of atomic formulas, we have $\mathcal{A}^{\neg}(\phi^{\neg}) = \pi_1(\mathcal{A}(\phi))$. Now, for any completed definition $\forall X_1 \cdots \forall X_n(p(X_1,...,X_n) \leftrightarrow \psi$ in $P^{\star}$, we have: $\mathcal{A} \models_9 p \leftrightarrow \psi \iff \pi_1(\mathcal{A}(p)) = \pi_1(\mathcal{A}(\psi)) \iff \mathcal{A}^{\neg}(p) = \mathcal{A}^{\neg}(\psi^{\neg})$ (thanks to the above remark) $\iff \mathcal{A}^{\neg} \models_3 p \leftrightarrow \psi^{\neg}$; and for any completed definition $\forall X_1 \cdots \forall X_n(\neg p(X_1,...,X_n) \leftrightarrow \psi$ in $P^{\star}$, we have: $\mathcal{A} \models_9 \neg p \leftrightarrow \psi \iff \pi_1(\mathcal{A}(\neg p)) = \pi_1(\mathcal{A}(\psi)) \iff \mathcal{A}^{\neg}(p') = \mathcal{A}^{\neg}(\psi^{\neg})$ (thanks to the above remark) $\iff \mathcal{A}^{\neg} \models_3 p' \leftrightarrow \psi^{\neg}$. □

**Proof of Theorem 4:** it follows from Proposition 7 (iii, iv, v) and [18] (Lemma 3.1). □

**Proof of Theorem 5:** it is an easy consequence of Proposition 7 (i, ii, v) and [18] (Theorem 3.4 and Corollary 3.5: namely if $\Pi$ is a call-consistent normal program and $\mathcal{M}$ is a 3-valued structure such that $\mathcal{M} \models_3 \Pi^\star$, then $\mathcal{M}$ has a 2-valued extension $\mathcal{N}$ such that $\mathcal{N} \models_2 \Pi^\star$). $\square$

**Proof of Theorem 6.** It follows directly from Proposition 7 (i, iii, v) and [18] (Theorem 3.6): $P^\star \vdash_9 \phi$ iff $P^{\star\neg} \vdash_3 \phi^\neg$ by Proposition 7 (v), iff $P^{\neg\star} \vdash_3 \phi^\neg$ by Proposition 7 (iii). Besides $P^\neg$ is call-consistent and strict w.r.t. $\phi^\neg$ in the sense of Kunen, so by Theorem 3.6 [18], $P^{\neg\star} \vdash_3 \phi^\neg$ iff $P^{\neg\star} \vdash_2 \phi^\neg$ iff $P^{\star\neg} \vdash_2 \phi^\neg$, i.e. for all 2-valued $\mathcal{B}$, $\mathcal{B} \models_2 P^{\star\neg}$ implies $\mathcal{B} \models_2 \phi^\neg$, i.e. (by Proposition 7 (i)) for all 4-valued $\mathcal{A}$, $\mathcal{A}^\neg \models_2 P^{\star\neg}$ implies $\mathcal{A}^\neg \models_2 \phi^\neg$. By Proposition 7 (v) this is equivalent to $P^\star \vdash_4 \phi$. $\square$

**Operational Semantics.**

The reduction to normal programs allows to consider that literals $A$ and $\neg A$ have a "separate life". Hence SLDNF resolution [18] (resp. constructive negation [19,7]) provide extended programs with correct (resp. complete) operational semantics, in the following way: the answer to a given goal $G$ in an extended logic program is obtained by combining the answers to $G$ and $G^\neg$ (in the corresponding normal program); each answer sets the value of one component $v_1$ or $v_2$ of the truth value $(v_1, v_2)$ of $G$: 'yes' is 1, 'no' is 0, no answer means $u$.

**Example.**

The rules in the example of Section 2 can be encoded in the following extended program $P$:

$eligible(X) :\!-\; highGPA(X).$
$eligible(X) :\!-\; minority(X), fairGPA(X).$
$\neg\; eligible(X) :\!-\; \neg\; fairGPA(X).$
$interview(X) :\!-\; /\; eligible(X), /\; \neg\; eligible(X).$

with the following facts:

$fairGPA(ann).$
$\neg\; highGPA(ann).$

Our Kunen-style semantics (weaker than the answer sets' semantics) suffices to deduce the expected assertion $interview(ann)$, i.e. $interview(ann)$ is a 9-valued consequence of the completed program. Note that by Theorem 6, the 4-valued consequences and 9-valued consequences of $P^\star$ are identical, since $P$ is call-consistent and strict w.r.t. $interview(ann)$.

## 5 CONNECTION WITH THE ANSWER SETS' SEMANTICS

In this Section we define answer sets for our extended programs, which are obtained from those of Gelfond and Lifschitz's by dropping their rule which globalizes contradictions (saying that a program which implies both $A$ and $\neg A$ implies anything). We prove that our answer sets for a given program $P$ are 4-valued models of $P^\star$, if explicit negation and negation as failure are interpreted by the connectives $\neg$ and $/$, respectively, i.e. that the logic underlying logic programming with classical negation and negation as failure is indeed Belnap's logic.

Let $P$ be a program with no negation by failure: define $\beta(P)$ as the least set $S$ of ground classical literals such that for every ground rule instance $L :- L_1, \ldots, L_m$ in $P$, $L_1, \ldots, L_m \in S \Longrightarrow L \in S$.

Let $P$ be an extended program (with negation by failure) and $S$ a set of ground classical literals: define $P^S$ as the program (with no negation by failure) obtained from $P$ by:

- removing every ground rule instance $L :- L_1, \ldots, L_m, /L_{m+1}, \ldots, /L_n$ such that for some $i$, $m+1 \leq i \leq n$, $L_i \in S$;
- removing all slashed literals from all other ground rules instances.

Now define an *answer set* of an extended program $P$ to be a solution $S$ to the equation $S = \beta(P^S)$.

Finally we define a translation $m$ between sets of ground classical literals and 4-valued structures: if $S$ is a set of ground classical literals, $m(S)$ is the structure whose domain is the Herbrand universe, that interprets terms by themselves and such that, if $A$ is any ground atomic formula:

- if $A \in S$, then $\pi_1(m(S)(A)) = 1$,
- if $\neg A \in S$, then $\pi_2(m(S)(A)) = 1$,
- if $A \notin S$, then $\pi_1(m(S)(A)) = 0$,
- if $\neg A \notin S$, then $\pi_2(m(S)(A)) = 0$.

**Theorem 8** *If $S$ is an answer set of an extended program $P$, then $m(S)$ is a 4-valued Herbrand model of $P^\star$.*

**Proof:** let $S$ be an answer set of $P$, i.e. $S = \beta(P^S)$, $a_1 \ldots a_n$ be Herbrand terms, and $\forall X_1 \cdots \forall X_n\ (L(X_1, ..., X_n) \leftrightarrow \psi$ be any completed definition in $P^\star$. We have to prove that $\pi_1(m(S)(\psi)) = \pi_1(m(S)(L)(a_1, ..., a_n))$.

- If $\pi_1(m(S)(L)(a_1, ..., a_n)) = 1$ then $L(a_1, ..., a_n) \in S$; because of the definition of $\beta$, there must be a ground rule instance $R^S : L(a_1, ..., a_n) :$

$- L_1, \ldots, L_k$ in $P^S$ such that $L_1, \ldots, L_k \in S$. This rule comes from a rule $R = (L :- L_1, \ldots, L_k, /L_{k+1}, \ldots /L_n)$ in $P$, and therefore $L_{k+1}, \ldots, L_n \notin S$. Thus $\pi_1(m(S)(L_1)) = \cdots = \pi_1(m(S)(L_k)) = \pi_1(m(S)(/L_{k+1})) = \cdots = \pi_1(m(S)(/L_n)) = 1$, and $\psi = \phi \vee \exists(L_1 \wedge \cdots \wedge L_k \wedge /L_{k+1} \wedge \cdots \wedge /L_n)$. Hence $\pi_1(m(S)(\psi)) = 1$.

- If $\pi_1(m(S)(L)(a_1, ..., a_n)) = 0$ then $L(a_1, ..., a_n) \notin S$; for all ground rule instance $R^S$ in $P^S$ of the form $(L(a_1, ..., a_n) :- L_1, \ldots, L_k)$, one of the $L_1, \ldots, L_k$ does not belong to $S$, say $L_i$, so that $\pi_1(m(S)(L_i)) = 0$ and hence $\pi_1(m(S)(\psi)) = 0$.
- $\pi_1(m(S)(L)(a_1, ..., a_n)) = u$ never happens. $\square$

Thus our answer sets can be identified with models in Belnap's logic. Besides the well-known results about answer sets' semantics for normal programs extend easily to our setting; we just sketch 3 theorems (for the definitions see Section 2):

**Theorem 9 ([6])** *If $P$ is a call-consistent extended program, then $P$ has an answer set.*

**Theorem 10 ([12])** *If $P$ is a locally stratified extended program, then $P$ has exactly one answer set.*

**Theorem 11 ([6])** *If $P$ is a p.o.c. extended program, then the answer sets of $P$ coincide with the 4-valued Herbrand models of $P^\star$.*

## 6 CONCLUSION

The contribution of this paper is twofold:

(i) From the viewpoint of Fitting's programs on bilattices, we extend the programs on (the bilattice of) Belnap's logic by the addition of a non-monotonic operator /, and we show that this notion of extended programs corresponds to the one of Gelfond and Lifschitz.

(ii) From the viewpoint of the extended programs of Gelfond and Lifschitz, we provide them with a logical semantics in the style of Kunen, and we show that the underlying logic is precisely Belnap's logic.

## References

[1] J.J. Alferes and L.M. Pereira, On Logic Program Semantics with Two Kinds of Negation, in: *Proceedings ILPS'92 Joint International Conference and Symposium on Logic Programming* (1992).

[2] A.R. Anderson, N.D. Belnap Jr. and J.M. Dunn, *Entailment: the Logic of Relevance and Necessity.* (Princeton University Press, vol.2 p.506-541, 1992).

[3] K.R. Apt, H.A. Blair and A. Walker, Towards a Theory of Declarative Knowledge, in: J. Minker, ed., *Foundations of Deductive Databases and Logic Programming* (Morgan Kaufmann, 1988).

[4] N.D. Belnap Jr., A Useful Four-Valued Logic, in: J.M. Dunn and G. Epstein, eds., *Modern Uses of Multiple-Valued Logic* (Reidel, 1977).

[5] K.L. Clark, Negation as Failure, in: H. Gallaire and J. Minker, eds., *Logic and Databases* (Plenum, New York, 1978).

[6] F. Fages, Consistency of Clark's Completion and Existence of Stable Models, Technical Report 90-15, Ecole Normale Supérieure, Paris (1990), and *Methods of Logic in Computer Science* **1** (1994).

[7] F. Fages, Constructive Negation by Pruning, *to appear in J. of Logic Programming* (1996).

[8] M. Fitting, A Kripke-Kleene Semantics for Logic Programs, *J. of Logic Programming* **2** (1985).

[9] M. Fitting, Bilattices and the Theory of Truth, *J. of Philosophical Logic* **18** (1989).

[10] M. Fitting, Bilattices and the Semantics of Logic Programming, *J. of Logic Programming* **11** (1991).

[11] M. Fitting, The Family of Stable Models, *J. of Logic Programming* **17** (1993).

[12] M. Gelfond and V. Lifschitz, The Stable Model Semantics for Logic Programming, in: *Proceedings ICLP'88, $5^{th}$ International Conference on Logic Programming* (1988).

[13] M. Gelfond and V. Lifschitz, Logic Programs with Classical Negation, in: *Proceedings ICLP'90, $7^{th}$ International Conference on Logic Programming* (1990).

[14] M. Gelfond and V. Lifschitz, Classical Negation in Logic Programs and Disjunctive Databases, *New Generation Computing* **9** (1991).

[15] M. Gelfond and V. Lifschitz, Representing Actions in Extended Logic Programs, in: *Proc. ILPS'92 Joint International Conference and Symposium on Logic Programming* (1992).

[16] M. Ginsberg, Multivalued Logics: a Uniform Approach to Reasoning in Artificial Intelligence, *Computational Intelligence* **4** (1988).

[17] K. Kunen, Negation in Logic Programming, *J. of Logic Programming* **4** (1987).

[18] K. Kunen, Signed Data Dependences in Logic Programs, *J. of Logic Programming* **7** (1989).

[19] P. Stuckey, Constructive Negation for Constraint Logic Programming, in: *Proceedings LICS'91 International Conference on Logic in Computer Science* (1991).

[20] M.H. Van Emden and R. Kowalski, The Semantics of Predicate Logic as a Programming Language, in: *J. of the Association for Computing Machinery* **23** (1976).

[21] G. Wagner, A Database Needs Two Kinds of Negation, in: B. Thalheim, J. Demetrovics and H.-D. Gerhardt, eds., MFDBS'91 (Springer, 1991).