Combining explicit Negation and Negation by Failure via Belnap's Logic

François FAGES
Paul RUET*

Laboratoire d'Informatique, URA 1327 du CNRS
Département de Mathématiques et d'Informatique
Ecole Normale Supérieure
*Thomson - LCR

LIENS - 94 - 15

September 1994

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

Paul Ruet
Thomson - LCR
Domaine de Corbeville
91404 Orsay Cedex, France
ruet@lcr.thomson.fr

&

François Fages
LIENS - CNRS
Ecole Normale Supérieure
45 rue d'Ulm, 75005 Paris, France
fages@dmi.ens.fr

August 31, 1994

Abstract

This paper deals with logic programs containing two kinds of negation: negation as failure and explicit negation, allowing two different forms of uncertainty 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 various forms of non-monotonic reasoning, by means of negative litterals. Simply infering 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 infered if A is false in the least Herbrand model of the program. In the program's completion approach, the clauses defining the 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 highly undecidable. 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 [10] 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 litteral should be infered. 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 [16] for the negation as failure rule w.r.t. the three-valued completion of the program, followed by stronger completeness results obtained by [17] for the constructive negation rule.

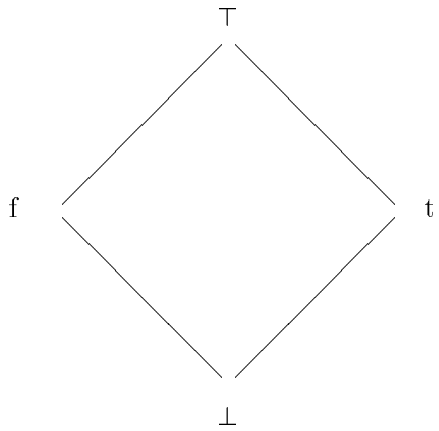
In this paper we study extended logic programs as introduced by Gelfond and Lifschitz [11, 12] (see also [19, 1]) to deal with two kinds of negation: explicit negation allowed in clause heads and bodies and negation by failure allowed in clause bodies only. These two negations allow two different forms of uncertainty reasoning in the presence of incomplete information: to infer $notA$, you may want to know that A cannot be infered (it is the case of negation by failure $/A$), or you may require an explicit inference process for $notA$, when e.g. the closed world assumption cannot be made on A (it is the case of explicit negation $\neg A$).

We develop a 9-valued Kunen-style semantics for extended programs and study the existence of 4-valued Belnap's models 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 [9], it corresponds rather to an extension of this frame to incorporate negation as failure. Furthermore we show that the answer sets of [12] 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 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]).

The best 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 $\{false\}$, $\bot$ for $\emptyset$ (indicating a lack of information) and $\top$ for $\{true, false\}$ (indicating inconsistency).



This set of truth values has two natural orderings: one is the subset relation, the *knowledge* ordering (*vertical* ordering in the picture), and the other one (*horizontal* ordering in the picture) represents the degree of *truth*. 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* [13, 8].

Meet and join under the truth ordering are denoted $\wedge$ and $\vee$; they are generalizations of the usual *conjunction* and *disjunction*. Meet and join under the knowledge ordering 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 [9] 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. the knowledge ordering, 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 consider also connectives that are non-monotonic w.r.t. the knowledge ordering, namely negation as failure *slash* /, which flips the diagram from bottom-left to top-right, switching f and $\top$, and t and $\bot$. Hence our logic programming with two negations is not an instance of logic programming on a bilattice in the sense of Fitting.

3 SYNTAX OF EXTENDED LOGIC PROGRAMS

The idea behind our extended programs is that, when dealing with possibly incomplete or inconsistent information, the deduction process of the falsity of a sentence A has

to be independent from that of the truth of A. Such a deduction process (e.g. logic program) 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 distinct 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 /).

Syntactically, our extended programs look like the ones of Gelfond and Lifschitz [11, 12], with negation as failure denoted / instead of *not*. This is intensional, and indeed we will see (Section 5) that their answer sets are (almost) models of the completed program, if negation as failure is interpreted by the connective / of Belnap's logic. (Gelfond and Lifschitz's definition of answer set includes a rule that globalizes contradictions, saying that a program which implies both A and $\neg A$ implies everything, and that is removed in our definition.)

Though sharing some resemblance with Fitting's logic programming on bilattices (see [9] and [13] for motivation on bilattices), there is one essential difference: the connectives considered by Fitting are all monotone in the inference ordering (the "knowledge" ordering of the bilattice), whereas negation as failure has to be represented by a non-monotone connective (/ is indeed not monotone under this knowledge ordering). Roughly speaking, as we shall see in the next Section, our extended programs are to Fitting's programs on Belnap's logic, what programs with negation as failure are to positive programs.

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 s.c. 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 $/L$ of a classical literal L. A literal of the form $/L$ is called a *slashed literal*. A *clause* is of the form:

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

where the L_i 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.

Call-consistency: 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 [16]) $p \sqsupseteq_{+1} q$ 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 $p \sqsupseteq_i q \ \& \ q \geq_j r \Rightarrow p \geq_{i \cdot j} r$.

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

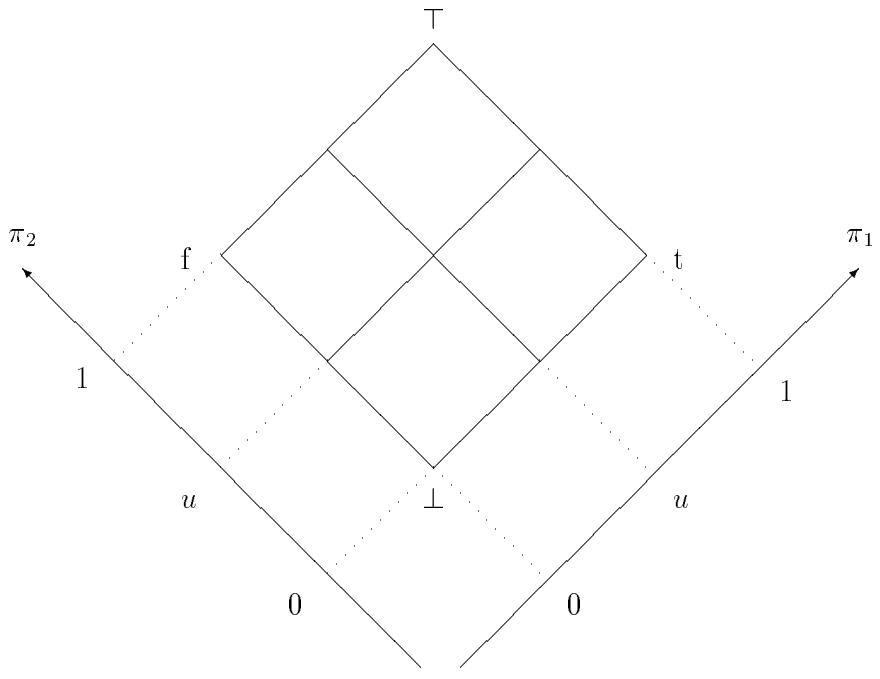
Stratified and p.o.c. extended programs: following [10] we say that an extended program P is *stratified* (resp. *locally stratified*) iff no predicate symbol or $\neg$-negated predicate symbol (resp. ground symbol) depends on itself through at least one / negation (but any positive number, odd or even).

Following [6] we say that an extended program P is *positive order consistent (p.o.c.)* iff the graph of positive (w.r.t. /) dependance among ground classical litterals has no infinite decreasing chain.

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 shall be provided with a 9-valued Kunen-sytle semantics.



Truth values are then handled as points in a square, whose coordinates are set according to the picture. For instance, $t = (1, 0)$, $\bot = (0, 0)$... 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 2 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 $/(v_1, v_2) = (\overline{v_1}, v_2)$.

In addition to the preceding connectives, we shall need one more connective $\leftrightarrow$ for the definition of the completed program (see below): $v \leftrightarrow w$ is t iff $\pi_1(v) = \pi_1(w)$ and f otherwise.

4.1 COMPLETED PROGRAM AND 9-VALUED MODELS

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

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

where $X_1 \ldots X_n$ are new variables.

Let P be an extended program and $L(X_1, ..., X_n) \text{ :- } \psi_i$ $(1 \leq i \leq m)$ 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]).

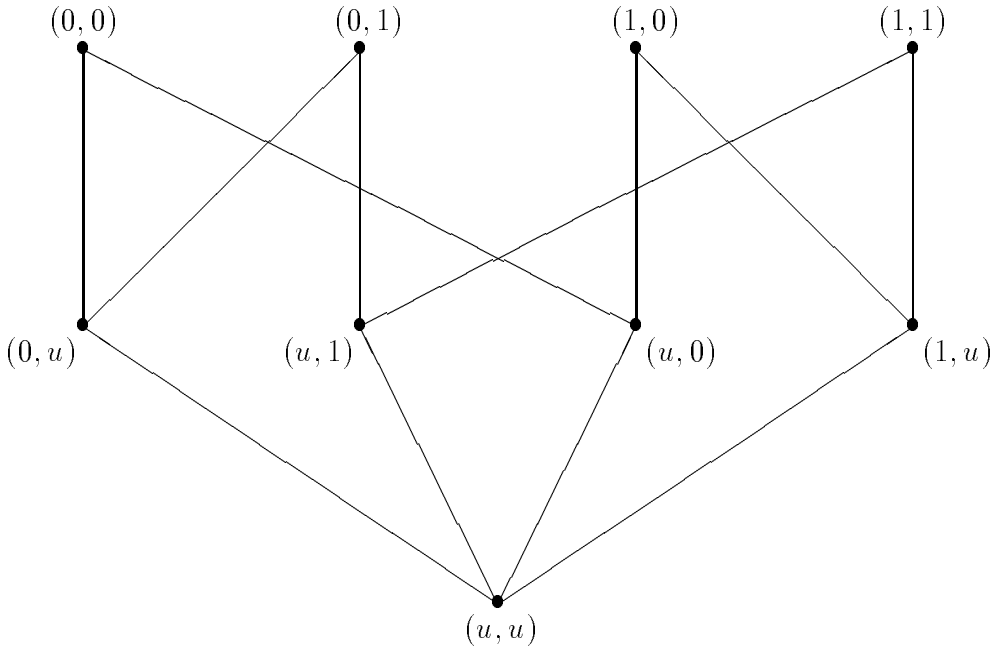
9-Valued structures.
A 9-*valued structure* $\mathcal{A}$ for the fixed language $\mathcal{L}$ consists of a nonempty set A (the *domain* of interpretation), and:
(i) for every n-ary function symbol f, $\mathcal{A}(f) : A^n \to A$ is a n-ary function,
(ii) for every n-ary predicate symbol p other than $=$, $\mathcal{A}(p)$ 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 $\mathcal{A} \models_4 P^\star$.

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 formula ϕ, $\pi_1(\mathcal{A}(\phi)) \leq_k \pi_1(\mathcal{B}(\phi))$ 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 $\leq_k$.

Intuitively, an extension of $\mathcal{A}$ is "less undefined" than $\mathcal{A}$. It is a concept different from that of "expansion" (see [16]) 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 come back temporary 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 ϕ, $\mathcal{M}(\phi) \leq_k \mathcal{N}(\phi)$. 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 1 *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 Ψ 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:

1.
   - $\pi_1(v) = 1$ 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.

2.
   - $\pi_2(v) = 1$ 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.

Intuitively, a clause in an extended program means the new truth value that shall be assigned to the head has to be the lub (w.r.t. the knowledge ordering $\preceq_k$ defined above) of the previous one and the one assigned to the body.

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

Theorem 1 *Let $\mathcal{A}$ be a 9-valued structure. $T_P(\mathcal{A}) = \mathcal{A}$ iff $\mathcal{A} \models_9 P^\star$.*

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 [7]), 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 the preceding Section:

Theorem 2 *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.*

In the next Section, we shall give proofs of these theorems through a "faithful" translation from extended programs to normal programs.

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 $L_0 :- L_1, \ldots, L_m, /L_{m+1}, \ldots, /L_n$ by $L_0^\neg :- L_1^\neg, \ldots, L_m^\neg, /L_{m+1}^\neg, \ldots, /L_n^\neg$. In $P^\neg$, /

stands for negation as failure. The notation / is unusual, but using $\neg$ could have lent to confusion. Similarly we can define $\Sigma^{\neg}$ for any set Σ 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 distinct 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 Π is a normal program, Ψ_Π denotes the immediate consequence operator of Kunen [16] on 3-valued structures, and $\Pi^\star$ denotes Clark's completed program [5].

Proposition 2 *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) 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 ϕ, $\pi_1(\mathcal{A}(\phi)) \leq_k \pi_1(\mathcal{B}(\phi))$ 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 ϕ, $\mathcal{A}^{\neg}(\phi) \leq_k \mathcal{B}^{\neg}(\phi)$ (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 ϕ 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 1: it follows directly from Proposition 2 (iii, iv, v) and [16] (Lemma 3.1). □

Proof of Theorem 2: it is an easy consequence of Proposition 2 (i, ii, v) and [16] (Theorem 3.4 and Corollary 3.5: namely if Π 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 $\mathcal{N} \models_2 \Pi^\star$). □

Operational Semantics.
The reduction to normal programs allows to consider that litterals A and $\neg A$ have a "separate life". Hence SLDNF resolution [16] and constructive negation [17] provide extended programs with operational semantics, in the following way: the answer to an 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.

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 droping their rule that 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 $L \text{ :- } L_1, \ldots, L_m$ 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 $L \text{ :- } L_1, \ldots, L_m, /L_{m+1}, \ldots, /L_n$ 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 3 *If S is an answer set of an extended program P, then $m(S)$ is a 4-valued model of $P^\star$.*

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

- If $\pi_1(m(S)(L)(a_1, ..., a_n)) = 1$ then $L(a_1, ..., a_n) \in S$; because of the definition of β, there must be a ground rule instance $R^S = (L(a_1, ..., a_n) \text{ :- } L_1, \ldots, L_k)$ in P^S such that $L_1, \ldots, L_k \in S$. This rule comes from a rule $R = (L \text{ :- } L_1, \ldots, L_k, /L_{k+1}, \ldots /L_n)$ 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 $(L(a_1, ..., a_n) \text{ :- } L_1, \ldots, L_k)$, 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.

□

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 2 important theorems (for the definitions see Section 3):

Theorem 4 (Gelfond and Lifschitz [10]) *If P is a locally stratified extended program, then P has exactly one answer set.*

Theorem 5 (Fages [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 APPLICATION TO KNOWLEDGE REPRESENTATION

The difference between $/p$ and $\neg p$ is essential when one cannot assume that the available information about p is complete, in other words when the "closed world assumption" is not applicable to p.

It arises when a database misses some information: the two kinds of negation allow us then to distinct between *temporary* and *definitive* lack of information. Similar facts can also arise due to human factors, like in the following example (taken from [12]).

A College uses the following rules for awarding scholarships to its students:

1. Every student with a GPA of at least 3.8 is eligible.
2. Every minority student with a GPA of at least 3.6 is eligible.
3. No student with the GPA under 3.6 is eligible.
4. The students whose eligibility is not determined by these rules are interviewed by the scholarship committee.

These rules can be encoded in the following extended program:

$eligible(X) \text{ :- } highGPA(X).$
$eligible(X) \text{ :- } minority(X), fairGPA(X).$
$\neg eligible(X) \text{ :- } \neg fairGPA(X).$
$interview(X) \text{ :- } / \, eligible(X), / \, \neg eligible(X).$

Assume this program is used in conjunction with a database containing the following facts about one of the students:

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

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. Our Kunen-style semantics (weaker than the answer sets' semantics) suffices to deduce the expected assertion $interview(ann)$, i.e. $interview(ann)$ is a 4-valued consequence of the completed program.

7 CONCLUSION

The contribution of this paper is twofold:

1. From the viewpoint of Fitting's programs on bilattices, we extend the programs on (the bilattice of) Belnap's logic by the adjunction of a non-monotonic operator $/$, and we show that this notion of extended programs corresponds to the one of Gelfond and Lifschitz.
2. 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.

The generalization of our work to logic programs on a general *bilattice* seems to raise new problems, at least in the spirit of this paper, as the simplicity of our semantics relies

on the cartesian product structure of the set of Belnap's truth values w.r.t. classical logic.

Though a generalization in the spirit of *annotated* programs [14] (which subsume bilattice programs) could be interesting and more natural, and constitute the matter of further work.

References

[1] ALFERES J.J. and PEREIRA L.M., On Logic Program Semantics with Two Kinds of Negation. *Proc. ILPS'92 Joint Int. Conf. and Symp. on Logic Programming* (1992).

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

[3] APT K.R., BLAIR H.A. and WALKER A., Towards a Theory of Declarative Knowledge. *In: J. Minker (ed.), Morgan Kaufmann:* Foundations of Deductive Databases and Logic Programming (1988).

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

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

[6] FAGES F., 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] FITTING M., A Kripke-Kleene Semantics for Logic Programs. *J. of Logic Programming* 2 (1985).

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

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

[10] GELFOND M. and LIFSCHITZ V., The Stable Model Semantics for Logic Programming. *Proc. ICLP'88 5^{th} Int. Conf. on Logic Programming* (1988).

[11] GELFOND M. and LIFSCHITZ V., Logic Programs with Classical Negation. *Proc. ICLP'90 7^{th} Int. Conf. on Logic Programming* (1990).

[12] GELFOND M. and LIFSCHITZ V., Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing* 9 (1991).

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

[14] KIFER M. and SUBRAHMANIAN V.S., Theory of Generalized Annotated Logic Programming and its Applications. *J. of Logic Programming* 12 (1992).

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

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

[17] STUCKEY P., Constructive Negation for Constraint Logic Programming. *Proc. LICS'91 Int. Conf on Logic in Computer Science* (1991).

[18] VAN EMDEN M.H. and KOWALSKI R., The Semantics of Predicate Logic as a Programming Language. *J. of the Association for Computing Machinery* 23 (1976).

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