On Logic Program Semantics with Two Kinds of Negation

José Júlio Alferes
Luís Moniz Pereira
CRIA Uninova and DCS, U.Nova de Lisboa
2825 Monte da Caparica, Portugal
{jja|lmp}@fct.unl.pt

Abstract

Recently several authors have stressed and showed the importance of having a second kind of negation in logic programs for use in deductive databases, knowledge representation, and nonmonotonic reasoning [6, 7, 8, 9, 13, 14, 15, 24].

Different semantics for logic programs extended with $\neg$-negation (extended logic programs) have appeared [1, 4, 6, 9, 11, 12, 17, 19, 24] but, contrary to what happens with semantics for normal logic programs, there is no general comparison among them, specially in what concerns the use and meaning of the newly introduced $\neg$-negation.

The goal of this paper is to contrast a variety of these semantics in what concerns their use and meaning of $\neg$-negation, and its relation to classical negation and to the default negation of normal programs, here denoted by *not*.

To this purpose we define a parametrizeable schema to encompass and characterize a diversity of proposed semantics for extended logic programs, where the parameters are two: one the axioms $AX_\neg$ defining $\neg$-negation; another the minimality conditions not_{cond} defining *not*-negation.

By adjusting these parameters in the schema we can then specify several semantics involving two kinds of negation [6, 11, 17, 19, 24]. Other semantics, dealing with contradiction removal [1, 4, 12, 22], are not addressed yet by the schema. The issue will be briefly touched upon in section 5, as well as that of incorporating disjunction.

The structure of the paper is as follows: we begin with preliminary definitions; in section 2 we present the parametrizeable schema; next we present properties important for the study of extended logic program semantics, and show for various $AX_\neg$ whether or not the resulting semantics complies with such properties; afterwards, in section 4, we reconstruct the plurality of semantics for extended logic programs in our schema by specifying for each their set $AX_\neg$ and their condition not_{cond}; finally we address further developments.

1 Language

An *extended logic program* (or x–program) P is a set of rules of the form: $H \leftarrow B_1, \ldots, B_n,\ not\ C_1, \ldots, not\ C_m$ where $n, m \geq 0$ and H, B_i, C_i are objective literals. An *objective literal* is either an atom A or its negation $\neg A$ where $\neg$ is one kind of negation introduced. The symbol *not* stands for negation by default, an additional kind of negation. $not\, L$ is termed a *default literal*. *Literals* are either objective or default ones.

By the *language* of P, $\mathcal{L}(P)$ we mean the set of all ground literals built from the symbols occuring in P.

Whenever unambigous we refer to extended logic programs simply as programs. Without loss of generality [18], a set of rules stands for all its ground instances wrt $\mathcal{L}(P)$. By *normal program* we mean any extended logic program where all objective literals are atoms, so that only default negation is present.

In the sequel, we translate every x–program P into a set of general clauses $\bar{P}$, which we dub a *clausal logic program*. The models and interpretations of clausal logic programs are the classical models and interpretations of sets of general clauses. Propositions of the form not_A (translation in $\bar{P}$ for $not\ A$ in P) are called *default* ones, all other propositions being the *objective* ones.

2 Generic semantics for programs with two kinds of negation

Within this section we present the above mentioned parametrizeable schema.

First we begin by defining two generic semantics for normal logic programs extended with an extra kind of negation: one extending the stationary semantics [18, 19] for normal programs (itself equivalent to well founded semantics [23]); another extending the stable model semantics [5]. We dub these semantics generic because they assume little about the extra kind of negation introduced. The meaning of the negation by default is however completely determined in each of the two semantics (both stationary and stable models) that we present.

Subsequently we generalize the schema in order to parametrize it as well w.r.t. negation by default.

2.1 Stationary semantics for programs with two kinds of negation

Here we redefine the stationary semantics of [19] in order to parametrize it with a generic second type of negation in addition to negation by default.

We start by defining stationary expansion of normal programs as in [19].

Definition 2.1 (Minimal Models) **[19]** *By minimal model of a theory T we mean a model M of T with the property that there is no smaller model*

N of T which coincides with M on default propositions. If a formula F is true in all minimal models of T then we write: $T \models_{CIRC} F$ and say that F is minimally entailed by T.

This amounts to the predicate circumscription $CIRC(T; \mathcal{O}; \mathcal{D})$ of theory T in which objective propositions $\mathcal{O}$ are minimized and default propositions $\mathcal{D}$ are fixed.

Definition 2.2 (Stationary Expansion of Normal Programs) [19] *A stationary expansion of a normal program P is any consistent theory P^* which satisfies the fixed point condition $P^* = \bar{P} \cup \{not\_L \mid P^* \models_{CIRC} \neg L\}$ where L is any arbitrary objective literal, $\bar{P}$ is the program obtained from P by replacing every literal of the form $not\, L$ by not_L, and where the distributive axiom $not(\neg L) \equiv \neg not\, L$ is assumed.*

Note that $\bar{P}$ is always a Horn set of clauses.

Example 1 Let $P = \{a \leftarrow not\, a;\quad b \leftarrow not\, a, c;\quad d \leftarrow not\, b\}$ whose clausal program is $\bar{P} = \{a \vee \neg not\_a;\quad b \vee \neg not\_a \vee \neg c;\quad d \vee \neg not\_b\}$. The only expansion of P is $P^* = \bar{P} \cup \{not\_b, not\_c, \neg not\_d\}$. In fact the minimal models of P^* are (throughout the examples we exhibit for clarity all literals):

$$\begin{array}{lcccccccl} \{ & not\_a, & not\_b, & not\_c, & \neg not\_d, & a, & \neg b, & \neg c, & d \;\} \\ \{ & \neg not\_a, & not\_b, & not\_c, & \neg not\_d, & \neg a, & \neg b, & \neg c, & d \;\} \end{array}$$

As P^* entails $\neg b$, $\neg c$, and d, it must contain $\{not\_b, not\_c, not\_\neg d\}$, and no more default literals. Note that by the distributive axiom $not\_\neg d \equiv \neg not\_d$.

In order to extend this definition to logic programs with a generic second kind of negation, we additionally transform the literals it negates into new atoms:

Definition 2.3 (Clausal Program of P $\bar{P}$) *The clausal program $\bar{P}$ of an x–program P is the clausal Horn program obtained by first denoting every literal in $\mathcal{L}(P)$ of the form: $\neg A$ by a new atom $\bar{A}$; $not\, A$ by a new atom not_A; $not\, \neg A$ by a new atom $not\_\bar{A}$, and then replacing in P those literals by their new denotation and, finally, reinterpreting the rule connective $\leftarrow$ as material implication $\Leftarrow$.*

Example 2 Let $P = \{a \leftarrow \neg b\}$. The clausal program $\bar{P}$ is $\{a \vee \neg \bar{b}\}$. Note that operations must be performed in the above order. If we would begin by first reinterpreting $\leftarrow$ as material implication we would get $\{a \vee b\}$.

Remark 2.1 *The symbol $\neg$ in $\bar{P}$ stands always for classical negation; the symbol $\neg$ in P stands for a negation whose meaning will be determined by the way it relates to the $\neg$ of $\bar{P}$, through axioms added to $\bar{P}$ (cf. definition 2.6). The context always determines the correct reading; be sure not to confuse the two uses.*

The resulting models of an x–program are determined by the models of its clausal program through an inverse transformation:

Definition 2.4 (Meaning of a clausal program P^*) *The meaning of a clausal program P^* is the union of the sets of all atoms:*

$$\begin{array}{rll@{\qquad\qquad}rll} A & \textit{such that} & P^* \models A & not\, A & \textit{such that} & P^* \models not\_A \\ \neg A & \textit{such that} & P^* \models \bar{A} & not\,\neg A & \textit{such that} & P^* \models not\_\bar{A} \end{array}$$

In order to specify the second kind of negation we introduce in $\bar{P}$ axioms defining it. For example, if we want the second negation to be classical negation we must add to $\bar{P}$ the set of clauses $\{\bar{A} \Leftrightarrow \neg A \mid A \in \mathcal{L}(P)\}$, where $\Leftrightarrow$ denotes material equivalence. In this case we would expect the semantics of P to be the same whether or not the first part of the transformation to $\bar{P}$ takes place.

Since we want this generic semantics to be an extension to the stationary semantics, we must guarantee that the semantics of a program without any occurence of $\neg$-negation is the same as for stationary semantics, whatever kind of $\neg$-negation is used and defined in the generic schema. To that end we must first minimize the atoms in the language of P, and only afterwards do we minimize the bar-ed atoms.

Definition 2.5 ($M \lesssim N$) *Let M and N be two models of a program $\bar{P}$ and M_{pos} (resp. N_{pos}) be the subset of M (resp. N) obtained by deleting from it all literals of the form $\bar{L}$.*

We say that $M \lesssim N$ iff: $M_{pos} \subseteq N_{pos} \vee (M_{pos} = N_{pos} \wedge M \subseteq N)$.

This definition is similar to the classical one plus a condition to the effect that, say, model $M_1 = \{\bar{a}\}$ is smaller than model $M_2 = \{a\}$.

Minimal models are defined as in definition 2.1 but with this new $\lesssim$ relation. The equivalence between minimality and circumscription is made through the ordered predicate circumscription $CIRC(T; \mathcal{O}; \mathcal{D})$ of the theory T, in which objective propositions $\mathcal{O}$ are minimized, but minimizing first propositions not of the form $\bar{A}$ and only afterwards the latter, and where default propositions $\mathcal{D}$ are fixed parameters.

The definition of stationary expansion of x–programs is then a generalization of definition 2.2, parametrized by the set of axioms $AX_\neg$ defining $\bar{A}$, plus this new notion of ordered minimality.

Definition 2.6 (Stationary $AX_\neg$ Expansions of x–programs) *A stationary expansion of an $AX_\neg$ x–program P is any consistent theory P^* which satisfies the following fixed point condition:*

$$P^* = \bar{P} \cup AX_\neg \cup \{not\_L \mid P^* \models_{CIRC} \neg L\}$$

where L is any arbitrary objective literal, and $AX_\neg$ is the set of axioms for $\neg$-negation in P.

A stationary expansion P^* of a program P is obtained by adding to the corresponding clausal program $\bar{P}$ the axioms defining $\neg$–negation, and the negations by default not_L of those and only those literals L which are false in all minimal models of P^*. The meaning of negation by default is that, in any stationary expansion P^*, not_L holds if and only if P^* minimally entails $\neg L$. Note that the definition of $AX_\neg$ can influence, by reducing the number of models, whether or not $\neg L$ is in all minimal models of P^*.

From the distributive axiom above we obtain directly (as in [19]):

Proposition 2.1 *A consistent theory P^* is a stationary expansion of an $AX_\neg$ x–program P iff:*

- *P^* is obtained by augmenting $\bar{P} \cup AX_\neg$ with some default propositions not_A and $\neg not\_A$ where A is an objective proposition;*
- *P^* satisfies the conditions: $P^* \models not\_A \equiv P^* \models_{CIRC} \neg A$ and $P^* \models \neg not\_A \equiv P^* \models_{CIRC} A$ for any objective proposition A.*

Example 3 Consider program $P = \{p \leftarrow a; \quad p \leftarrow \neg a; \quad q \leftarrow not\, p\}$, where $\neg$ in P is classical negation, i.e. $AX_\neg = \{\bar{a} \Leftrightarrow \neg a, \bar{p} \Leftrightarrow \neg p, \bar{q} \Leftrightarrow \neg q\}$. The only stationary expansion of P is:

$$P_1^* = \bar{P} \cup AX_\neg \cup \{\neg not\_p, not\_\bar{p}, not\_q, \neg not\_\bar{q}, not\_a, \neg not\_\bar{a}\}$$

In fact, the only minimal model of P_1^* is: $\{\neg not\_p, not\_\bar{p}, not\_q, \neg not\_\bar{q}, not\_a, \neg not\_\bar{a}, p, \neg\bar{p}, \neg q, \bar{q}, \neg a, \bar{a}\}$ and the conditions of proposition 2.1 hold. Note how the $\lesssim$ relation prefers this model to other models that would be minimal if the classical $\leq$ were to be enforced. For example, the classically minimal model:

$$\{\neg not\_p, not\_\bar{p}, not\_q, \neg not\_\bar{q}, not\_a, \neg not\_\bar{a}, p, \neg\bar{p}, q, \neg\bar{q}, \neg a, \bar{a}\}$$

is not minimal when the $\lesssim$ relation is considered.

If $\neg$ in P is defined by $AX_\neg = \{\bar{a} \Rightarrow \neg a, \bar{p} \Rightarrow \neg p, \bar{q} \Rightarrow \neg q\}$, i.e. $\neg$ in P is a *strong* negation in the sense that it just implies classical negation in $\bar{P}$, the only stationary expansion of P is:

$$P_2^* = \bar{P} \cup AX_\neg \cup \{not\_p, not\_\bar{p}, \neg not\_q, not\_\bar{q}, not\_a, not\_\bar{a}\}$$

In fact, the only minimal model of P_2^* is: $\{not\_p, not\_\bar{p}, \neg not\_q, not\_\bar{q}, not\_a, not\_\bar{a}, \neg p, \neg\bar{p}, q, \neg\bar{q}, \neg a, \neg\bar{a}\}$ and the conditions of proposition 2.1 hold.

We now define the semantics of a program based on its stationary expansions.

Definition 2.7 (Stationary $AX_\neg$ Semantics of x–programs)
A stationary $AX_\neg$ model of a program P is the meaning of P^*, where P^* is a stationary $AX_\neg$ expansion of P.

The stationary $AX_\neg$ semantics of an x–program P is the set of all stationary $AX_\neg$ models of P.

If $S = \{M_k \mid k \in K\}$ is the semantics of P, then the intended meaning of P is $M = \bigcap_{k \in K} M_k$.

Example 4 The meaning of the program of example 3 is $\{p, \neg q, \neg a, not\ q, not\ a, not\ \neg p\}$ if we use classical negation, and $\{q, not\ p, not\ \neg p, not\ \neg q, not\ a, not\ \neg a\}$ if we use strong negation.

Example 5 Consider $P = \{a \leftarrow not\ b; \quad \neg a\}$, where $\neg$ is a weak form of negation determined by $AX_\neg = \{\neg A \Rightarrow \bar{A} \mid A \in \mathcal{L}(P)\}$.

The only stationary expansion of P is $P^* = \bar{P} \cup AX_\neg \cup \{\neg not\_a, \neg not\_\bar{a}, not\_b, \neg not\_\bar{b}\}$, determining thus the meaning of P as $M = \{a, \neg a, not\ b, \neg b\}$. The fact that both a and $\neg a$ belong to M is not a problem since the weak form of negation allows that. Note that $\neg A \Rightarrow \bar{A}$ is equivalent to $A \vee \bar{A}$, and allows models with both A and $\bar{A}$. Literal $\neg b$ also appears in M forced by the weak negation.

Now we state in what sense this semantics is a generalization of stationary semantics.

Theorem 2.1 (Generalization of Stationary Semantics)
Let P be a (nonextended) normal logic program, and let $AX_\neg$ be such that no clause of the form $A_1 \vee \ldots \vee A_n$ where $\{A_1, \ldots, A_n\} \subseteq \mathcal{L}(P)$ is a logical consequence of it.

M is a stationary $AX_\neg$ model of P iff M (modulo the $\neg$-literals) is a stationary model of P.

The restriction on the form of $AX_\neg$ is meant to avoid unusual definitions of $\neg$-negation as, for instance:

Example 6 Let $P = \{a \leftarrow b\}$, and $AX_\neg = \{a \vee \neg\bar{b}, \bar{b}\}$. P has a stationary $AX_\neg$ model $\{a, not\ \neg a, not\ b, \neg b\}$ which is not a stationary model of P. Note however that a is in the model because it is a logical consequence of $AX_\neg$ irrespective of the program.

2.2 The parametrizeable schema

Stable Models Semantics [5] has a one-to-one correspondence with stable expansions [10], and the latter can be obtained simply by replacing $\models_{CIRC}$ by $\models_{CWA}$ in the definition of stationary expansion of normal programs.

As with the stationary semantics of extended programs, a generic definition of stable semantics for extended programs can also be obtained, with $P^* \models_{CWA} \neg L$ as the condition for adding negation by default.

So, in general a new parameter in the schema is desireable in order to specify how default negation is to be added to an expansion.

Definition 2.8 **($\langle AX_\neg, not_{cond}\rangle$ Expansion of x–programs)**
A $\langle AX_\neg, not_{cond}\rangle$ expansion of an x–program P is any consistent theory P^* which satisfies the following fixed point condition:

$$P^* = \bar{P} \cup AX_\neg \cup \{not\_L \mid not_{cond}(L)\}$$

where L is any arbitrary objective literal.

The definition of a generic semantics is similar to that of stationary semantics.

Definition 2.9 **($\langle AX_\neg, not_{cond}\rangle$ Semantics of x–programs)**
A $\langle AX_\neg, not_{cond}\rangle$ model of a program P is the meaning of P^*, where P^* is a $\langle AX_\neg, not_{cond}\rangle$ expansion of P.

The semantics of a program P is the set of all $\langle AX_\neg, not_{cond}\rangle$ models of P. The intended meaning of P is the intersection of all models of P.

We define Stable $AX_\neg$ Semantics as the generic semantics where:

$$not_{cond}(L) = P^* \models_{CWA} \neg L.$$

With this definition proposition 2.1 and theorem 2.1 are also valid for stable models.

3 Properties required of extended logic programs

In this section we present properties important for the study of extended logic program semantics, and show for some $AX_\neg$ whether or not the resulting semantics complies with such properties. Here we examine the cases of:

- *classical negation* i.e. $AX_\neg = \{\bar{A} \Leftrightarrow \neg A \mid A \in \mathcal{L}(P)\}$
- *strong negation* i.e. $AX_\neg = \{\bar{A} \Rightarrow \neg A \mid A \in \mathcal{L}(P)\}$
- *weak negation* i.e. $AX_\neg = \{\neg A \Rightarrow \bar{A} \mid A \in \mathcal{L}(P)\}$
- *pseudo negation* i.e. $AX_\neg = \{\}$.

for stationary and stable semantics. In section 4.1 we study our semantics $WFSX$, which relates more directly both kinds of negation, and where $not_{cond}(L) = (P^* \models_{CIRC} \neg L \vee P^* \models_{CIRC} \bar{L})$, by introducing *explicit* negation.

We concentrate next only on properties determined by the $\neg$-negation. For comparative studies of semantics concerning negation by default see [2, 3].

Property 1 (Consistency) *A semantics is consistent iff, for any program P, if M is a stationary (resp. stable) model of P then for no atom $A \in \mathcal{L}(P)$ $\{A, \neg A\} \subseteq M$.*

In other words, a semantics is consistent if there is no need for testing for consistency within the final (stationary or stable) models of a program.

Example 7 Let $P = \{a \leftarrow not\, b; \quad \neg a \leftarrow not\, b\}$, where $\neg$ is weak negation. The only stationary expansion of P is $P^* = \bar{P} \cup \{\neg A \Rightarrow \bar{A} \mid A \in \mathcal{L}(P)\} \cup \{not\_b, not\_\bar{b}\}$. The only minimal model of $P^* = \{a, \neg not\_a, \bar{a}, \neg not\_\bar{a}, \neg b, \neg \bar{b}, not\_b, not\_\bar{b}\}$ is consistent. Although the meaning of $P^* = \{a, \neg a, not\_b, not\_\bar{b}\}$ is inconsistent.

As shown with the previous example, semantics with weak negation might not be consistent. The same happens for semantics with pseudo negation.

Semantics with classical or strong negation are consistent because, by the very definition of $AX_{\neg}$, for every atom $A \in \mathcal{L}(P)$, $\neg A \vee \neg \bar{A} \in P^*$, for every expansion P^* of any program P, and thus no model of P^* has A and $\bar{A}$. Thus the meaning of P^* can never have both A and $\neg A$.

Property 2 (Coherence) *A semantics is coherent iff, for any program P, whenever M is a stationary (resp. stable) model of P, if $\neg A \in M$ then $not\, A \in M$, and if $A \in M$ then $not\, \neg A \in M$.*

This property plays an important rôle if we consider the second kind of negation instrumental for specifying the falsity of literals. In that case coherence can be read as: *if A is declared false then it can be assumed false by default.* It turns out that, for both stationary and stable semantics, coherence is equivalent to consistency; hence the importance of coherence.

Theorem 3.1 *A stationary (or stable) semantics is coherent iff it is consistent.*

Proof: We prove this theorem here only for the case of a stationary semantics. The proof for stable semantics is quite similar and is omitted for brevity.

($\rightarrow$) If a stationary semantics is coherent then for any P^* every model M of P^* having $\bar{A}$ also has not_A. By proposition 2.1 $not\_A \in M \Leftrightarrow \neg A \in M$. Similarly we conclude that for every M if $A \in M$ the $\neg \bar{A} \in M$. Thus, given that models of clausal programs are always total, every model containing A does not contains $\bar{A}$, and every model containing $\bar{A}$ does not contain A, which is the consistency property.

($\leftarrow$) This part of the proof is similar and also omitted for brevity. $\diamondsuit$

Property 3 (Supportedness) *A semantics is necessarily supportive iff, for any program P, whenever M is a stationary (resp. stable) model of P then, for every objective literal L, if $L \in M$ there exists in P at least one rule of the form: $L \leftarrow B_1, \ldots, B_n, not\, C_1, \ldots, not\, C_m$ such that:*

$$\{B_1, \ldots, B_n, not\, C_1, \ldots, not\, C_m\} \subseteq M.$$

Since that for any program P, $\bar{P} \cup \{not\_L \mid P^* \models_{CIRC} \neg L\}$ is a Horn clausal program, a stationary (or a stable) semantics such that $AX_\neg$ does not contain any clause with positive propositions is necessarily supportive. Thus, semantics with pseudo or strong negation are necessarily supportive.

Semantics that introduce in $AX_\neg$ such clauses might not be necessarily supportive. For example, if $\neg$ is classical negation necessary supportedness does not hold:

Example 8 Let $P = \{a \leftarrow b; \quad \neg a\}$. The only stationary $\{\bar{A} \Leftrightarrow \neg A\}$ model is $M = \{not\ a, \neg a, not\ b, \neg b\}$. As $\neg b \in M$, and there is no rule for $\neg b$, the semantics is not necessarily supportive.

This property closely relates to the use of logic as a programming language. One does not expect objective literals to be true unless rules stating their truth condition are introduced; in other words, except for default propositions, no implicit information should be expected. We argue that if one wants the result of the previous program one should write $P = \{\neg b \leftarrow \neg a; \quad \neg a\}$ or, if disjunction is introduced, $P = \{a \leftarrow b;\ \neg a;\ b \vee \neg b\}$.

4 Fixing the set $AX_\neg$ and the condition $not_{cond}(L)$

In this section we reconstruct some semantics for x–programs simply by specifying the set $AX_\neg$ and the condition $not_{cond}(L)$ w.r.t. the generic semantics defined above. We contribute this way for a better understanding of what type of second negation each of those semantics uses, and what are the main differences among them. Moreover, we show a definition for the semantics of [11] in terms of a two-valued logic, and clarifiy the meaning of its explicit negation.

We begin by reconstructing answer-sets semantics [6] for programs with consistent answer-sets (equivalent to the semantics of [24]).

Theorem 4.1 (Answer-Sets Semantics) *An interpretation M is an answer-set of a program P iff M is a stable $\{\bar{A} \Rightarrow \neg A \mid A \in \mathcal{L}(P)\}$ model of P (modulo the syntactic representation of models).*

This theorem leads us to the conclusion that answer-sets semantics extends stable models semantics with strong negation. Thus, from the results of section 3, we conclude that answer-sets semantics is consistent, coherent and supportive.

Note that if instead of strong negation one uses pseudo negation and a test for consistency in the final models, the result would be the same. However, we think that the formalization as in theorem 4.1 is more accurate because the consistency there is dealt within the fixpoint condition, with no need for meta–level constraints, and the properties exhibited are those of strong negation and not of pseudo negation. For example, coherence (a property of strong negation but not of pseudo negation) is obeyed by answer-sets semantics.

One semantics extending well founded semantics with $\neg$-negation is presented in [17]. There it is claimed that the method used in [6] can be applied to semantics other than stable models, and that method is used to define the proposed semantics. It happens however that the meaning of $\neg$ is not the same as for answer-sets, in the sense that different $AX_\neg$s are used:

Theorem 4.2 (WFS plus $\neg$ as in [17]) *An interpretation M is an extended stable model of a program P iff M is a consistent stationary $\{\}$ model of P.*

Note the need for testing consistency in stationary models of the semantics. As seen in section 3, this semantics does not comply with coherence, which in our opinion is an important property of $\neg$, satisfied by answer-sets semantics.

Next we reconstruct the semantics presented in [19]. There the semantics is defined similarly to our generic definition, but where $AX_\neg$ is absent, and no transformation on literals of the form $\neg A$ and $not\,\neg A$ occurs. With this similarity the reconstruction follows easily:

Theorem 4.3 (Stationary Semantics with classical negation) *An interpretation M is a stationary model (in the sense of [19]) of a program P iff M is a stationary $\{\bar{A} \Leftrightarrow \neg A \mid A \in \mathcal{L}(P)\}$ model of P.*

Due to lack of space we do not present here the proofs of these theorems. Nevertheless we would like to note the importance of the new minimality relation $\bar{\leq}$ in the proof of the last one. We must ensure that the equivalences introduce via $AX_\neg$ do not affect minimal models of the program where $\bar{A}$ is replaced by $\neg A$ for every atom A. If the $\leq$ relation were used, this would not be the case since $\bar{A} \Leftrightarrow \neg A$ has two minimal $\leq$ models, $\{A, \neg\bar{A}\}$ and $\{\neg A, \bar{A}\}$. As the relation used is $\bar{\leq}$, the only minimal model of the equivalence is the latter, which is the same model of the program after the above transformation.

From the results of section 3 we conclude that this semantics does not comply with supportedness. Nevertheless, this semantics is the only one reconstructed here that introduces *real* classical negation into normal logic programs. We argue that, comparing it with semantics with strong negation, this is not a big advantage since, once disjunction is added to logic programs with strong negation, a programmer can state in the language that the negation is classical rather than strong. This can be done simply by adding rules of the form $A \vee \neg A$. Moreover, the programmer has the opportunity of stating which negation, strong or classical, is used for each of the atoms in the language, by adding or not for each of them such disjunctive rules. This issue will be further discussed in section 5.2.

4.1 Well Founded Semantics with Explicit Negation

Recently [11] we presented the semantics $WFSX$, which extends WFS to programs with a second type of negation that we dub *explicit*.

$WFSX$ follows naturaly from the single coherence requirement: $\neg L$ implies $not\, L$ (if L is explicitly false, L must be false) for any literal L.

Example 9 [11] Consider program $P = \{a \leftarrow not\, b,\quad b \leftarrow not\, a,\quad \neg a \leftarrow\}$.

If $\neg a$ were to be simply considered as a new atom symbol, say a', and WFS used to define the semantics of P (as suggested in [17]), the result would be $\{\neg a, not\,\neg b\}$, so that $\neg a$ is true and a is undefined. We insist that $not\, a$ should hold because $\neg a$ does. Accordingly, the WFSX of P is $\{\neg a, b, not\, a, not\,\neg b\}$, since b follows from $not\, a$.

The formal definition of this semantics is made by embedding that requirement into the very definition of (3-valued) interpretation, and then by straightforwardly adapting to it the formal techniques used for WFS in [16].

Since this semantics exhibits all the above mentioned properties of strong negation and is defined as an extension of WFS, it seems that it should be closely related to stationary semantics with strong negation. In fact:

Theorem 4.4 (WFSX Semantics and Strong Negation) *If an interpretation M is a stationary $\{\bar{A} \Rightarrow \neg A \mid A \in \mathcal{L}(P)\}$ model of P then M is an (WFSX) extended stable model of a program P.*

Thus $WFSX$ gives semantics to more programs and, whenever both semantics give a meaning to a program, the WF model of $WFSX$ is a (possibly proper) subset of that of stationary semantics with strong negation. The differences between $WFSX$ and stationary semantics with strong negation are best shown with the help of an example.

Example 10 Consider the program $P = \{a \leftarrow not\, a,\ b \leftarrow a,\ \neg b \leftarrow\}$, which has no strong stationary models. According to $WFSX$ its well founded model (and only extended stable model) is $M = \{\neg b, not\, b, not\,\neg a\}$. Note that M is not even a model in the (usual) sense of [16], because for the second rule the truth value of the head (false) is smaller than the truth value of the body (undefined).

In [11] a new truth valuation function is defined that agrees with the required definition of extended stable models. The main difference between this and the truth valuation function of [16] is that it allows models where a rule has an undefined body and a false head, just in case the explicit negation of the head is true. In other words in $WFSX$ $\neg$–negation overrides undefinedness. The truth of $\neg L$ is an *explicit* declaration that L is false.

Any semantics complying with proposition 2.1 cannot have M as a model of the program: not_b is in an expansion iff $\neg b$ is in all minimal models of that expansion, but if this is the case then (by the second rule clause) $\neg a$ should also be in all minimal models, which would necessarily entail $not\, a$ in the expansion.

In order to reconstruct $WFSX$ in our schema, a new condition for adding default negation is required, forcing a default literal not_L to assuredly belong to an expansion also in the case where the explicit negation $\bar{L}$ is in all minimal models.

Theorem 4.5 (WFSX Semantics) *An interpretation M is an extended stable model of a program P iff M is the meaning of a P^* such that:*

$$P^* = \bar{P} \cup \{not\_L \mid P^* \models_{CIRC} \neg L \text{ or } P^* \models_{CIRC} \bar{L}\}$$

Example 11 The program P of example 10 has now an expansion $P^* = \bar{P} \cup \{not\_b, \neg not\_\bar{b}, not\_\bar{a}\}$. In fact the minimal models are:

$$\{not\_a, not\_\bar{a}, not\_b, \neg not\_\bar{b}, a, \neg\bar{a}, b, \bar{b}\}$$
$$\{\neg not\_a, not\_\bar{a}, not\_b, \neg not\_\bar{b}, \neg a, \neg\bar{a}, \neg b, \bar{b}\}$$

In all those models we have: $\neg\bar{a}$ so we must introduce $not\_\bar{a}$; have $\bar{b}$ so we must introduce $\neg not\_\bar{b}$ and, by the second disjunct, not_b. Thus P^* is an expansion.

5 Further Developments

5.1 Contradiction Removal

In [1, 4, 12] semantics are presented that avoid some contradictions by making certain *not* literals undefined.

Example 12 Let $P = \{a \leftarrow not\, b; \quad \neg a\}$. None of the semantics discussed in section 4 gives a meaning to this program. Since there are no clauses for b in $\bar{P}$, $\neg b$ is in all minimal models. Thus $not\, b$ must be in the semantics. As $not\, b$ leads to a contradiction in a no meaning is defined for P. All the semantics referred above assign a meaning to P by undefining $not\ b$. For example with the semantics of [1] the meaning of P is $\{not\, a, \neg a, not\, \neg b\}$.

The idea underlying these semantics is: *if an assumption supports a contradiction then take back that assumption.*

In order to capture this idea within the schema described above in this paper one has to allow a *not* literal possibly not to be added to $\bar{P} \cup AX_\neg$ by the fixpoint condition, thus weaking that condition. For instance, for stationary expansions one could have instead: $P^* = \bar{P} \cup AX_\neg \cup S$ where $S \subseteq \{not\_L \mid P^* \models_{CIRC} \neg L\}$.

Example 13 In the previous example, P has several such fixpoints. For instance: $P_1^* = \bar{P} \cup AX_\neg$, $P_2^* = P_1^* \cup \{not\_\bar{b}\}$, $P_3^* = P_1^* \cup \{not\_a, \neg not\_\bar{a}, not\_\bar{b}\}$. The meaning of P_3^* is the meaning ascribed to P by [1].

As shown in this example there is a choice of subsets to consider. To choose a definite subset we need a new parameter in the schema. If we choose the definite rule that elects the intersection of the maximal sets S satisfying the fixpoit condition, P_3^* is the only expansion.

The study of criteria leading to the designation of preferred subsets deserves further attention; [1] examines in detail a preference based on CWA assumptions.

5.2 Logic Programs with $\neg$-Negation and Disjunction

Given the similarities between this generic definition of semantics for x–programs and that of stationary semantics for normal logic programs, it is easy to extend the former for extended disjunctive logic programs (or disjunctive x–programs) based on the extension of the latter for disjunctive normal programs [19].

First we have to extend the definition of $\bar{P}$ for the case of disjunctive programs. This extension is obtained simply by adjoining to definition 2.3: "[...] *reinterpreting the connective* $\vee$ *in logic programs as classical disjunction*". With this new context we define:

Definition 5.1 (Stationary $AX_\neg$ Expansion of disj. x–programs) *A Stationary $AX_\neg$ expansion of a disjunctive x–program P is any consistent theory P^* which satisfies the following fixed point condition (where the distributive axiom $not\,(A \wedge B) \equiv not\,A \vee not\,B$ is assumed):*

$$P^* = \bar{P} \cup AX_\neg \cup \{not\,F \mid P^* \models_{CIRC} \neg F\}$$

where F is an arbitrary conjunction of positive (resp. negative) objective literals.

Given this definition the semantics follows similarly to section 2.

Example 14 Let $P = \{p \leftarrow not\,a; \quad p \leftarrow not\,\neg b; \quad a \vee \neg b\}$ and $AX_\neg$ be the axioms for explicit negation. The only stationary $AX_\neg$ expansion of P is

$$P^* = \bar{P} \cup AX_\neg \cup \{not\_\bar{a}, not\_b, \neg not\_p, not\_\bar{p}, not\_a \vee not\_\bar{b}, \neg not\_a \vee \neg not\_\bar{b}\}.$$

Thus the only stationary $AX_\neg$ model is $\{p, not\,\neg p, not\,\neg a, not\,b\}$.

Henceforth, the way is open for the study of the interaction between $\neg$ and disjunction in each of the semantics for x–programs, and comparisons between those semantics via disjunction. One result concerning the latter is the formalization of the comparison between the use of classical or explicit negation stated at the end of the previous section:

Theorem 5.1 *Let P be an x–program. M is a stationary $\{\bar{A} \Leftrightarrow \neg A\}$ model of P iff it is a stationary $\{\bar{A} \Rightarrow \neg A\}$ model of $P \cup \{A \vee \neg A \mid A \in \mathcal{L}(P)\}$.*

It is known [19] that a definition such as 5.1 makes program disjunctions exclusive. This is seen in example 14. In order to treat disjunctions as inclusive rather than exclusive, in nonextended disjunctive programs, it suffices to replace $\models_{CIRC}$ by $\models_{WECWA}$ in the definition of expansions [19], where $WECWA$ stands for Weak Extended Closed World Assumption [21] or Weak Generalized Closed World Assumption [20]. This is not the case for extended disjunctive programs.

Acknowledgements

We thank ESPRIT BRA COMPULOG (no. 3012), Instituto Nacional de Investigação Científica, Junta Nacional de Investigação Científica e Tecnológica, and Gabinete de Filosofia do Conhecimento for their support. Thanks to Joaquim Aparício, Gabriel David, and Luís Monteiro for their comments.

References

[1] J. N. Aparício, L. M. Pereira, and J. J. Alferes. Contradiction removal semantics with explicit negation. Technical report, AI Centre, Uninova, March 1992.

[2] J. Dix. Classifying semantics of logic programs. In A. Nerode, W. Marek, and V. S. Subrahmanian, editors, *Logic Programming and NonMonotonic Reasoning'91*, pages 166–180. MIT Press, 1991.

[3] P. M. Dung. On the relations between stable and well-founded semantics of logic programs. *Theoretical Computer Science*, 1992. To appear.

[4] P. M. Dung and P. Ruamviboonsuk. Well founded reasoning with classical negation. In A. Nerode, W. Marek, and V. S. Subrahmanian, editors, *Logic Programming and NonMonotonic Reasoning'91*, pages 120–132. MIT Press, 1991.

[5] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R. A. Kowalski and K. A. Bowen, editors, *5th Int. Conf. on Logic Programming*, pages 1070–1080. MIT Press, 1988.

[6] M. Gelfond and V. Lifschitz. Logic programs with classical negation. In Warren and Szeredi, editors, *7th Int.Conf. on Logic Programming*, pages 579–597. MIT Press, 1990.

[7] Katsumi Inoue. Extended logic programs with default assumptions. In Koichi Furukawa, editor, *8th Int. Conf. on Logic Programming'91*, pages 490–504. MIT Press, 1991.

[8] R. Kowalski. Problems and promises of computational logic. In John Lloyd, editor, *Computational Logic Symposium*, pages 1–36. Springer-Verlag, November 1990.

[9] R. Kowalski and F. Sadri. Logic programs with exceptions. In Warren and Szeredi, editors, *7th Int. Conf. on Logic Programming*. MIT Press, 1990.

[10] R. C. Moore. Semantics considerations on nonmonotonic logic. *Artificial Intelligence*, 25:75–94, 1985.

[11] L. M. Pereira and J. J. Alferes. Well founded semantics for logic programs with explicit negation. In B. Neumann, editor, *European Conf. on AI'92*. John Wiley & Sons, Ltd, 1992.

[12] L. M. Pereira, J. J. Alferes, and J. N. Aparício. Contradiction Removal within Well Founded Semantics. In A. Nerode, W. Marek, and V. S. Subrahmanian, editors, *Logic Programming and NonMonotonic Reasoning'91*, pages 105–119. MIT Press, 1991.

[13] L. M. Pereira, J. N. Aparício, and J. J. Alferes. Counterfactual reasoning based on revising assumptions. In Ueda and Saraswat, editors, *Int. Logic Programming Symp.'91*. MIT Press, 1991.

[14] L. M. Pereira, J. N. Aparício, and J. J. Alferes. Hypothetical reasoning with well founded semantics. In B. Mayoh, editor, *Scandinavian Conf. on AI'91*. IOS Press, 1991.

[15] L. M. Pereira, J. N. Aparício, and J. J. Alferes. Nonmonotonic reasoning with well founded semantics. In Koichi Furukawa, editor, *8th Int. Conf. on Logic Programming'91*, pages 475–489. MIT Press, 1991.

[16] H. Przymusinska and T. Przymusinski. Semantic issues in deductive databases and logic programs. In R. Banerji, editor, *Formal Techniques in Artificial Intelligence*. North Holland, 1990.

[17] T. Przymusinski. Extended stable semantics for normal and disjunctive programs. In Warren and Szeredi, editors, *7th Int. Conf. on Logic Programming*, pages 459–477. MIT Press, 1990.

[18] T. Przymusinski. Stationary semantics for disjunctive logic programs and deductive databases. In Debray and Hermenegildo, editors, *North American Conf. on Logic Programming'90*, pages 40–57. MIT Press, 1990.

[19] T. Przymusinski. A semantics for disjunctive logic programs. In Loveland, Lobo, and Rajasekar, editors, *ILPS'91 Workshop in Disjunctive Logic Programs*, 1991.

[20] A. Rajasekar, J. Lobo, and J. Minker. Weak generalized closed world assumptions. *Automated Reasoning*, 5:293–307, 1989.

[21] K. Ross and R. Topor. Inferring negative information from disjunctive databases. *Automated Reasoning*, 4:397–424, 1988.

[22] Chiaki Sakama. Extended well–founded semantics for paraconsistent logic programs. In *Fifth Generation Computer Systems*, pages 592–599. ICOT, 1992.

[23] A. Van Gelder, K. A. Ross, and J. S. Schlipf. The well-founded semantics for general logic programs. *Journal of ACM*, pages 221–230, 1990.

[24] G. Wagner. A database needs two kinds of negation. In B. Thalheim, J. Demetrovics, and H-D. Gerhardt, editors, *MFDBS'91*, pages 357–371. Springer-Verlag, 1991.