Default negated conclusions: why not ?

Carlos Viegas Damásio and Luís Moniz Pereira

CRIA, Uninova and DCS, U. Nova de Lisboa
2825 Monte da Caparica
Portugal
{cd|lmp}@fct.unl.pt

Abstract. We address the issue of the syntactical assymetry of logic programs: can one allow default negated literals in the heads ?
This issue has been resolved positively, with full generality for the Stable Models based Answer Sets Semantics and, more recently, for the Well-founded Model based Static Semantics, for a wide class of programs. The work by Jonker also explores this issue for IMEX semantics.
Here we examine one possible reading for default negation in the heads, and show how it is captured by our paraconsistent Well-founded Semantics with Explicit Negation, $WFSX_{\text{p}}$, through the use of a suitable program transformation, without thus requiring any new fixpoint semantics. Furthermore, we extensively compare the three above mentioned approaches to our own.

1 Introduction

One of the main issues in logic programming (LP) is the definition of semantics for negation(s). The first form of negation, negation as failure, was introduced at the dawn of logic programming, within the $PROLOG$ language. Many efforts of the logic programming community have focused on imparting a declarative semantics to negation as failure. This task has not been easy and it took fifteen years before generally accepted semantics were found.

Two main families of semantics for normal logic programs appeared in the late eighties, namely well-founded semantics (WFS) [11] based ones and stable model semantics (SM) [12] based ones. WFS is defined for every normal logic program and assigns it a unique meaning. Not so with SM. The main differences between these two semantics result from their treatment of infinite recursion through negation by default.

More recently, a second form of negation was proposed by several authors [19, 13, 23, 20, 4]) to provide a mechanism for explicitly declaring the falsity of literals which was not available before. The importance of extending LP with a second kind of negation, "$\neg$", has been stressed because of its use in deductive databases, knowledge representation, and non-monotonic reasoning. Different semantics for extended LPs with $\neg$-negation have appeared (e.g. [13, 21, 20, 24]). The "coherence principle" of [20] relates the two forms of negation, default and explicit: it stipulates that the latter entails the former, i.e if L ($\neg L$) is entailed then so is $not\,\neg L$ ($not\,L$), and it is adopted as a fundamental principle of the *WFSX* semantics [20].

The introduction of explicit negation requires being able to reason with, or at least detect, contradictory knowledge. Indeed, information is not only normally incomplete but contradictory as well. One specific and natural generalization of *WFSX* for paraconsistent extended logic programs, $WFSX_{\mathrm{p}}$, defined in [1], is taken as the base semantics in this paper and recapitulated below.

One issue, that of the syntactical asymmetry of (extended) logic programs arises. Is it not possible to have default negated literals in the heads of rules or even in disjunctive heads ?

This problem has been addressed and solved in its full generality by [18, 15] for Answer Sets Semantics, where a semantics is proposed based on the autoepistemic flavoured logic MBNF [17] (minimal belief and negation as failure). More recently, Przymusinski introduced Static Semantics [22], via a semantical framework extending the notions of stable, well-founded, and stationary (or partial stable models) for a wide class of logic programs, including disjunctive ones with default negated literals in rule heads. Also in the work by Jonker [16], this issue of having a form of default negation in the heads was explored.

In this paper we will study one possible understanding of default negation in the heads, captured by $WFSX_{\mathrm{p}}$, by making use of a suitable program transformation, and without thus requiring any new fixpoint semantics. This also provides additional ground as well for our claim of generality and of great expressive power of the $WFSX_{\mathrm{p}}$ semantics.

The approach in [22] views default negation in the heads as a kind of pruning device. First the static expansions are computed for the subprogram containing no default negation in the heads. The remaining rules do not create any new static expansion, but only possibly remove some of the static expansions for the normal logic program fragment.

The generalization of Answer Sets in [18] encompassing *not* literals in the heads, has a flavour similar to the proposal of Przymusinski, but defined only for a strictly smaller class of (general) logic programs.

In what follows, we assume the reader acquainted with the terminology of extended logic programs, in particular that atoms or explicitly negated atoms are called objective literals, and that default literals are the default negated objective literals. The set of all objective literals in the language of a program P is denoted by $\mathcal{H}_P$. For details see [13, 20]. All definitions and results in this paper obtain only for the ground instances of programs.

We start by recalling the alternating fixpoint definition for $WFSX_{\mathrm{p}}$ which appeared in [1]. Afterwards, we motivate and define the Generalized Paraconsistent Well-founded Semantics with Explicit Negation ($WFSX^{\mathrm{g}}_{\mathrm{p}}$). In subsequent sections we compare $WFSX^{\mathrm{g}}_{\mathrm{p}}$ with the three main approaches in the literature, namely Generalized Answer Sets, Static Expansions, and *IMEX* negation. In the last section we draw out some conclusions.

2 Paraconsistent *WFSX*

To tackle the issue of paraconsistency in extended logic programs, an extension of *WFSX* was put forward in [1], defined by means of an alternating fixpoint construction which we briefly review in this section. One fundamental notion is Gelfond-Lifschitz's program division operator. In definition 1 every literal $\neg A$ is understood as a new atom, say $\neg\_A$, unrelated to A. Therefore, the existence of a pair of literals $A - \neg A$ is not problematic. This holds for the program as well as the interpretation in that definition. Mark that the interpretation is assumed two-valued, and thus can be represented as a set of objective literals.

Definition 1. [12] Let P be an extended program and I a two-valued interpretation. The GL–transformation $\frac{P}{I}$ is the definite program obtained from P by removing all rules containing a default literal $not\, A$ such that $A \in I$, and by then removing all the remaining default literals from P. By definition, $\Gamma_P I$ is the least two-valued model of $\frac{P}{I}$, which is always obtainable using the T_P operator of [10].

To impose the coherence requirement, Alferes et al. resort to the semi-normal version of an extended logic program.

Definition 2. [20, 4] The semi-normal version P_s of a program P is obtained from P by adding to the (possibly empty) $Body$ of each rule $L \leftarrow Body$ the default literal $not\, \neg L$, where $\neg L$ is the complement of L wrt. explicit negation.

The semi-normal version of a program introduces a new anti-monotonic operator: $\Gamma_{P_s}(S)$. Below we use $\Gamma(S)$ to denote $\Gamma_P(S)$, and $\Gamma_s(S)$ to denote $\Gamma_{P_s}(S)$.

Theorem 3. [1] *The operator $\Gamma\Gamma_s$ is monotonic, for arbitrary sets of literals.*

Consequently every program has a least fixpoint of $\Gamma\Gamma_s$. This defines the semantics for paraconsistent logic programs. It also ensures that the semantics is well-defined, i.e. that it assigns meaning to every extended logic program.

Definition 4. [1] Let P be an extended logic program and T a fixpoint of $\Gamma\Gamma_s$; then $T \cup not\,(\mathcal{H}_P - \Gamma_s T)$ is a paraconsistent partial stable model of P (PSM_p). The paraconsistent well-founded model of P, $WFM_p(P)$, is the least PSM_p under set inclusion order.

To enforce consistency on the paraconsistent partial stable models, Alferes et al. need only insist the extra condition $T \subseteq \Gamma_s T$ be verified, which succintly guarantees that, for no objective literal, L and $not\, L$ simultaneously hold. So it automatically rejects contradictory models where L and $\neg L$ are both hold since otherwise, by coherence, they would also entail $not\, L$ and $not\, \neg L$. Therefore, $WFSX_{\mathrm{p}}$ generalizes *WFSX* by not imposing the extra condition. Furthermore, for normal logic programs it coincides with *WFS*. We have proven that appropriate extensions of Dix's strong and weak principles [8, 7] to the paraconsistent case are all obeyed by $WFSX_{\mathrm{p}}$.

Another important property satisfied by the $PSM_p s$ is supportedness:

Definition 5. Let P be an extended logic program. An interpretation I is a supported model of P iff

1. An objective literal L is in I iff there is a rule for L in P such that all literals in its body are true in I;
2. A default literal $not\,L$ is in I iff at least one of the following holds:
   (a) The explicit complement of L belongs to I, i.e. $\neg L \in I$;
   (b) Every rule for L has a literal L' in its body such that the default complement of L' is in I, i.e. $not\,L' \in I$.

Theorem 6. *Let P be an extended logic program and M a PSM_p of P. Then M is a supported model of P.*

The converse of the above theorem is not valid, i.e. there are supported models which are not fixpoints of $\Gamma\Gamma_s$. This can be the case when there are infinite positive dependencies among objective literals. In general, whenever programs have no such dependencies among literals, the supported models are isomorphic to the fixpoints of $WFSX_{\mathrm{p}}$. The importance of this result is due to the circumstance that any extended logic program P can be transformed into an equivalent program with no objective literals in the bodies of rules. This means that it is possible to find an equivalent program where the characterization of $WFSX_{\mathrm{p}}$ fixpoints by the supported models holds, i.e. a model-theoretic definition for $WFSX_{\mathrm{p}}$. This equivalent program is obtained by application of an appropriate partial evaluation transformation. For lack of space we cannot present the transformation.

3 Default negation in the heads

In this section we propose a variant of $WFSX_{\mathrm{p}}$ for a semantics coping with default negated literals in the heads. One motivation for such a semantics is to permit specifying the conditions, in the body of a rule, under which a specific default literal is to be mandatorily adopted. Our starting point will be the notion of supported models of the previous section. We will pursue the following intended meaning for a rule $not\,L \leftarrow Body$: "If the body is true then $not\,L$ must be true".

Mark that no consequence is warranted from the rule when its body is not entailed. This corresponds to a form of reasoning mechanism which explicitly states reasons for making default assumptions. This meaning introduces an asymmetry in the semantics of the full language because, in the case of objective literal conclusions, default reasoning about the objective literal occurs in the situation of all its rule bodies being false[1]. To do so, we start by augmenting the language of extended logic programs to allow default literals in the heads.

[1] A similar mechanism can be postulated for the rules with default negated heads. We have also explored this possibility, resulting on a semantics with unintuitive behaviour. We cannot present this semantics here for lack of space.

Definition 7 General logic programs. Given a language $\mathcal{L}$, a general logic program is a set of rules of the form $G_0 \leftarrow G_1, \ldots, G_m$ where each G_i is either an objective or a default literal with respect to the underlying language $\mathcal{L}$.

The notion of supported models for $\mathit{WFSX}_{\mathrm{p}}$ can easily be altered to consider the new type of rules. First, we extend the notion of supported models according to the intended semantic intuition.

Definition 8 Ultrasupported model. Let P be a general logic program and I an interpretation. The definition of ultrasupported model is obtained by adding the following alternative condition to the second case of definition 5.

(c) there is a rule for $\mathit{not}\, L$ in P such that all literals in its body are true in I.

We now proffer some examples.

Example 1. Let P_1 be the general program with the two facts a and $\mathit{not}\, a$, i.e. $P = \{a; \mathit{not}\, a\}$. This program has a single ultrasupported model $\{a, \mathit{not}\, a, \mathit{not}\, \neg a\}$.

The same unique ultrasupported model is assigned to $P_2 = \{a \leftarrow \mathit{not}\, a; \mathit{not}\, a\}$, as expected.

None of the three aproaches in the literature discussed in the introduction attribute any meaning to these programs.

Example 2. Consider program $P_3 = \{a \leftarrow \mathit{not}\, b; b \leftarrow \mathit{not}\, a; \mathit{not}\, a\}$ This program has two ultrasupported models: $\{\mathit{not}\, a, \mathit{not}\, \neg a, b, \mathit{not}\, \neg b\}$ and $\{a, \mathit{not}\, a, \mathit{not}\, \neg a, b, \mathit{not}\, b, \mathit{not}\, \neg b\}$. The latter is the minimal one under set inclusion ordering.

The approach of Przymusinski assigns a model isomorphic to $\{b, \mathit{not}\, a\}$. The program has a unique answer set, namely $\{b, \mathit{not}\, a, \mathit{not}\, \neg a, \mathit{not}\, \neg b\}$, which corresponds to our least ultrasupported model.

Example 3. Finally, let P_4 be the program $\{a \leftarrow b; b \leftarrow c; c \leftarrow \mathit{not}\, c; \mathit{not}\, b\}$. This program also has two ultrasupported models:

$$U_1 = \{a, \mathit{not}\, a, \mathit{not}\, \neg a, b, \mathit{not}\, b, \mathit{not}\, \neg b, c, \mathit{not}\, c, \mathit{not}\, \neg c\}$$
$$U_2 = \{\mathit{not}\, a, \mathit{not}\, \neg a, \mathit{not}\, b, \mathit{not}\, \neg b, \mathit{not}\, \neg c\}$$

This program has a single static expansion making every literal undefined, except for b which is false. This program has no answer sets according to [18], and for literals a, b and c Jonker's semantics assigns the same meaning as our least ultrasupported model.

Now we state the main result of the paper. Our motivation is to avoid having to extend the $\mathit{WFSX}_{\mathrm{p}}$ semantics to general logic programs, so as not to require a new fixpoint definition, but still capture the semantics of general programs. We will achieve our purpose by transforming any general logic program into an extended one, without loss of generality. We will show that this transformation preserves the ultrasupported models, modulo the introduction of new literals that can be abstracted out from the model. The coherence principle plays a fundamental role in this matter.

Definition 9 P^{not} **transformation.** Let P be a general logic program. The extended logic program P^{not} is constructed as follows, where A is an atom:

1. For each literal A ($\neg A$) in $\mathcal{H}_P$ program P^{not} contains the rule "$A \leftarrow A^p$" ("$\neg A \leftarrow A^n$"), where A^p (A^n) is a new objective literal not in $\mathcal{H}_P$;
2. For each rule of the form "$A \leftarrow Body$" ("$\neg A \leftarrow Body$") program P^{not} contains the rule "$A^p \leftarrow Body$" ("$A^n \leftarrow Body$");
3. For each rule of the form "$not\, A \leftarrow Body$" ("$not\, \neg A \leftarrow Body$") program P^{not} contains the rule "$\neg A^p \leftarrow Body$" ("$\neg A^n \leftarrow Body$").

Example 4. Consider program $\{\neg a \leftarrow not\, b; c \leftarrow b; not\, c; \neg a \leftarrow c; b; not\, \neg c \leftarrow a\}$ and its associated transformed program P^{not}:

$a \leftarrow a^p.$	$a^n \leftarrow not\, b.$	$b \leftarrow b^p.$	$b^p.$	$c \leftarrow c^p.$	$c^p \leftarrow b.$
$\neg a \leftarrow a^n.$	$a^n \leftarrow c.$	$\neg b \leftarrow b^n.$		$\neg c \leftarrow c^n.$	$\neg c^p.$
					$\neg c^n \leftarrow a.$

Theorem 10. *Let P be a general logic program. Then I is a supported model of P^{not} iff I', the restriction of I to the atoms of the language of P, is an ultrasupported model of P.*

The extension of $WFSX_{\mathrm{p}}$ to general logic programs is immediate:

Definition 11 $WFSX_{\mathrm{p}}^{\mathrm{g}}$**.** Let P be a general logic program. The semantics of P is given by the paraconsistent partial stable models of P^{not} restricted to the language of P. Each such model will be called a general paraconsistent partial stable model. The least one with respect to the knowledge ordering is dubbed the general paraconsistent well-founded model, or simply paraconsistent well-founded model, and denoted by $WFM_{\mathrm{p}}^{\mathrm{g}}(P)$.

One issue to be settled is whether there always exists a least general paraconsistent well-founded model. Indeed there is.

Proposition 12. *Let P be any general logic program. Its general paraconsistent well-founded model exists and is unique.*

It might be argued that we cannot define our semantics for general logic programs just by making use of the P^{not} transformation. Notice first that all paraconsistent partial stable models of P^{not} are supported models of P^{not} and therefore ultrasupported models of P. But we know that the converse does not hold in general; however for programs free of positive cycles we have an "equivalence". So for this class of programs our definition makes perfect sense. Also mark that the P^{not} transformation does not introduce positive cycles that were not already present in P and no positive cycle exists using a rule with a default literal in the head. Therefore the supported models having no fixpoint counterpart are due to the same type of problem arising already with the $WFSX_{\mathrm{p}}$. Consequently, the definition of the more general $WFSX_{\mathrm{p}}^{\mathrm{g}}$ is the natural extension of $WFSX_{\mathrm{p}}$ to general logic programs.

Example 5. Consider the general logic program P_4 of example 3. By performing the P^{not} transformation we obtain the following extended logic program:

$$\begin{array}{llll} a \leftarrow a^p. & b \leftarrow b^p. & \neg b^p. & c \leftarrow c^p. \\ a^p \leftarrow not\, b. & b^p \leftarrow c. & & c^p \leftarrow not\, c. \end{array}$$

This program has two supported models, depicted below:

$$\begin{array}{l} S_1 = \{a, a^p, \neg b^p\} \cup not\,\{b, \neg a, \neg b, \neg c, \neg a^p, b^p, \neg c^p, a^n, \neg a^n, b^n, \neg b^n, c^n, \neg c^n\} \\ S_2 = \{a, b, c, a^p, b^p, \neg b^p, c^p\} \cup not\,\mathcal{H}_{P^{not}} \end{array}$$

corresponding, as expected, to the following ultrasupported models U_1 and U_2:

$$U_1 = \{a\} \cup not\,\{b, \neg a, \neg b, \neg c\} \; U_2 = \{a, b, c\} \cup not\,\{a, b, c, \neg a, \neg b, \neg c\}$$

We have that $WFSX_{\mathrm{p}}$ is powerful enough to capture a semantics with default negated literals in rule heads. The converse question can also be posed:

- Are general programs under $WFSX_{\mathrm{p}}^{\mathrm{g}}$, without explictly negated literals, capable of simulating $WFSX_{\mathrm{p}}$?

The answer is yes and is given by the program transformation of definition 13 and theorem 14. This means that we can either introduce default negated atoms in the heads of rules or introduce explicit negation in the language of programs: the two semantics have the same expressive power.

Definition 13. Let P be an extended logic program. Construct general program $P^{no\neg}$ as follows:

- Substitute very literal A, $\neg A$, $not\, A$ and $not\, \neg A$ in P by, respectively, A^p, A^n, $not\, A^p$ and $not\, A^n$ in $P^{no\neg}$;
- For each atom in the language of P add the rules "$not\, A^p \leftarrow A^n$" and "$not\, A^n \leftarrow A^p$" to $P^{no\neg}$.

Program $P^{no\neg}$ has no occurrence of explicitly negated literals.

Example 6. Let P be the program $\{c \leftarrow not\, \neg a; a \leftarrow not\, b; b \leftarrow not\, a; \neg a\}$. Its transformed program $P^{no\neg}$ is:

$$\begin{array}{llll} c^p \leftarrow not\, a^n. & a^p \leftarrow not\, b^p. & a^n. & b^p \leftarrow not\, a^p. \\ not\, c^p \leftarrow c^n. & not\, a^p \leftarrow a^n. & & not\, b^p \leftarrow b^n. \\ not\, c^n \leftarrow c^p. & not\, a^n \leftarrow a^p. & & not\, b^n \leftarrow b^p. \end{array}$$

Theorem 14. *Let P be an extended logic program. Then A (resp. $\neg A$, $not\, A$, $not\, \neg A$) belongs to $WFM_p(P)$ iff A^p (resp. A^n, $not\, A^p$, $not\, A^n$) belongs to $WFM_{\mathrm{p}}^{\mathrm{g}}(P^{no\neg})$.*

Example 7. Consider programs P and $P^{no\neg}$ of the previous example. Then, we have $WFM_{\rm p}\,(P) = \{\neg a, b\} \cup not\,\{a, \neg b, c, \neg c\}$ and $WFM^{\rm g}_{\rm p}\,(P^{no\neg})$ is:

$$\{a^n, b^p\} \cup not\,\{a^p, b^n, c^p, c^n\} \cup not\,\{\neg a^p, \neg a^n, \neg b^p, \neg b^n, \neg c^p, \neg c^n\}$$

The reader can check that the relation stated in theorem 14 is satisfied. Notice the extra $\neg$-literals in $WFM^{\rm g}_{\rm p}\,(P^{no\neg})$, which do not have any counterpart in $WFM_{\rm p}\,(P)$. This is so because we have defined *WFSX*$^{\rm g}_{\rm p}$ for general logic programs, not for the restricted form of programs used, i.e. with no explicitly negated literals.

One might argue that, given the above result, we do not need extended logic programs. We do not agree because it is much more natural and clearer to express knowledge with extended logic programming. Furthermore, it can be shown that *WFSX*$^{\rm g}_{\rm p}$ does not obey the fundamental principle of partial evaluation [8], but *WFSX*$_{\rm p}$ does. According to our view, a program with default negated literals in rule heads should first be transformed into an extended logic program under *WFSX*$_{\rm p}$ semantics.

4 Jonker's *IMEX* negation

In [16] a new form of negation, "$\sim$", is introduced and named "imex negation", standing for "<u>im</u>plicit and <u>ex</u>plicit negation." The rationale is that "$\sim A$" should be true "whenever there is an explicit reason for A to be false or because there is no argument for the truth of A." The author provides an isomorphism between the semantics of imex logic programs and extended ones, which increases the size of the original imex program, this being used as an argument in favour of imex logic programs. We will show that the semantics of an imex logic program P can be obtained from the semantics of an *isomorphic* extended logic program P^{imex}, and that imex logic programming is simply a syntactic restriction of extended logic programs.

We will show that the well-founded model of an imex logic program can be obtained too via an equivalent and isomorphic general logic program under *WFSX*$^{\rm g}_{\rm p}$ semantics, where the imex negation symbol "$\sim$" is substituted everywhere by $not\,A$. It seems to us that this characterization is much simpler than the original one by Jonker. Furthermore, we contemplate the full generality of the syntax of general logic programs by means of a paraconsistent semantics; this is not addressed in [16]. We begin by presenting a few definitions.

Definition 15 IMEX. [16] An imex logic program is a logic program that consists of rules of the form $X_0 \leftarrow X_1, \ldots, X_n$ where for $0 \leq i \leq n$, each imex literal X_i being either $\sim A$ or A, for some atom A. IMEX is the class of all imex logic programs.

Jonker defines a notion of well-founded model for imex programs. The intuition and the desired results are illustrated in the following example:

Example 8. [16] Consider the following two imex logic programs:

$$P_1 = \{d \leftarrow\sim a; \quad b \leftarrow\sim c; \quad c \leftarrow\sim b, \sim d; \quad \sim a \leftarrow b\}$$
$$P_2 = \{d \leftarrow\sim a; \quad b \leftarrow\sim c, \sim d; \quad c \leftarrow\sim b; \quad \sim a \leftarrow b\}$$

The imex well-founded models of the above programs are $\{\sim a, b, \sim c, d\}$ and $\{\sim a, \sim b, c, d\}$, respectively. In both cases $\sim a$ is true because of an implicit reason, namely there is no argument for thre truth of a. However, in the first program, there is also an explicit reason for a to be false, namely via rule $\sim a \leftarrow b$, whereas in P_2 there is no such explicit reason because $\sim b$ is true.

The watchful reader will have noticed that if one substitutes the imex negation symbol in the heads by explicit negation, and in the bodies by default negation one will get a paraconsistent well-founded model which is isomorphic to Jonker's well-founded model: we simply ignore away literals of the form $\neg A$ and $not\,\neg A$ in our model. This is easily brought out with the imex program $\{a \leftarrow\sim b; b \leftarrow\sim a; \sim b\}$ whose well-founded model is $\{a, \sim b\}$.

Jonker provides in her thesis the following program transformation from imex logic programs to extended logic programs.

Definition 16. [16] Let P be an imex logic program. Then the operator Υ : $IMEX \rightarrow ELP$ is defined as follows:

1. Substitute any occurence of $\sim A$ in bodies of rules in P by $not\,A$ in $\Upsilon(P)$;
2. Substitute any occurence of $\sim A$ in the heads of rules in P by $\neg A$ in $\Upsilon(P)$;
3. For every atom A in the language of P add the rule $\neg A \leftarrow not\,A$ to $\Upsilon(P)$.

Theorem 17. [16] *Let P be an imex logic program. Then P is semantically equivalent to $\Upsilon(P)$ under* WFSX[2]*: $A \in WF(P)$ iff $A \in WFM(\Upsilon(P))$ and $\sim A \in WF(P)$ iff $\neg A \in WFM(\Upsilon(P))$.*

Example 9. Consider program P_1 of example 8. The program $\Upsilon(P)$ is

$$\begin{array}{llll} d \leftarrow not\,a. & b \leftarrow not\,c. & c \leftarrow not\,b, not\,d. & \neg a \leftarrow b. \\ \neg d \leftarrow not\,d. & \neg b \leftarrow not\,b. & \neg c \leftarrow not\,c. & \neg a \leftarrow not\,a. \end{array}$$

Whose well-founded model is $\{\neg a, b, \neg c, d\} \cup not\,\{a, \neg b, c, \neg d\}$.

By removing step 3 in definition 16 we still guarantee an isomorphism between imex semantics and *WFSX* semantics. The effect of the rules of the form $\neg A \leftarrow not\,A$ is to turn $\neg A$ equivalent to $not\,A$. But, by the Υ transformation, there is no explicitly negated literal in the body of a rule; therefore if $not\,A$ is true then it is irrelevant to make $\neg A$ true. We only have to keep the other implication, i.e. if $\neg A$ is true then $not\,A$ should be true. This is guaranteed by coherence.

Theorem 18. *Let P be an imex logic program. Construct extended logic program P^{imex} as follows:*

[2] The imex semantics is not defined for the paraconsistent case.

1. *Substitute any occurence of $\sim A$ in bodies of rules in P by $not\, A$ in $\Upsilon(P)$;*
2. *Substitute any occurence of $\sim A$ in the heads of rules in P by $\neg A$ in $\Upsilon(P)$;*

Then P is semantically equivalent to $\Upsilon(P)$ under WFSX*: $A \in WF(P)$ iff $A \in WFM(P^{imex})$ and $\sim A \in WF(P)$ iff $not\, A \in WFM(P^{imex})$.*

Example 10. Consider once more program P_1 of example 8. Its corresponding extended logic program P^{imex} is $\{d \leftarrow not\, a; b \leftarrow not\, c; c \leftarrow not\, b, not\, d; \neg a \leftarrow b\}$. The well-founded model of P^{imex} is $\{\neg a, b, d\} \cup not\, \{a, \neg b, c, \neg c, \neg d\}$. Discarding all literals with occurences of explicit negation we obtain $\{b, d, not\, a, not\, c\}$, which corresponds exactly to the well-founded model of the original imex program.

This shows that imex logic programs have isomorphic extended logic programs with isomorphic models, rebutting the arguments in [16] for the use of imex logic programs.

We end this section by showing that imex logic programs can alternatively be seen as general logic programs without occurences of explicitly negated literals:

Theorem 19. *Let P be an imex logic program. General logic program P^{ix} is obtained from P by substituting every occurence of $\sim A$ by $not\, A$. Then P is semantically equivalent to P^{ix} under* $\mathrm{WFSX}_{\mathrm{p}}^{\mathrm{g}}$*: $A \in WF(P)$ iff $A \in WFM_{\mathrm{p}}^{\mathrm{g}}\left(P^{ix}\right)$ and $\sim A \in WF(P)$ iff $not\, A \in WFM_{\mathrm{p}}^{\mathrm{g}}\left(P^{ix}\right)$.*

Example 11. General logic program P_1^{ix} corresponding to program P_1 of example 8 is $\{d \leftarrow not\, a; b \leftarrow not\, c; c \leftarrow not\, b, not\, d; not\, a \leftarrow b.\}$ Its generalized well-founded model is $WFM_{\mathrm{p}}^{\mathrm{g}}\left(P_1^{ix}\right) = \{b, d\} \cup not\, \{a, c, \neg a, \neg b, \neg c, \neg d\}$ which corresponds, according to the above theorem, to the well-founded model of the original imex logic program.

The main conclusion of the comparison of extended logic programs with imex logic programming is that the latter is just syntactic sugar, based on rather cumbersome definitions. Another argument against imex logic programming is that it fails to obey the Generalized Principle of Partial Evaluation.

5 Lifschitz & Woo's Generalized Answer Sets

One of the most influential semantics for logic programming, besides well-founded semantics, has been Stable Models [12] and its successor extensions [13, 14, 18, 15]. In [17] a very general and unifying autoepistemic-based framework is provided for answer sets semantics and its variations, including disjunction, and discussion of the first-order case. [18] restricts this framework to what the authors call "PL-theories" (protected literal theories), which have a natural correspondence to disjunctive extended logic programs, and allowing positive occurrences of default negated literals. This provides a complete symmetry in the logic programming language. We refer the reader to the literature for the required definitions.

A general logic program may have none, one, or several answer sets.

Example 12. Consider the following three general logic programs:

$$P_1 = \{a \leftarrow c; c \leftarrow not\, b; not\, a \leftarrow not\, b\} \qquad P_2 = \{\neg b \leftarrow not\, a; not\, a \leftarrow \neg b\}$$
$$P_3 = \{a \leftarrow not\, b; c \leftarrow not\, d; b \leftarrow not\, a; d \leftarrow not\, c; not\, c \leftarrow not\, a; not\, d \leftarrow a\}$$

The first program has no answer sets, since $not\, b$ should be true and therefore also c and a, thereby violating the last rule. Program P_2 has a single answer set, namely $\{\neg b\}$. The last program has two answer sets, $\{a, c\}$ and $\{b, d\}$. Notice how the two rules, with positive occurences of *not* , discard the other two possible ways of satisfying the four remaining rules, as if they were treated like integrity constraints.

We can easily relate generalized answer sets with the general paraconsistent partial stable models.

Theorem 20. *Let P be a general logic program; then S is a consistent answer set of P iff $S \cup not\, (\mathcal{H}_P - S)$ is a generalized PSM_p of P.*

This shows, as would be expected, that all consistent answer sets of a general logic program may be captured by $WFSX_{\rm p}^{\rm g}$ models. A corollary of the above theorem is that S is a consistent (generalized) answer set of P iff S is the restriction to the language of P of a consistent answer set of P^{not}. Thus, the P^{not} program transformation is not entirely $WFSX_{\rm p}$ semantics specific. Remark that the transformation given in [15] makes use of integrity constraints to define the meaning of general logic programs, with the result that the transformation is not generally applicable as ours is and, more importantly, that it cannot handle default inconsistencies, i.e. simultaneous truth of a and $not\, a$.

The generalized well-founded model of a general logic program partially captures the semantics given by the intersection of all generalized answer-sets, when they exist. That is, $WFSX_{\rm p}^{\rm g}$ is a sceptical version of the generalized answer sets semantics. It is still open to much debate which of the base semantics, *WFS* or *SM*, should be preferred. However, semantics based on *WFS* seem to be more prone to simple implementation than stable model based ones. Furthermore, our extension obeys all of Dix's strong principles. It is well-known [9] that Stable Models, and thus Answer Sets, do not obey cumulativity, rationality, and relevance. In addition, $WFSX_{\rm p}^{\rm g}$ has a non-explosive behaviour when faced with inconsistent programs. The same does not happen with generalized answer sets.

6 Przymusinski's Static Expansions

One of the main problems in recent years has been the extending of the semantics of logic programs to disjunctive ones. One approach has been expplored in [22]. The idea is that a disjunctive logic programming should be translated into an implication in an appropriate autoepistemic logic. This is necessary to capture the meaning of the negation by default operator. To give this autoepistemic interpretation to disjunctive logic programs, under generalizations of *WFS*, Przymusinski defined the Static Semantics [22] and the *Autoepistemic Logic of Beliefs*, *AEB*. The language of *AEB*, is a propositional modal language, $\mathcal{K}_{\mathcal{B}}$, with the standard connectives, propositional letter $\perp$ (denoting *false*) and a modal operator $\mathcal{B}$, called the *belief* operator. The atomic formulae of the form $\mathcal{B}F$, where F is an arbitrary formula of $\mathcal{K}_{\mathcal{B}}$, are called *belief atoms.* One translates logic program rules into belief theories by substituting $not\, C$ by $\mathcal{B}\neg C$ and the rule implication sign by classical implication. The corresponding belief theory has a unique least static expansion which corresponds to the well-founded model of the original program [22]. Furthermore, all static expansions are partial stable (or stationary) models, and vice-versa.

We will restrict our comparison to general logic programs without occurrences of explicitly negated literals, since the above framework does not consider them. In [22] "$\neg$" (strong negation) is used. However, the problems we will point out also carry over for this more general setting. Furthermore, interpreting the new negation symbol as strong negation has additional problems, discussed in [2, 3, 6]. The introduction of a second negation requires additional axioms in the theory. For such a framework, using "explicit" rather than "strong" negation and capturing *WFSX* see [3, 6]. An alternative comparison, via theorem 14, would be to find a correspondence between the general paraconsistent well-founded model and the least static expansion of a related belief theory. Then we would implicitly define a translation from extended logic programs into belief theories.

To proceed with the comparisons with [22], we first extend the transformation of normal logic programs into belief theories to consider positive occurrences of *not* s. This is achieved by replacing a rule of the form

$$not\, A \leftarrow B_1, \ldots, B_m, not\, C_1, \ldots, not\, C_n \text{ by } B_1, \ldots, B_m, \mathcal{B}\neg C_1, \ldots, \mathcal{B}\neg C_n \supset \mathcal{B}\neg A$$

As seen previously (section 3) this translation does not provide a one-to-one correspondence between our generalized paraconsistent well-founded model and the least static expanion. We will now detail the reasons why this is so. Let us first analyze the "problematic" example again:

Example 13. Consider the general logic program P_4 of example 3. The corresponding belief theory T is:

$$T = \{b \supset a; \qquad c \supset b; \qquad \mathcal{B}\neg c \supset c; \qquad \mathcal{B}\neg b\}$$

According to our semantics' intuitions we expect this theory to derive $\mathcal{B}\neg b$ and $\mathcal{B}\neg a$ in its least static expansion We conclude $\mathcal{B}\neg b$ from the above belief theory, i.e. $not\, b$ in the original logic program. The other literals are undefined, i.e. neither $\mathcal{B}a$ nor $\mathcal{B}\neg a$ are entailed, and the same holds for c. Note that in the $WFM^{\mathrm{g}}_{\mathrm{p}}(P_4)$ we derive, additionally, $not\, a$.

The example shows that *AEB* is not capable of propagating the falsity of literals overriding undefined atoms, i.e. the undefinedness assigned to b in the basis of rule $b \leftarrow c$ (since c is undefined) is not overridden by the fact $\mathcal{B}\neg b$.

This seems to us undesirable. When the propagation of believed false atoms is not necessary both approaches seem to coincide, extrapolating from [2, 3, 6]. This is a claim that we leave without proof. A more problematic issue regards introducing paraconsistency in the above belief theories. We think it is not a trivial task, and certainly the underlying logic shouldn't be classical logic, since the propagation of believed false atoms should be taken into account.

Example 14. Consider the following two general logic programs:

$$P_1 = \{b \leftarrow a; \quad a; \quad not\, a\} \qquad\qquad P_2 = \{b; \quad b \leftarrow a; \quad a; \quad not\, a\}$$

According to our proposed translation the corresponding belief theories are

$$T_1 = \{a \supset b; \quad a; \quad \mathcal{B}\neg a\} \qquad\qquad T_2 = \{b; \quad a \supset b; \quad a; \quad \mathcal{B}\neg a\}$$

The first problem is that by Przymusinski's axiom (D) we must believe simultaneously in $\{\mathcal{B}a, \mathcal{B}\neg a, \neg\mathcal{B}a, \neg\mathcal{B}\neg a\}$, since $\mathcal{B}\neg a$ belongs to the theory and also $\mathcal{B}a$ by application of the necessitation rule to the true formula a. The negations of the two previous belief atoms follow by (D). Therefore, we need a paraconsistent modal logic to provide meaning to these programs. Furthermore, the propagation of the falsity of a should entail also $\mathcal{B}\neg b$ in the first program, to coexist with b and $\mathcal{B}b$. This effect is not desirable in the second example, since b is true and so we want to disregard the effect of the rule $b \leftarrow a$ because we have a rule for b with true body. This effect is indeed obtained with $WFSX_{\mathrm{p}}^{\mathrm{g}}$, with program P_2 having model $\{b, not\,\neg b, a, not\, a, not\,\neg a\}$. Notice this difference of behaviour in programs P_1 and P_2 is even more dramatic for a classical negation based logic, since the above two theories can be easily argued to be logically equivalent: if a is true then b is true by *modus ponens* and we can substitute the implication by $\{a, b\}$.

We conclude by agreeing that AEB is an interesting framework for giving an autoepistemic characterization to disjunctive logic programs, and it provides an extension by letting belief atoms appear in the heads of rules. The way these belief atoms are treated is problematic for the purposes of defining an alternative paraconsistent autoepistemic logic like definition of $WFSX_{\mathrm{p}}$ and $WFSX_{\mathrm{p}}^{\mathrm{g}}$. Moreover, the results do not extend easily for the paraconsistent case. However we believe that the problem can be fixed by means of the Belief Completion technique [5], of adding additional rules for the purpose of propagating atoms believed false.

7 Conclusions

In this paper we have shown that our paraconsistent version of well-founded semantics with explicit negation is powerful enough to define an intuitive and motivated interpretation of program rules with default negated literals in the heads, by means of a transformation to programs without them. Even though the semantics is given by a particular program transformation, the ideas are semantical ones. Another argument supporting this claim is the existence of alternative program transformations resulting in the same $WFSX_{\mathrm{p}}^{\mathrm{g}}$ semantics. One such transformation is readily obtainable from $P^{no\neg}$ and P^{ix} of Definition 13, and Theorem 19, which cannot be presented here for lack of space.

We've related our default negation in the heads transformational semantics with Jonker's *IMEX* negation, showing that for consistent normal logic programs both approaches coincide, ours being of much simpler formulation and extending her framework to paraconsistent and extended logic programs.

We also showed that $WFSX_{\mathrm{p}}^{\mathrm{g}}$ is more sceptical than Generalized Answer Sets semantics, the latter being models of our programs under $WFSX_{\mathrm{p}}^{\mathrm{g}}$. For the case of consistent generalized answer sets, they are in one-to-one correspondence with the answer sets of the transformed program P^{not}, thus lending support the generality of our basic principles.

Regarding the relation of Static Expansions with $WFSX_{\mathrm{p}}^{\mathrm{g}}$ we showed the latter, but not the former, is capable of propagating falsities of belief atoms. Also, the generalization of Static Semantics to the paraconsistent case is problematic.

In our view, $WFSX_{\mathrm{p}}^{\mathrm{g}}$, because of the structural properties it enjoys, is the more preferable encompassing semantics for non–disjunctive general logic programs. Furthermore, no other semantics is defined for the whole class of general logic programs. Its extension to disjunctive logic programs will be the subject of future work.

Acknowledgments

We thank JNICT - Portugal and the PRAXIS PROLOPPE project for their support. We thank José Alferes for helpful discussions and the anonymous referees for their comments.

References

1. J. J. Alferes, C. V. Damásio, and L. M. Pereira. A logic programming system for non-monotonic reasoning. *Journal of Automated Reasoning*, 14(1):93–147, 1995.
2. J. J. Alferes and L. M. Pereira. On logic program semantics with two kinds of negation. In K. Apt, editor, *Joint International Conference and Symposium on Logic Programming*, pages 574–588, Washington, USA, November 1992. MIT Press.
3. J. J. Alferes and L. M. Pereira. Belief, provability and logic programs. *Journal of Applied Non-Classical Logics. Logics in AI. A selection of extended versions of JELIA'94 papers*, 5(1), 1995.
4. J. J. Alferes and L. M. Pereira. *Reasoning with Logic Programming.* Springer–Verlag, 1996. In print.
5. J. J. Alferes, L. M. Pereira, and T. C. Przymusinski. Belief revision in non-monotonic reasoning. In C. Pinto-Ferreira and N. J. Mamede, editors, *Proc. EPIA'95, LNAI 990*, pages 41–56, 1995.
6. J. J. Alferes, L. M. Pereira, and T. C. Przymusinski. "classical" negation in non-monotonic reasoning and logic programming. In *AI & Math'96*, 1996. To appear. Extended version available as technical report.

7. J. Dix. A Classification-Theory of Semantics of Normal Logic Programs: I. Strong Properties. *Fundamenta Informaticae*, XXII(3):227–255, 1995.
8. J. Dix. A Classification-Theory of Semantics of Normal Logic Programs: II. Weak Properties. *Fundamenta Informaticae*, XXII(3):257–288, 1995.
9. J. Dix. Semantics of Logic Programs: Their Intuitions and Formal Properties. An Overview. In A. Fuhrmann and H. Rott, editors, *Logic, Action and Information. Proceedings of the Konstanz Colloquium in Logic and Information (LogIn '92)*, pages 241–329. DeGruyter, 1995.
10. M. V. Emden and R. Kowalski. The semantics of predicate logic as a programming language. *Journal of ACM*, 4(23):733–742, 1976.
11. A. V. Gelder, K. A. Ross, and J. S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, 1991.
12. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R. Kowalski and K. A. Bowen, editors, *ICLP'88*, pages 1070–1080, 1988.
13. M. Gelfond and V. Lifschitz. Logic programs with classical negation. In Warren and Szeredi, editors, *ICLP'90*, pages 579–597. MIT Press, 1990.
14. M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385, 1991.
15. K. Inoue and C. Sakama. On positive occurrences of negation as failure. In J. Doyle, E. Sandewall, and P. Torasso, editors, *Proc. KR'94*, pages 293–304. Morgan-Kaufmann, 1994.
16. C. Jonker. *Constraints and Negations in Logic Programming*. PhD thesis, Department of Philosophy - Utrecht University, September 1994.
17. V. Lifschitz. Minimal belief and negation as failure. Technical report, Department of Computer Science and Department of Philisophy, University of Texas at Austin, 1992.
18. V. Lifschitz and T. C. Woo. Answer sets in general nonmonotonic reasoning (preliminary report). In B. Nebel, C. Rich, and W. Swartout, editors, *Principles of Knowledge Representation and Reasoninng: Proceedinsg of the Third International Conference (KR'92)*, pages 603–614. Morgan-Kaufmann, 1992.
19. D. Pearce and G. Wagner. Reasoning with negative information I: Strong negation in logic programs. In L. Haaparanta, M. Kusch, and I. Niiniluoto, editors, *Language, Knowledge and Intentionality*, pages 430–453. Acta Philosophica Fennica 49, 1990.
20. L. M. Pereira and J. J. Alferes. Well founded semantics for logic programs with explicit negation. In B. Neumann, editor, *European Conference on Artificial Intelligence*, pages 102–106, Wien, Austria, August 1992. John Wiley & Sons.
21. T. Przymusinski. Extended stable semantics for normal and disjunctive programs. In Warren and Szeredi, editors, *7th International Conference on Logic Programming*, pages 459–477. MIT Press, 1990.
22. T. C. Przymusinski. Static semantics for normal and disjunctive logic programs. *Annals of Mathematics and Artificial Intelligence*, 14:323–357, 1995.
23. G. Wagner. A database needs two kinds of negation. In B. Thalheim, J. Demetrovics, and H.-D. Gerhardt, editors, *Mathematical Foundations of Database Systems*, pages 357–371. LNCS 495, Springer–Verlag, 1991.
24. G. Wagner. Vivid logic: Knowledge-based reasoning with two kinds of negation. *Lecture Notes on Artificial Intelligence*, 764, 1994.

This article was processed using the LaTeX macro package with LLNCS style