Contradiction Removal Semantics with Explicit Negation

Luís Moniz Pereira, José J. Alferes, Joaquim N. Aparício

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

Abstract. Well Founded Semantics for logic programs extended with eXplicit negation ($WFSX$) is characterized by that, in any model, whenever $\neg a$ (the explicit negation of a) holds, then $\sim a$ (the negation by default of a) also holds.
When explicit negation is used contradiction may be present (e.g. a and $\neg a$ both hold for some a) and thus no semantics is given to the program. We introduce here the notion of removing some contradictions, through identifying the set of models obtained by revising closed world assumptions. One such unique model is singled out as the contradiction free semantics ($CRSX$). When contradiction does not arise, the contradiction removal semantics coincides with $WFSX$.

1 Introduction

Recently, several authors have stressed and showed the importance of having an explicit second kind of negation within logic programs, for use in deductive databases, knowledge representation, and nonmonotonic reasoning [2, 4, 5, 7, 6, 14, 15, 16, 23, 21].

Some proposals for extending logic programming semantics with a second kind of negation has been advanced. One such extension is the Answer Set semantics (AS) [4], which is shown to be an extension of Stable Model (SM) semantics [3] from the class of logic programs [8] to those with a second form of negation. In [7] another proposal for such extension is introduced, based on the SM semantics, where implicitly a preference between negative information (exceptions) over positive information is assumed. However AS semantics is not well founded. The meaning of the program is defined as the intersection of all answer sets and it is known that the computation of this intersection is computationally expensive. Another extension to include a second kind of negation is suggested by Przymusinski in [20]. Although the set of models identified by this extension enjoys the well founded property, it gives some less intuitive results [1] with respect to the coexistence of both forms of negation. Based on the XSM semantics, Przymusinski [21] also introduces the Stationary semantics where the second form of negation is classical negation. But classical negation also entails that the logic programs under Stationary semantics no longer admit a procedural reading.

On the other hand, $WFSX$[1] (Well Founded Semantics with eXplicit negation) [9], which we prefer, is an extension to Well Founded Semantics [22] including a second form of negation called *explicit negation*, preserving the well founded property (cf.[1] for a comparison of the above approaches) and procedural reading. Furthermore, *explicit negation* is characterized by that, in any model, whatever the classical literal l, whenever $\neg l$ holds $\sim l$, i.e. the negation by default or implicit negation of l also holds, and l is false, thus avoiding the less intuitive results concerning the relation between the two forms of negation (cf. [1] for other approaches).

When a second form of negation is introduced contradiction may be present (i.e. l and $\neg l$ hold for some l) and no semantics is given by $WFSX$. We define here the $CRSX$ semantics, extending $WFSX$ by introducing the notion of removing some contradictions and identifying the models obtained by revising closed world assumptions supporting those contradictions. One unique model, if any such revised model exists, is singled out as the contradiction free semantics. When no contradiction is present $CRSX$ semantics reduces to $WFSX$ semantics. This work is an improvement and above all a generalization of [10] to programs with a second kind of non–pseudo negation. It captures a wide variety of nonmonotonic reasoning forms in logic programming, as exhibited elsewhere [17].

Furthermore, under $WFSX$ programs admit a procedural logic programming reading, which is not the case if truly classical negation plus material implication are used, as in [21], where case analysis is condoned. Under $WFSX$ rules in the program are unidirectional (contrapositives are not implicit), maintaining the procedural flavour; the rule connective, $\leftarrow$, is not material implication, but is rather like an inference rule.

The paper is organized as follows: first we briefly review the $WFSX$ semantics introduced in [9] and present some examples when $WFSX$ semantics is used. Since some programs may have no semantics, next we introduce the $CRSX$, a process of identifying negative literals which are true by the CWA inference rule, as sources of contradiction, and show how contradiction may be removed. For each way of removing contradiction in program P we construct a program P' such that $WFSX(P')$ is consistent. We then present some properties concerning the $CRSX$ semantics defined.

2 Language

Given a first order language $Lang$ [18], an extended logic program is a set of rules of the form

$$H \leftarrow B_1, \ldots, B_n, \sim C_1, \ldots, \sim C_m \qquad m \geq 0, n \geq 0$$

where $H, B_1, \ldots, B_n, C_1, \ldots, C_m$ are classical literals. A (syntactically) classical literal (or explicit literal) is either an atom A or its explicit negation $\neg A$. We also use the symbol $\neg$ to denote complementary literals in the sense of explicit

[1] In [12, 13] it is shown how $WFSX$ relates to default theory.

negation. Thus $\neg\neg A = A$. The symbol $\sim$ stands for negation by default[2]. $\sim L$ is called a default literal. Literals are either classical or default. A set of rules stands for all its ground instances w.r.t. *Lang*. When $n = m = 0$ we may simply write H instead of $H \leftarrow$.

As in [18], we expand our language by adding to it the proposition $\mathbf{u}$ such that for every interpretation I, $I(\mathbf{u}) = 1/2$. By a non-negative program we mean a program whose premises are either classical literals or $\mathbf{u}$. Given a program P we denote by $\mathcal{H}_P$ (or simply $\mathcal{H}$) its Herbrand base. If S is a set of literals $\{L_1, \ldots, L_n\}$, by $\sim S$ we mean the set $\{\sim L | L \in S\}$.

If S is a set of literals then we say S is *contradictory* (resp. *inconsistent*) iff there is classical literal L such that $\{L, \neg L\} \subseteq S$ (resp. $\{L, \sim L\} \subseteq S$). In this case we also say that S *is contradictory w.r.t. to* L, (resp. S *is inconsistent w.r.t. to* L). S is agnostic w.r.t. L iff neither $L \in S$ nor $\sim L \in S$.

3 *WFSX* overview

In this section we briefly review $WFSX$ semantics for logic programs extended with explicit negation. For full details the reader is referred to [9].

$WFSX$ follows from one basic "coherence" requirement:

$$\neg L \Rightarrow \sim L \tag{1}$$

i.e. (if L is explicitly false, L must be false) for any explicit literal L.

Example 1. Consider program $P = \{a \leftarrow \sim b, \quad b \leftarrow \sim a, \quad \neg a \leftarrow\}$.

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 [19]), the result would be $\{\neg a, \sim\neg b\}$, so that $\neg a$ is true and a is undefined. We insist that $\sim a$ should hold, and a not, because $\neg a$ does. Accordingly, the WFSX of P is $\{\neg a, b, \sim a, \sim\neg b\}$, since b follows from $\sim a$.

We begin by providing a definition of interpretation for programs with explicit negation which incorporates coherence from the start.

Definition 1 Interpretation. By an interpretation I of a language *Lang* we mean any set $T \cup \sim F$[3], where T and F are disjoint subsets of classical literals over the Herbrand base, and if $\neg L \in T$ then $L \in F$ (coherence)[4]. The set T contains all ground classical literals *true* in I, the set F contains all ground classical literals *false* in I. The truth value of the remaining classical literals is *undefined* (The truth value of a default literal $\sim L$ is the 3-valued complement of L.)

[2] This designation has been used in the literature instead of the more operational *"negation as failure (to prove)"*. Another appropriate designation is *"implicit negation"*, in contradistinction to *"explicit negation"*.

[3] By $\sim\{a_1, \ldots, a_n\}$ we mean $\{\sim a_1, \ldots, \sim a_n\}$.

[4] For any literal L, if L is explicitly false L must be false. Note that the complementary condition "if $L \in T$ then $\neg L \in F$" is implicit.

We next extend with an additional rule the P modulo I transformation of [18], itself an extension of the Gelfond-Lifschitz modulo transformation, to account for coherence.

Definition 2 P/I **transformation.** Let P be an extended logic program and let I be an interpretation. By P/I we mean a program obtained from P by performing the following three operations for every atom A :

- Remove from P all rules containing a default premise $L = \sim A$ such that $A \in I$.
- Remove from P all rules containing a non–default premise L (resp. $\neg L$) such that $\neg L \in I$ (resp. $L \in I$).
- Remove from all remaining rules of P their default premises $L = \sim A$ such that $\sim A \in I$.
- Replace all the remaining default premises by proposition $\mathbf{u}$[5].

The resulting program P/I is by definition non-negative, and it always has a unique $least(P/I)$, where $least(P/I)$ is:

Definition 3 Least-operator. We define $least(P)$, where P is a non-negative program, as the set of literals $T \cup \sim F$ obtained as follows:

- Let P' be the non-negative program obtained by replacing in P every negative classical literal $\neg L$ by a new atomic symbol, say $'\neg\_L'$.
- Let $T' \cup \sim F'$ be the least 3-valued model of P'.
- $T \cup \sim F$ is obtained from $T' \cup \sim F'$ by reversing the replacements above.

The least 3-valued model of a non-negative program can be defined as the least fixpoint of the following generalization of the van Emden–Kowalski least model operator Ψ for definite logic programs:

Definition 4 Ψ^* **operator.** Suppose that P is a non-negative program, I is an interpretation of P and A is a ground atom. Then $\Psi^*(I)$ is an interpretation defined as follows:

- $\Psi^*(I)(A) = 1$ iff there is a rule $A \leftarrow A_1, \ldots, A_n$ in P such that $I(A_i) = 1$ for all $i \leq n$.
- $\Psi^*(I)(A) = 0$ iff for every rule $A \leftarrow A_1, \ldots, A_n$ there is an $i \leq n$ such that $I(A_i) = 0$.
- $\Psi^*(I)(A) = 1/2$, otherwise.

To avoid incoherence, a partial operator is defined that transforms any non-contradictory set of literals into an interpretation, whenever contradiction[6] is not present.

Definition 5 The Coh **operator.** Let $I = T \cup \sim F$ be a set of literals such that T is not contradictory. We define $Coh(I) = I \cup \sim \{\neg L \mid L \in T\}$.

[5] The special proposition $\mathbf{u}$ is *undefined* in all interpretations.

[6] We say a set of literals S is contradictory iff for some literal L, $L \in S$ and $\neg L \in S$.

Definition 6 The Φ operator. Let P be a logic program and I an interpretation, and let $J = least(P/I)$. If $Coh(J)$ exists we define $\Phi_P(I) = Coh(J)$. Otherwise $\Phi_P(I)$ is not defined.

Example 2. For the program of example 1 we have

$$P/\{\neg a, b, \sim a, \sim\neg b\} = \{b \leftarrow; \neg a \leftarrow\},$$

$$least(P/\{\neg a, b, \sim a, \sim\neg b\}) = \{\neg a, b, \sim a\}$$

and:

$$Coh(\{\neg a, b, \sim a\}) = \{\neg a, b, \sim a, \sim\neg b\}.$$

Definition 7 WFS with explicit negation. An interpretation I of an extended logic program P is called an Extended Stable Model (XSM) of P iff $\Phi_P(I) = I$. The F-least Extended Stable Model is called the Well Founded Model. The semantics of P is determined by the set of all XSMs of P.

Example 3. Let $P = \{a \leftarrow \sim b, \sim c; b \leftarrow \sim a; \neg c \leftarrow \sim d\}$. This program has a least model $M_1 = \{\sim d, \neg c, \sim c, \sim\neg a, \sim\neg b, \sim\neg d\}$ and two Extended Stabel Models $M_2 = M_1 \cup \{\sim a, b\}$ and $M_3 = M_1 \cup \{a, \sim b\}$. Considering model M_1 we have for $P/M_1 = \{a \leftarrow \mathbf{u}; b \leftarrow \mathbf{u}; c' \leftarrow\}$, and

$$least(P/M_1) = J = \{\sim d, c', \sim a', \sim b', \sim d', \sim c\} = \{\sim d, \sim\neg d, \neg c, \sim c, \sim\neg a, \sim\neg b\}.$$

Example 4. Let P be:

$$\begin{array}{rl} \mathrm{a} \leftarrow \sim a & (i) \\ b \leftarrow \sim a & (ii) \\ \neg b \leftarrow & (iii) \end{array}$$

After the transformation, program P' has a rule $b' \leftarrow$, and there is no way in proving $\sim b$ from rules (i) and (ii). And we have $least(P'/\{b', \sim b, \sim a'\}) = \{b', \sim a'\} = M$ which corresponds to the model $\{\neg b, \sim\neg a\}$ if the coherence principle is not applied. In our case we have $Coh(M) = \{\neg b, \sim b, \sim\neg a\}$ which is the intended result.

Definition 8 Contradictory program. An extended logic program P is contradictory iff it has no semantics, i.e. there exists no interpretation I such that $\Phi_P(I) = I$.

4 Revising contradictory extended logic programs

Once we introduce explicit negation programs are liable to be contradictory:

Example 5. Consider program $P = \{a \leftarrow ; \neg a \leftarrow \sim b\}$. Since we have no rules for b, by CWA it is natural to accept $\sim b$ as true. By the second rule in P we have $\neg a$, leading to an inconsistency with the fact a. Thus no set containing $\sim b$ may be a model of P.

We argue that the CWA may not be held of atom b since it leads to a contradiction. We show below how to revise[7] this form of contradiction, by making a suitable revision of the incorrect CWA on b. The semantics we introduce identifies $\{a, \sim\neg a\}$ as the intended meaning of P, where b is revised to undefined. Assuming b false leads to a contradiction; revising it to true instead of undefined would not minimize the revised interpretation.

4.1 Contradictory Well Founded Model

In order to revise possible contradictions we need to identify those contradictory sets implied by applications CWA. The main idea is to compute all consequences of the program, even those leading to contradictions, as well as those arising from contradictions. The following example provides an intuitive preview of what we intend to capture:

Example 6. Consider program P :

$$\begin{array}{rlrl} a \leftarrow \sim b & \text{(i)} & d \leftarrow a & \text{(iii)} \\ \neg a \leftarrow \sim c & \text{(ii)} & e \leftarrow \neg a & \text{(iv)} \end{array}$$

1. $\sim b$ and $\sim c$ hold since there are no rules for either b or c
2. $\neg a$ and a hold from 1 and rules (i) and (ii)
3. $\sim a$ and $\sim\neg a$ hold from 2 and inference rule (1) (cf. page 3)
4. d and e hold from 2 and rules (iii) and (iv)
5. $\sim d$ and $\sim e$ hold from 3 and rules (iii) and (iv), as they are the only rules for d and e
6. $\sim\neg d$ and $\sim\neg e$ hold from 4 and inference rule (1) (cf. page 3).

The whole set of literals is then:

$$\{\sim b, \sim c, \neg a, a, \sim a, \sim\neg a, d, e, \sim d, \sim e, \sim\neg d, \sim\neg e\}.$$

N. B. We extend the language with the special symbol $\perp$. For every pair of classical literals $\{L, \neg L\}$ in the language of P we implicitly assume a rule $\perp \leftarrow L, \neg L$[8].

Definition 9 Pseudo–interpretation. A pseudo–interpretation (or p–interpretation for short) is a possibly contradictory set of ground literals from the language of a program.

We extend the Θ operator[18] from the class of interpretations to the class of p–interpretations, and we call this the Θ^x operator (x standing for eXtended).

[7] We treat contradictory programs extending the approach of [10, 11].

[8] This is not strictly necessary but simplifies the exposition. Furthermore, without loss of generality, we only consider rules $\perp \leftarrow L, \neg L$ for which rules for both L and $\neg L$ exist in P. We also use the notation $\perp_L$ to denote the head of rule $\perp \leftarrow L, \neg L$.

Definition 10 The Θ^x operator. Let P be a logic program and J a p–interpretation. The operator $\Theta^x_J : \mathcal{I} \rightarrow I$ on the set $\mathcal{I}$ of all 3-valued p–interpretations of P is defined as follows: If $I \in \mathcal{I}$ is a p–interpretation of P and A is a ground classical literal then $\Theta^x_J(I)$ is the p–interpretation defined by:

1. $\Theta^x_J(I)(A) = 1$ iff there is a rule $A \leftarrow L_1, \ldots, L_n$ in P such that for all $i \leq n$ either $\hat{J}(L_i) = 1$, or L_i is positive and $I(L_i) = 1$;
2. $\Theta^x_J(I)(A) = 0$ iff one of the following holds:
   (a) for every rule $A \leftarrow L_1, \ldots, L_n$ in P there is an $i \leq n$, such that either $\hat{J}(L_i) = 0$, or L_i is positive and $I(L_i) = 0$;
   (b) $\hat{J}(\neg A) = 1$;
3. $\Theta^x_J(I)(A) = 1/2$ otherwise.

Note that the only difference between this definition and the definition of Θ operator introduced in [18] is condition (2b) capturing the coherence requirement, or inference rule (1). Furthermore, since it is defined over the class of p–interpretations, it allows that for a given literal L, we may have $\Theta^x_J(I)(L) = 1$ as well as $\Theta^x_J(I)(L) = 0$.

Proposition 11. For every p–interpretation J, the operator Θ^x_J is monotone and has a unique least fixed point given by $\Theta^{x\uparrow w}_J$, also denoted by $\Omega^x(J)$[9].

Definition 12 p–model. Given a program, a p–model is a p–interpretation I such that:

$$I = \Omega^x(I) \tag{2}$$

Remark. Note that if a p–model M is contradictory w.r.t. to L then M is inconsistent w.r.t. to L by virtue of inference rule (1), although the converse is not true.

Definition 13 Well Founded Model. The pseudo Well Founded Model M_P of P is the F–least p–model.

The non–minimal models satisfying (2) above (pseudo) Extended Models (XMs for short). To compute the p–model M_P we define the following transfinite sequence $\{I_\alpha\}$ of fixed points:

$$\begin{aligned} I_0 &= \langle \emptyset, \emptyset \rangle \\ I_{\alpha+1} &= \Omega^x(I_\alpha) = \Theta^{x\ \uparrow w}_{I_\alpha} \\ I_\delta &= \textstyle\bigcup_{\alpha<\delta} I_\alpha \text{ for limit ordinal } \delta \end{aligned}$$

Equivalently, the pseudo well founded model M_P of P is the F–least fixed point of (2) and is given by $M_P = I_\lambda = \Omega^{x\uparrow\lambda}$.

Definition 14. A program P is contradictory iff $\bot \in M_P$.

---

[9] Recall [18] that the F-least interpretation used to compute the least fixed point of $\Theta^{x\uparrow w}_J$ is $\sim\mathcal{H}_P$.

Example 7. Recall the program of example 6:

$$P = \{a \leftarrow \sim b\ ; \neg a \leftarrow \sim c\ ; d \leftarrow a\ ; e \leftarrow \neg a\}.$$

$$\begin{aligned}
\Theta_{I_0}^{\uparrow 0} &= \{\sim a, \sim\neg a, \sim b, \sim\neg b, \sim c, \sim\neg c, \sim d, \sim\neg d, \sim e, \sim\neg e\} \\
\Theta_{I_0}^{\uparrow 3} = \Theta_{I_0}(\Theta_{I_0}^{\uparrow 2}) &= \{\sim b, \sim c, \sim\neg b, \sim\neg c, \sim\neg d, \sim\neg e\} = \Theta_{I_0}^{\uparrow w} = I_1 \\
\Theta_{I_1}^{\uparrow 3} = \Theta_{I_1}(\Theta_{I_1}^{\uparrow 2}) &= \{a, d, \neg a, e, \sim b, \sim\neg b, \sim c, \sim\neg c, \sim\neg d, \sim\neg e\} = \Theta_{I_1}^{\uparrow w} = I_2 \\
\Theta_{I_2}^{\uparrow 2} = \Theta_{I_2}(\Theta_{I_2}^{\uparrow 1}) &= \{a, \sim\neg a, \neg a, \sim a, d, \sim\neg d, e, \sim\neg e, \sim b, \sim\neg b, \sim c, \sim\neg c, \sim d, \sim\neg d, \sim e\} = \\
&= \Theta_{I_2}^{\uparrow w} = I_3 \\
\Theta_{I_3}^{\uparrow 2} = \Theta_{I_3}(\Theta_{I_3}^{\uparrow 1}) &= \{a, \sim\neg a, \neg a, \sim a, d, \sim\neg d, e, \sim\neg e, \sim b, \sim\neg b, \sim c, \sim\neg c, \sim d, \sim\neg d, \sim e\} = \\
&= \Theta_{I_3}^{\uparrow w} = I_4 = I_3
\end{aligned}$$

so the program is contradictory.

4.2 Removing the contradiction

In order to get revised non–contradictory consistent models we must know where contradiction arises from and prevent it. In this section we identify sets of default literals true by CWA whose revision to undefined can remove contradiction, by withdrawing the support of the CWAs on which the contradiction depends.

Definition 15 Dependency set. A Dependency Set of a literal L in a program P, represented as $DS(L)$, is obtained as follows:

1. If L is a classical literal:
   (a) if there are no rules for L then the only $DS(L) = \{L\}$.
   (b) for each rule $L \leftarrow B_1, \ldots, B_n (n \geq 0)$ in P for L, there exists one $DS_k(L) = \{L\} \cup \bigcup_i DS_{j(i)}(B_i)$ for each different combination k of one $j(i)$ for each i.
2. For a default literal $\sim L$:
   (a) if there are no rules in P for L then a $DS(\sim L) = \{\sim L\}$.
   (b) if there are rules for L then choose from every rule for L a single literal. For each such choice there exist several $DS(\sim L)$; each contains $\sim L$ and one dependency set of each default complement[10] of the chosen literals.
   (c) if there are rules for $\neg L$ then there are, **additionally**, dependency sets $DS(\sim L) = \{\sim L\} \cup DS_k(\neg L)$ for each k.

Example 8. $P = \{a \leftarrow \sim b\ ; \neg a \leftarrow \sim c\ ; d \leftarrow a\ ; e \leftarrow \neg a\}$. In this case we have the following dependency sets:

$$\begin{array}{llll}
DS(\sim b) & = \{\sim b\} & DS_1(\sim a) & = \{\sim a, b\} \\
DS(\sim c) & = \{\sim c\} & DS_2(\sim a) & = \{\sim a, \neg a, \sim c\} \\
DS(a) & = \{a, \sim b\} & & \\
DS(\neg a) & = \{\neg a, \sim c\} & & \\
DS_1(\sim d) & = \{\sim d\} \cup DS_1(\sim a) & = \{\sim d, \sim a, b\} & \\
DS_2(\sim d) & = \{\sim d\} \cup DS_2(\sim a) & = \{\sim d, \sim a, \neg a, \sim c\} & \\
DS(\perp_a) & = \{\perp_a, a, \neg a, \sim b, \sim c\} & &
\end{array}$$

[10] The default complement of a classical literal L is $\sim L$; that of a default literal $\sim L$ is L.

Definition 16 Support of a literal. A support $SS_M(L)$ w.r.t. to a model M is a non–empty dependency set $DS(L)$ such that $DS(L) \subseteq M$. If there exists a $SS_M(L)$ we say that L is supported in M.

For simplicity, a support w.r.t. the pseudo WFM M_P of P is represented by $SS(L)$.

Definition 17 Support of a set of literals. A support w.r.t. to a model M is:

$$SS_{M\,k}(\{L_1, \ldots, L_n\}) = \bigcup_i SS_{M\,j(i)}(L_i)$$

For each combination k of $j(i)$ there exists one support.

With the notion of support we are able to identify which literals support a contradiction, i.e. the literal $\perp$. In order to remove a contradiction we must change the truth value of at least one literal from each support set of $\perp$. One issue is for which literals we allow to initiate change of their truth values; another is how to specify a notion of minimal change.

As mentioned before, we only wish to initiate revision on default literals true by *CWA* in a manner made precise later. To identify such *revising* literals we first define:

Definition 18 Default supported. A default literal $\sim A$ is default supported w.r.t. M if **all** supports $SS_M(\sim A)$ have only default literals.

Example 9. Let $P = \{\neg a;\ a \leftarrow \sim b;\ b \leftarrow c;\ c \leftarrow d\}$. The only support of $\perp$ is $\{\neg a, a, \sim b, \sim c, \sim d\}$, and default supported literals are $\sim b$, $\sim c$, and $\sim d$. Here we are not interested in revising the contradiction by undefining $\sim b$ or $\sim c$ because they depend on $\sim d$. The reason is that we are attempting to remove only contradictions based on CWAs. Now, the CWA of a literal that is supported on another depends on the CWA of the latter.

In order to make precise what we mean we first present two definitions:

Definition 19 Self supported set. A set of default literals S is self supported w.r.t. a model M iff there exists a $SS_M(S) = S$.

Definition 20 Revising and co–Revising literals. Given a program P with pseudo well-founded model M_P, we define co–$\mathcal{R}_P$, the *co–revising literals* induced by P, as the set of literals belonging to some minimal self supported set w.r.t. M_P. We define $\mathcal{R}_P$, the *revising literals*, as the set of co–revising literals L such that $\neg L \notin M_P$. The next examples motivate these definitions.

Example 10. Let $P = \{\neg p;\quad p \leftarrow \sim a;\quad \neg a \leftarrow \sim b\}$.

The co-revising literals are $\{\sim a, \sim b\}$ and the revising are $\{\sim b\}$. The difference is that to revise $\sim a$ one needs to change the truth value of $\neg a$ as well, because of coherence. To revise $\sim b$ there is no such need. Revising $\neg a$ only is not enough since then $\sim a$ becomes true by default.

Example 11. In the program of example 9, the self supported sets w.r.t. M_P are $\{\sim b, \sim c, \sim d\}$, $\{\sim c, \sim d\}$, and $\{\sim d\}$. Thus the only revising literal is $\sim d$. Note how the requirement of minimality ensures that only CWA literals not depending on other CWAs are revising. In particular:

Proposition 21.

1. If there are no rules for L then $\sim L$ is a co–revising literal.
2. If there are no rules for L nor for $\neg L$ then $\sim L$ is a revising literal.
3. If $\sim L$ is co–revising and default supported then it is revising.

An atom can also be false by CWA if in a positive *"loop"*. Such cases are also accounted for:

Example 12. Let P_1 and P_2 be:

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

For P_1 self supported sets are: $\{\sim b, \sim c\}$, $\{\sim b\}$, and $\{\sim c\}$. Thus $\sim b$ and $\sim c$ are revising. For P_2 the only minimal self supported set is $\{\sim c\}$ thus only $\sim c$ is revising. The only support set of $\sim b$ is $\{\sim b, \sim c\}$. In P_2 it is clear that $\sim b$ depends on $\sim c$. So $\sim b$ is not revising. In P_1 the truth of $\sim b$ can support itself. Thus $\sim b$ is also revising.

Another class of literals is needed in the sequel. Informally, indissociable literals are those that strongly depend on each other, so that their truth value must always be the same. It is impossible to change the truth value of one without changing the truth value of another. So:

Definition 22 Indissociable set of literals. A set of default literals S is indissociable iff

$$\forall \sim a, \sim b \in S \quad \sim a \in \bigcap_i SS_i(\sim b) \ .$$

i.e. each literal in S belong to every support of every literal in S.

Proposition 23. If S is a minimal self supported set and the only $SS(S)$ is S, then S is an indissociable set of literals.

Example 13. In P below, $\{\sim a, \sim b, \sim c\}$ is a set of indissociable literals:

$$\begin{array}{rl} \neg p \leftarrow & \qquad a \leftarrow b \\ p \leftarrow \sim a & \qquad b \leftarrow c \\ & \qquad c \leftarrow a \end{array}$$

Example 14. Let P be:

$$\begin{array}{lll} \neg p \leftarrow & a \leftarrow b & a \leftarrow c \\ p \leftarrow \sim a & b \leftarrow a & \end{array}$$

We have:

$$\begin{aligned} SS(\sim c) &= \{\sim c\} \\ SS(\sim b) &= \{\sim a, \sim b, \sim c\} \\ SS(\sim a) &= \{\sim a, \sim b, \sim c\} \end{aligned}$$

and the unique indissociable set of literals is $\{\sim c\}$ which is also the set of revising literals.

Given the revising literals we find on which the contradiction rests. This is done by finding the supports of $\bot$ where revising literals occur only as leaves (these constitute the corresponding assumption sets):

Definition 24 Assumption set. Let P be a program with (pseudo) WFM M_P and $L \in M_P$. An assumption set $AS(L)$ is defined as follows, where $\mathcal{R}_P$ is the set of the revising literals induced by P :

1. If L is a classical literal:
   (a) if there is a fact for L then the only $AS(L) = \{\}$.
   (b) for each rule $L \leftarrow B_1, \ldots, B_n (n \geq 1)$ in P for L such that $\{B_1, \ldots, B_n\} \subseteq M_P$, there exists one $AS_k(L) = \bigcup_i AS_{j(i)}(B_i)$ for each different combination k of one $j(i)$ for each i.
2. For a default literal $\sim L$:
   (a) if $\sim L \in \mathcal{R}_P$ then the only $AS(\sim L) = \{\sim L\}$.
   (b) if $\sim L \in co - \mathcal{R}_P$ then there is a $AS(\sim L) = \{\sim L\}$.
   (c) if $\sim L \notin co - \mathcal{R}_P$ then choose from every rule for L a single literal whose default complement belongs to M_P. For each such choice there exist several $AS(\sim L)$; each contains one assumption set of each default complement of the chosen literals.
   (d) if $\neg L \in M_P$ then there are, **additionally**, assumption sets $AS(\sim L) = AS_k(\neg L)$ for each k.

Definition 25. A program P is *revisable* iff no assumption set of $\bot$ is empty.

This definition entails a program P is not revisable if $\bot$ has some support without co–revising literals.

Example 15. Consider $P = \{\neg a;\ a \leftarrow \sim b\ ; b \leftarrow \sim c\ ; c\}$ with

$$M_P = \{\bot, a, \sim a, \neg a, \sim b, c\}.$$

The only support of $\bot$ is $SS(\bot) = \{\bot, a, \sim a, \neg a, \sim b, c\}$. $co - \mathcal{R}_P = \{\}$. Thus $AS(\bot) = \{\}$ and the program is not revisable.

Definition 26 Removal set. A Removal Set (RS) of a literal L of program P is a set of literals formed by the union of one **non–empty** subset from each $AS_P(L)$.

Note that although the program may induce revising literals, it is not enough for a program to be revisable.

In order to make minimal changes that preserve the indissociability of literals we define:

Definition 27 Minimal contradiction removal sets. Let R be a minimal removal set of $\bot$. A Minimal Contradiction Removal Set of program P is the smallest set $MCRS$ such that $R \subseteq MCRS$ and $MCRS$ is inclusive of indissociable literals.

Definition 28 Contradiction removal sets. A contradiction removal set (or CRS for short) of a program P is either a $MCRS$ or the union of $MCRS$s.

Example 16. ex:review $P = \{a \leftarrow \sim b \; ; \;\; b \leftarrow \sim a \; ; \;\; \neg a\}$.

$$DS(\neg a) = \{\} \; DS(a) = \{\sim b\} \; DS(\sim b) = \{a\}$$

$$DS(\bot_a) = DS(a) \cup DS(\neg a) \supseteq \{\neg a, a, \sim b\}$$

The M_P is obtained as follows:

$$\begin{array}{ll} I_0 = \emptyset & I_3 = \{\neg a, \sim a, \sim\neg b, b\} = \Theta^x_{I_2}\uparrow^w \\ I_1 = \{\neg a, \sim\neg b\} = \Theta^x_{I_0}\uparrow^w & I_4 = \{\neg a, \sim a, \sim\neg b, b\} = \Theta^x_{I_3}\uparrow^w \\ I_2 = \{\neg a, \sim a, \sim\neg b\} = \Theta^x_{I_1}\uparrow^w & I_5 = I_4 \end{array}$$

$M_P = \{\neg a, \sim a, b, \sim\neg b\}$ and $\bot_a \notin M_P$; thus the program is non–contradictory.

Example 17. Consider $P = \{a \leftarrow \sim b \; ; \neg a \leftarrow \sim c \; ; d \leftarrow a \; ; e \leftarrow \neg a\}$.

Since $\sim b$ and $\sim c$ are both revising literals $AS(\bot) = \{\sim b, \sim c\}$. The contradiction removal sets are:

$$\begin{array}{l} CRS_1 = RS_1(\bot_a) = \{\sim b\} \\ CRS_2 = RS_2(\bot_a) = \{\sim c\} \\ CRS_3 = RS_3(\bot_a) = \{\sim b, \sim c\} \end{array}$$

Example 18. Let P be:

$$\begin{array}{ll} a \leftarrow \sim b & \neg a \leftarrow \sim f \\ \neg a \leftarrow \sim d & \neg d \leftarrow \sim e \end{array}$$

with $M_P = \{\sim b, \sim\neg b, \sim d, \sim e, \sim\neg e, \sim f, \sim\neg f, a, \neg a, \neg d, \sim a, \sim\neg a\}$.

Note that $\sim d$ is not co–revising, since there exists $SS(\sim d) = \{\sim d, \neg d, \sim e\}$. The revising literals are $\sim b$, $\sim e$, and $\sim f$. The assumptions sets are:

$$\begin{array}{l} AS_1(\bot) = \{\sim b, \sim e\} \\ AS_2(\bot) = \{\sim b\} \\ AS_3(\bot) = \{\sim b, \sim f\} \end{array}$$

Thus the only contradiction removal set is $\{\sim b\}$.

Example 19. Consider the program $P = \{\neg a; a \leftarrow \sim d; \neg d \leftarrow \sim e\}$. We have $M_P = \{\sim e, \sim \neg e, \neg d, \sim d, a, \neg a, \sim \neg a, \sim a\}$ which is contradictory. The only revising literal is $\sim e$ and $\sim d$ is a co–revising literal. Hence one $AS(\sim d) = \{\sim d\}$ and the other $AS(\sim d) = \{\sim e\}$. The only CRS is $\{\sim d, \sim e\}$.

4.3 Contradiction Free Programs

Next we show that for each contradiction removal set there is a non-contradictory program obtained from the original one by a simple update. Based on these programs we define the $CRSX$ semantics.

Definition 29 CWA inhibition rule. The CWA inhibition rule for an atom A is $A \leftarrow \sim A$.

Any program P containing a CWA inhibition rule for atom A has no models containing $\sim A$.[11]

Definition 30 Contradiction free program. For each contradiction removal set CRS_i of a program P we engender the contradiction free program:

$$P_{CRS_i} =_{def} P \cup \{A \leftarrow \sim A |\ \sim A \in CRS_i\} \tag{3}$$

Proposition 31. For any contradiction removal sets i and j, $CRS_i \subseteq CRS_j \Rightarrow M_{P_{CRS_j}} \subseteq M_{P_{CRS_i}}$.

Theorem 32 Soundness of contradiction free programs. A contradiction free program P_{CRS} is non–contradictory, i.e. it has $WFSX$ semantics, and $M_{P_{CRS}} \subseteq M_P$.

Example 20. Consider the program P :

$$\begin{array}{ll} a \leftarrow \sim b & \neg c \leftarrow \sim d \\ b \leftarrow \sim a, \sim c & c \leftarrow \sim e \end{array}$$

The well founded model is

$$M_P = \{\bot_c, \sim d, \sim\neg d, \sim e, \sim\neg e, \neg c, c, \sim\neg c, \sim c, \sim b, \sim\neg b, a, \sim\neg a\}.$$

The contradiction removal sets are:

$$CRS_1 = \{\sim d\}\ \ CRS_2 = \{\sim e\}\ \ CRS_3 = \{\sim d, \sim e\}$$

with CRS_1 and CRS_2 being minimal w.r.t. set inclusion.

[11] This rule can be seen as the *productive* integrity constraint $\leftarrow \sim A$. In fact, since the WF Semantics implicitly has in it the *productive* constraint $\leftarrow A, \sim A$, the inhibition rule can be seen as the minimal way of expressing by a program rule that $\sim A$ leads to an inconsistency.

– $P_{CRS_1} = P \cup \{d \leftarrow \sim d\}$, with the unique model

$$M_P = \{\sim e, \sim\neg e, c, \sim\neg c, \sim b, \sim\neg b, a, \sim\neg a, \sim\neg d\}.$$

– $P_{CRS_2} = P \cup \{e \leftarrow \sim e\}$, with well founded model

$$M_P = \{\sim d, \neg c, \sim c, \sim\neg a, \sim\neg b, \sim\neg d\}.$$

– $P_{CRS_3} = P \cup \{e \leftarrow \sim e, d \leftarrow \sim d\}$ with well founded model

$$M_P = \{\sim\neg a, \sim\neg b, \sim\neg e, \sim\neg d\}.$$

Definition 33 $CRSX$ **Semantics.** Given a revisable contradictory program P let CRS_i be any contradiction removal set for P. An interpretation I is a $CRSX$ model of P iff:

$$I = \Phi_{P_{CRS_i}}(I) \ . \tag{4}$$

The least (w.r.t. $\subseteq$) $CRSX$ model of P is called the $CRWFM$ model[12].

The contradiction removal semantics logic programs extended with explicit negation is defined by the $WFSX$ well founded models of the revised programs defined by (4), representing the different forms of revising a contradictory program.

Example 21. For program P_1 of example 12 the assumption sets are $AS_{1_1} = \{\sim b\}$ and $AS_{1_2} = \{\sim b, \sim c\}$. Thus the only CRS is $\{\sim b\}$, and the only $CRSX$ model is $\{\neg a, \sim a, \sim c\}$. For program P_2 the only assumption set is $AS_2 = \{\sim c\}$. Thus the only CRS is $\{\sim c\}$, and the only $CRSX$ model is $\{\neg a, \sim a\}$.

Example 22. Let P be:

$$\begin{array}{ll} a \leftarrow \sim b & b \leftarrow c \\ \neg a \leftarrow \sim c & c \leftarrow b \end{array}$$

The only self supported set is $S = \{\sim b, \sim c\}$. Moreover the only support of S is itself. Thus $\sim b$ and $\sim c$ are revising and indissociable. As the only assumption set of $\perp$ is $\{\sim b, \sim c\}$ there are three removal sets: $\{\sim b\}$, $\{\sim c\}$, and $\{\sim b, \sim c\}$. Without indissociability one might think that for this program there would exist three distinct ways of removing the contradiction. This is not the case, since the XSMs of P_{R_1}, P_{R_2}, and P_{R_3} are exactly the same, i.e. they all represent the same revision of P. This is accounted for by introducing indissociable literals in minimal contradiction removal sets. In fact there exists only one $MCRS$ $\{\sim b, \sim c\}$ and thus the only contradiction free program is P_{R_3}.

Theorem 34. For every i, j

$$CRS_i \neq CRS_j \Rightarrow WFM(P_{CRS_i}) \neq WFM(P_{CRS_j}) \ .$$

Theorem 35. The collection of contradiction free programs is a upper semi–lattice under set inclusion of rules in programs, and the set of revised models under set inclusion is a lower semi-lattice. There is a one–to–one correspondence between elements of both semi-lattices.

[12] This model always exists (cf. theorem 35.)

Acknowledgements

We thank Esprit BRA Compulog (no. 3012) and Compulog 2 (no. 6810), INIC and JNICT for their support.

References

1. J. J. Alferes and L. M. Pereira. On logic program semantics with two kinds of negation. In K. Apt, editor, *IJCSLP'92*, pages 574–588. MIT Press, 1992.
2. P. M. Dung and P. Ruamviboonsuk. Well founded reasoning with classical negation. In A. Nerode, W. Marek, and V. S. Subrahmanian, editors, *LPNMR'91*, pages 120–132. MIT Press, 1991.
3. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R. A. Kowalski and K. A. Bowen, editors, *5th ICLP*, pages 1070–1080. MIT Press, 1988.
4. M. Gelfond and V. Lifschitz. Logic programs with classical negation. In Warren and Szeredi, editors, *ICLP*, pages 579–597. MIT Press, September 1990.
5. K. Inoue. Extended logic programs with default assumptions. In Koichi Furukawa, editor, *ICLP'91*, pages 490–504. MIT Press, 1991.
6. R. Kowalski. Problems and promises of computational logic. In John Lloyd, editor, *Computational Logic Symposium*, pages 1–36. Springer-Verlag, 1990.
7. R. Kowalski and F. Sadri. Logic programs with exceptions. In Warren and Szeredi, editors, *ICLP*, pages 598–613. MIT Press, 1990.
8. J. W. Lloyd. *Foundations of Logic Programming.* Symbolic Computation. Springer–Verlag, 1984.
9. L. M. Pereira and J. J. Alferes. Well founded semantics for logic programs with explicit negation. In B. Neumann, editor, *ECAI'92*, pages 102–106. John Wiley & Sons, Ltd, 1992.
10. 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, *LPNMR*, pages 105–119. MIT Press, 1991.
11. L. M. Pereira, J. J. Alferes, and J. N. Aparício. The extended stable models of contradiction removal semantics. In P. Barahona, L. M. Pereira, and A. Porto, editors, *EPIA'91*, LNAI 541, pages 105–119. Springer–Verlag, 1991.
12. L. M. Pereira, J. J. Alferes, and J. N. Aparício. Default theory for well founded semantics with explicit negation. In D. Pearce and G. Wagner, editors, *Logics for AI - JELIA'92*, LNAI 633, pages 339–356. Springer-Verlag, 1992.
13. L. M. Pereira, J. J. Alferes, and J. N. Aparício. Well founded semantics with explicit negation and default theory. Technical report, AI Centre, Uninova, March 1992. *Submitted.*
14. L. M. Pereira, J. N. Aparício, and J. J. Alferes. Counterfactual reasoning based on revising assumptions. In V. Saraswat and K. Ueda, editors, *ILPS*, pages 566–580. MIT Press, 1991.
15. L. M. Pereira, J. N. Aparício, and J. J. Alferes. Hypothetical reasoning with well founded semantics. In B. Mayoh, editor, *Third Scandinavian Conf. on AI.* IOS Press, 1991.
16. L. M. Pereira, J. N. Aparício, and J. J. Alferes. Nonmonotonic reasoning with well founded semantics. In K. Furukawa, editor, *ICLP91*, pages 475–489. MIT Press, 1991.

17. L. M. Pereira, J. N. Aparício, and J. J. Alferes. Non–monotonic reasoning with logic programming. *Journal of Logic Programming. Special issue on Nonmonotonic reasoning*, 1993. To appear.
18. 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.
19. T. Przymusinski. Extended stable semantics for normal and disjunctive programs. In Warren and Szeredi, editors, *ICLP'90*, pages 459–477. MIT Press, 1990.
20. T. C. Przymusinski. Extended stable semantics for normal and disjunctive programs. In Warren and Szeredi, editors, *ICLP90*, pages 459–477. MIT Press, 1990.
21. T. C. Przymusinski. A semantics for disjunctive logic programs. In D. Loveland, J. Lobo, and A. Rajasekar, editors, *ILPS Workshop on Disjunctive Logic Programs*, 1991.
22. A. Van 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.
23. 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.