Contradiction: when avoidance equals removal Part II

Luís Moniz Pereira and José Júlio Alferes

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

Abstract. This paper is the continuation of [1] in this volume. There we present a sceptical semantics which avoids contradiction for extended logic programs plus integrity contraints in the form of denials, based on the notion of optative hypotheses *–an abductive approach.* In this part we define a program revision method for removing contradiction from contradictory programs under *WFSX*, based on the notion of revisable hypotheses *–a belief revision approach–* and show the equivalence between the contradiction avoidance semantics and the *WFSX* of revised programs obtained by contradiction removal.
The motivation, as well as some preliminary definitions can be found in [1]. Proofs of all theorems are omitted for brevity, but exist in an extended version of this work.

1 Introduction

It was argued in the introduction of [1] that, to deal with the issue of contradiction brought about by closed world assumptions, rather than defining more sceptical semantics one can rely instead on a less sceptical semantics and accompany it with an assumption revision process that restores consistency.

In this part we define such a revision process for programs contradictory with respect to *WFSX*, that relies on taking back assumptions about the truth of negative literals. The set of negative literals over which a revision can be made, i.e. the assumption of their truthfulness can be removed, is called the set of *revisable literals*, and can be any subset of *not* $\mathcal{H}$.

In [6] a revision semantics was defined where only base closed world assumptions are revisable, i.e. those default literals whose complement has no rules. In [8] the notion of base closed world assumption was improved, in order to deal with the case of direct loops, i.e. without interposing *nots*[2]. The notion of revisables there presented is similar to the notion of prime optatives in [1].

We show in section 5 that the issue of which are the revisables (in contradiction removal) is tantamount to that of which are the optatives (in contradiction

* We thank JNICT and Esprit BR project Compulog 2 (no 6810) for their support.

[2] E.g. if *not* a is considered a base closed world assumption because a program has no rules for a, then there is no reason for *not* a not being so considered if the only rule for a is $a \leftarrow a$.

avoidance). Thus the discussion on primacy of optatives in [1] is applicable to the issue of what literals are to be revisables. So no restriction is made here on which default literals are considered revisables, and they are supposed to be provided along with the program[3].

For instance, in the wobbly wheel example of part I, the revisable literals might be:

$$\{not\ fd, not\ lv, not\ pt, not\ bs\}.$$

By not introducing *not ft* in this set, we are declaring that, in order to remove contradiction, we will not consider directly revising its truth. However, this does not mean that by revision of some other literal the truth value of *not ft* will not change.

We take back revisable assumptions, i.e. assumptions on the truthfulness of revisable literals, in a minimal way and in all alternative ways of removing contradiction. Moreover, we identify a single unique revision that defines a sceptical revision process which includes all alternative contradiction removing revisions, so as not to prefer one over the other.

The structure of this part is as follows: first we present *WFSX* and a paraconsistent extension of it. Then we identify the intended revisions declaratively. Afterwards we define some useful constructible sets for establishing the causes of and the removal of contradiction within *WFSX*, and prove that the result of their use concurs with the intended revisions defined. Finally we show the equivalence between the contradiction avoidance semantics and the *WFSX* of revised programs obtained by contradiction removal.

2 Paraconsistent *WFSX*

In this section we present both the *WFSX* and its paraconsistent extension. The presentation is focused in the paraconsistent extension, and the special case of *WFSX* is pointed out.

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

Example 1. Consider program P :

$$\begin{array}{rlrl} a \leftarrow not\ b & \text{(i)} & d \leftarrow not\ a & \text{(iii)} \\ \neg a \leftarrow not\ c & \text{(ii)} & e \leftarrow not\ \neg a & \text{(iv)} \end{array}$$

[3] The declaration of revisable literals by the user is akin to that of abducible literals. Although some frameworks identify what are the abducibles for some particular problems (cf. [4] where they are of the form a^*), theories of abduction, for the sake of generality, make no restriction on which literals are abducible, and assume them to be provided by the user.

1. *not* b and *not* 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. *not* a and *not* $\neg a$ hold from 2 and the coherence principle[4] relating the two negations
4. d and e hold from 3 and rules (iii) and (iv)
5. *not* d and *not* e hold from 2 and rules (iii) and (iv), as they are the only rules for d and e
6. *not* $\neg d$ and *not* $\neg e$ hold from 4 and the coherence principle.

The whole set of literal consequences is then:

$$\{not\ b, not\ c, \neg a, a, not\ a, not\ \neg a, d, e, not\ d, not\ e, not\ \neg d, not\ \neg e\}.$$

For the purpose of defining *WFSX* and its paraconsistent extension, we begin by defining what is an interpretation.

Definition 1. A *p-interpretation* I is any set $T \cup not\ F$, such that if $\neg L \in T$ then $L \in F$.

A p-interpretation is an *interpretation* iff T and F are disjoint.

Let $QI = QT \cup not\ QF$ be a set of literals. We define $Coh^p(QI)$ as the p-interpretation $T \cup not\ F$ such that $T = QT$ and $F = QF \cup \{\neg L \mid L \in T\}$.

The definition of *WFSX* (in [5]) is based on a modulo transformation. Without loss of generality, and for the sake of technical simplicity, we consider that programs are always in their canonical form, i.e. for each rule of the program and any objective literal, if L is in the body then *not* $\neg L$ also belongs to the body of that rule[5].

Definition 2. Let P be an canonical extended logic program, and I an interpretation. Then $\frac{P}{I}$ *(P modulo I* is the program obtained from P by performing the following three operations: remove from P all rules containing a default literal $L = not\ A$ such that $A \in I$; remove from all remaining rules of P their default literals $L = not\ A$ such that $not\ A \in I$; replace all the remaining default literals by proposition $\mathbf{u}$.

In this definition one can apply the first two operations in any order, because the conditions of their application are disjoint for any interpretation. A potential conflict would rest on applying both the first and the second operation, but that can never happen because if some $A \in I$ then $not\ A \notin I$, and vice-versa.

[4] Recall, from part I, that it is stated as: if $\neg L$ holds, *not* L holds too, for every objective literal L.

[5] When the coherence principle is adopted, the truth value of L coincides with that of $(L, not\ \neg L)$. Taking programs in canonical form simplifies the techniques since we don't need to concern ourselves with objective literals in the bodies in the modulo transformation, but only with default literals, just as for non-extended programs.

This is not the case for p-interpretations pI, where for some objective literal A both A and $not\ A$ might belong to pI. Thus if one applies the transformation to p-interpretations, different results are obtained depending on the order of the application of the first two operations.

Example 2. Consider P of example 1, and let us compute:

$$\frac{P}{\{a, \neg a, not\ \neg a, not\ a, not\ b, not\ c\}}.$$

If one applies the operations in the order they are presented: Rules (iii) and (iv) of P are removed because both a and $\neg a$ belong to the p-interpretation; $not\ b$ and $not\ c$ are removed from the bodies of rules since $not\ b$ and $not\ c$ belong to the p-interpretation. The resulting program is $\{a \leftarrow\ ;\ \neg a \leftarrow\}$.

But if one applies the second operation first: $not\ b$, $not\ c$, $not\ a$, and $not\ \neg a$ are removed from the bodies of rules since $not\ b$, $not\ c$, $not\ a$, and $not\ \neg a$ belong to the p-interpretation; Since no literals remain in the body of rules no other operation is applicable. The resulting program in this case is $\{a \leftarrow d;\ \neg a \leftarrow e\}$.

In order to make the transformation independent of the order of application of the operations we define the corresponding transformation for the paraconsistent case as being nondeterministic in the order of application of those rules.

Definition 3. Let P be an canonical extended logic program and let I be a p-interpretation. By *a* $\frac{P}{I}p$ *program* we mean any program obtained from P by first non-deterministically applying the operations until they are no longer applicable:

- Remove all rules containing a default literal $L = not\ A$ such that $A \in I$.
- Remove from rules their default literals $L = not\ A$ such that $not\ A \in I$.

and by next replacing all remaining default literals by proposition $\mathbf{u}$.

In order to get all consequences of the program, even those leading to contradictions, as well as those arising from contradictions, we consider the consequences of all possible such $\frac{P}{I}p$ programs.

Definition 4. Let P be an canonical extended logic program, I a p-interpretation, and let $P_1, \ldots P_n$ be all the possible results of $\frac{P}{I}p$. Then:

$$\Phi^p{}_P(I) = \bigcup_{1 \le i \le n} Coh^p(least(P_i))$$

Theorem 5. *The Φ^p operator is monotone under set inclusion of p-interpretations.*

Given that Φ^p is monotonic, then for every program it always has a least fixpoint, and this fixpoint can be obtained by iterating Φ^p starting from the empty set:

Definition 6. The *paraconsistent* WFSX of a (canonical) extended logic program P, denoted by $WFSX_p(P)$, is the least fixpoint of Φ^p applied to P.

If some literal L belongs to the paraconsistent *WFSX* of P we write $P \vdash_p L$.

Proposition 7. WFSX_p *is defined for every program with ICs.*

Now we can give a definition of a contradictory program with ICs:

Definition 8. A program P with language $Lang$ where A is an atom, and a set of integrity constraints IC is *contradictory* iff

$$P \cup IC \cup \{\bot \leftarrow A, \neg A \mid A \in Lang\} \vdash_p \bot$$

In this section we always refer to the paraconsistent *WFSX* as an extension of *WFSX* for non-contradictory programs. This is so because:

Proposition 9. *For a non-contradictory program P the paraconsistent* WFSX *coincides with* WFSX.

3 Declarative Revisions

Before tackling the question of which assumptions to revise to abolish contradiction, we begin by showing how to impose in a program a revision that takes back some revisable assumption, identifying rules of a special form, which have the effect of prohibiting the falsity of an objective literal in models of a program. Such rules can prevent an objective literal being false, hence their name:

Definition 10. The *inhibition rule* for a default literal $not\ L$ is $L \leftarrow not\ L$. By $IR(S)$ where S is a set of default literals, we mean:

$$IR(S) = \{L \leftarrow not\ L \mid not\ L \in S\}$$

These rules state that if $not\ L$ is true then L is also true, and so a contradiction arises. Intuitively this is quite similar to the effect of integrity rules of form $\bot \leftarrow not\ A$. Technically, the difference is that the removal of such a contradiction in the case of inhibition rules is dealt with by *WFSX* itself, whereas in the case of the integrity rules it isn't.

Proposition 11. *Let P be any program such that for some objective literal L, $P \not\vdash_p \neg L$. Then $P \cup \{L \leftarrow not\ L\} \not\vdash_p not\ L$. Moreover, if there are no other rules for L, the truth value of L is undefined in* $\mathrm{WFSX}_p(P)$.

These rules allow, by adding them to a program, one to force default literals in the paraconsistent *WFSX* to become undefined. Note that changing the truth value of a revisable literal from true to undefined is less committing than changing it to false[6].

[6] In order to obtain revisions where the truth value of revisable literals is changed from true to false, one has to iterate the process we're about to define. The formal definition of such revisions can be found in [13].

To declaratively define the intended program revisions void of contradiction we start by considering the resulting *WFSX*s of the programs obtained from all possible ways of adding to a program P inhibition rules for revisable literals (some may still be contradictory programs).

Several different revisions might be equivalent, from the standpoint of their consequences.

Example 3. Let $P = \{\bot \leftarrow not\ a;\ a \leftarrow b;\ b \leftarrow a;\ a \leftarrow c\}$, with revisables $\{not\ a, not\ b, not\ c\}$.

Note that adding $a \leftarrow not\ a$, $b \leftarrow not\ b$, or both, has the same consequences, since undefining a leads to the undefinedness of b and vice-versa. Considering all three as distinct can be misleading because it appears that the program has three different revisions.

Revisables $not\ a$ and $not\ b$ are said to be indissociable, and it is indifferent to introduce inhibition rules for one, the other, or both. In the sequel, we coalesce the three revisions into a single standard one, that adds both inhibition rules.

Definition 12. Let P be an extended logic program with revisables Rev. The set $Ind(S) \supseteq S$ of *indissociable literals* of a set of default literals S is a subset of Rev such that:

- $Ind(S) \subseteq WFSX_p(P)$ and $WFSX_p(P \cup IR(S)) \cap Ind(S) = \{\}$

i.e. $Ind(S)$ is the set of all revisables that change their truth value from true to undefined, once inhibition rules are added for every default literal of S to change their value.

Example 4. In example 3 $Ind(\{not\ a\}) = Ind(\{not\ b\}) = \{not\ a, not\ b\}$ and $Ind(\{not\ c\}) = \{not\ a, not\ b, not\ c\}$.

Definition 13. A *submodel of a program P* with integrity rules, and revisables Rev, is any pair $\langle M, R\rangle$ where R is a subset of Rev closed under indissociables, i.e $\forall S \subseteq R,\ Ind(S) \subseteq R$, and $M = WFSX_p(P \cup \{L \leftarrow not\ L \mid not\ L \in R\})$[7].

In a submodel $\langle M, R\rangle$ we dub R the submodel revision, and M are the consequences of the submodel revision. A submodel is contradictory iff M is contradictory (i.e. either contains $\bot$ or is not an interpretation). Note how there is a one-to-one correspondence between submodels and program revisions.

The existence of $WFSX_p(P)$ for any program P (cf. proposition 7) grants that M exists for every subset of Rev. Thus:

Proposition 14. *The submodels $\langle M, R\rangle$ of any program P with revisables Rev forms a complete lattice under set inclusion on the submodel revisions.*

Example 5. Let $P = \{p \leftarrow not\ q;\ \neg p \leftarrow not\ r;\ a \leftarrow not\ b\}$, with $Rev = \{not\ q, not\ r, not\ b\}$. Its submodels lattice is depicted in fig. 1. For simplicity, contradictory models are not presented in full.

[7] For a study of submodels based on the XSMs instead of on the well-founded model see [7].

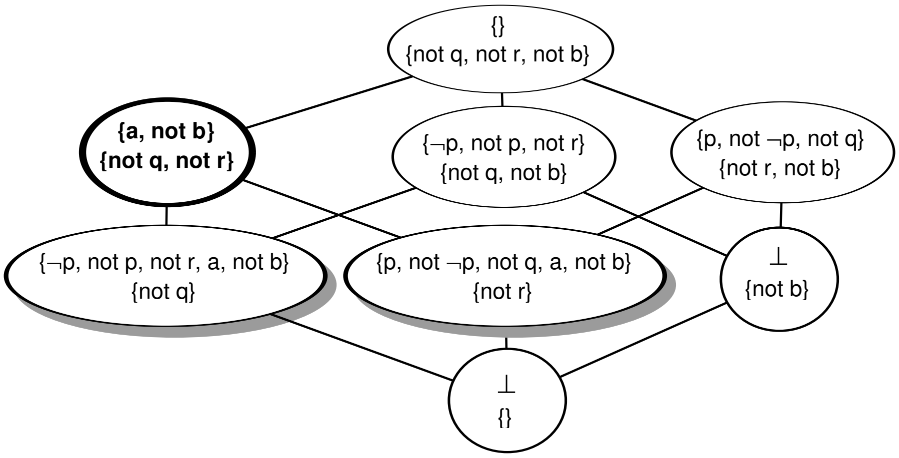

Fig. 1. Submodels lattice of example 5

As we are interested in revising contradiction in a minimal way, we care about those submodels that are non-contradictory and, among these, about those that are minimal in the submodels lattice.

Definition 15. A submodel $\langle M, R\rangle$ is a *minimal non-contradictory submodel* (MNS for short) of P iff it is non-contradictory and there exists no other non-contradictory submodel $\langle M', R'\rangle$, such that $R' \subset R$.

Let P be a program with revisables Rev, and $\langle M, R\rangle$ some MNS of P. A *minimally revised program* MRP of P is: $P \cup IR(R)$.

By definition, each MNS of a program P reflects a revision of P, $P \cup RevRule$[8], that guarantees non-contradiction, and such that for any set of rules $RevRule' \subseteq RevRule$ closed under indissociables, $P \cup RevRule'$ is contradictory. In other words, each MNS reflects a revision of the program that restores consistency, and which adds a minimal set of inhibition rules for revisables.

Proposition 16. *If P is non-contradictory its single MNS is $\langle \mathrm{WFSX}(P), \{\}\rangle$, and P itself is its only minimally revised program.*

Example 6. The MNSs of P in example 5 are the shadowed submodels of fig. 1, and the minimally revised programs are:

$$MRP_1 = \{p \leftarrow not\ q; \neg p \leftarrow not\ r; a \leftarrow not\ b; q \leftarrow not\ q\}\ \ and$$
$$MRP_2 = \{p \leftarrow not\ q; \neg p \leftarrow not\ r; a \leftarrow not\ b; r \leftarrow not\ r\}.$$

[8] Where $RevRule$ is the set of inhibition rules for the submodel revision.

Each of these two programs is a transformation of the original one that minimally removes contradiction by taking back the assumption of truth of some revisables via their inhibition rules. In this example, one can remove the contradiction in p either by going back on the closed world assumption of falsity of q (or truth of *not* q) or on the falsity of r. The program that has the first effect is MRP_1, the one with the second being MRP_2. Having no reason to render q alone or r alone undefined, it is natural that a sceptical revision should accomplish the effect of undefining them both.

Definition 17. The *sceptical submodel* of a program P is the join $\langle M_J, R_J\rangle$ of all MNSs of P. The *sceptical revised program* of P is $P \cup IR(R_J)$.

Example 7. The sceptical submodel of P of example 5 is depicted in bold in fig. 1. Note that inhibiting b is irrelevant for revising P, and how taking the join of the MNSs captures what is required.

It is important to guarantee that the sceptical revision indeed removes contradiction from a program. This is so because:

Proposition 18. *Let $\langle M_1, R_1\rangle$ and $\langle M_2, R_2\rangle$ be any two non-contradictory submodels. Then submodel $\langle M, R_1 \cup R_2\rangle$ is also non-contradictory.*

Example 8. Consider program P :

$$\begin{array}{lll} p \leftarrow not\ a & q \leftarrow not\ r & \neg a \leftarrow not\ b \\ \neg p \leftarrow not\ a & r \leftarrow not\ s & \end{array}$$

with $Rev = \{not\ q, not\ a, not\ b\}$.

Its MNSs are:

$$\begin{array}{l} \langle \quad \{r, not\ s, p, not\ q\} \quad , \{not\ a, not\ b\}\rangle \\ \langle\{r, not\ s, \neg p, \neg a, not\ a, not\ b\}, \quad \{not\ q\} \quad \rangle \end{array}$$

Its sceptical submodel is $\langle\{r, not\ s\}, \{not\ a, not\ b, not\ q\}\rangle$.

It is clear that with these intended revisions some programs have no revision. This happens when contradiction has a basis on non-revisable literals.

Example 9. Let $P = \{a \leftarrow not\ b;\ b \leftarrow not\ c;\ \neg a;\ c\}$, with $Rev = \{not\ c\}$.

The only submodels of P are:

$$\langle WFSX_p(P), \{\}\rangle \text{ and } \langle WFSX_p(P \cup \{c \leftarrow not\ c\}), \{not\ c\}\rangle.$$

As both these submodels are contradictory, P has no MNS and thus no revisions. Note that if *not* b were revisable, the program would have a revision $P \cup \{b \leftarrow not\ b\}$. If *not* b were absent from the first rule, P would have no revision no matter what the revisables.

Definition 19. A contradictory program P with revisables Rev is *unrevisable* iff it has no non-contradictory submodel.

4 Contradiction Support and Removal

Submodels characterize which are the possible revisions and the minimality criterium. Of course, a procedure for finding the minimal and the more sceptical submodels can hardly be based on their declarative definition: one has to generate all the possible revisions to select those intended. In this section we define a revision procedure, and show that it concurs with the declaratively intended revisions.

The procedure relies on the notions of contradiction supports and of contradiction removal sets. Informally, contradiction supports are sets of revisable literals present in the $WFSX_p$ which are sufficient to support $\perp$, i.e. contradiction[9]. From their truth the truth of $\perp$ inevitably follows.

Contradiction removal sets are built from the contradiction supports. They are minimal sets of literals chosen from the supports such that any support of $\perp$ has at least one literal in the removal set. Consequently, if all literals in some contradiction removal set were to become undefined in value then no support of $\perp$ would subsist. Thus removal sets are the hitting sets of the supports.

Example 10. Consider program P of example 5. Its only contradiction support is $\{not\ q, not\ r\}$, and its contradiction removal sets are $\{not\ q\}$ and $\{not\ r\}$.

Suppose we had q undefined as a result of introducing rules for q. In that case $\perp$ would also be undefined, the program becoming non-contradictory. The same would happen if r alone became undefined. No other set, not containing one of these two alternatives, has this property.

Definition 20. Given a program P with revisables Rev, the *supports of a literal* L belonging to the $WFSX_p$ (any of which represented as $SS(L)$) are obtained as follows:

1. If L is an objective literal:
   (a) If there is a fact for L then a support of L is $SS(L) = \{\}$.
   (b) For each rule $L \leftarrow B_1, \ldots, B_n$ $(n \geq 1)$ in P such that $\{B_1, \ldots, B_n\} \subseteq WFSX_p(P)$, there is a support $SS(L) = \bigcup_i SS_{j(i)}(B_i)$, for each combination of one $j(i)$ for each i.
2. If $L = not\ A$ (where A is an objective literal):
   (a) If $L \in Rev$ then a support of L is $SS(L) = \{L\}$.
   (b) If $L \notin Rev$ and there are no rules for A then a support of L is $SS(L) = \{\}$.
   (c) If $L \notin Rev$ and there are rules for A, choose from each rule with a non-empty body for A, a literal such that its default complement belongs to $WFSX_p(P)$. For each such multiple choice there exist several $SS(L)$; each contains one support of each default complement of the chosen literals.
   (d) If $\neg A \in WFSX_p(P)$ there are, additionally, supports $SS(L) = SS_k(\neg A)$ for each k.

[9] This notion is a special case of the notion of Suspect Sets introduced in declarative debugging in [12]

Example 11. Consider program P of example 8, whose paraconsistent well-founded consequences are:

$$WFSX_p(P) = \{not\ s, r, not\ q, p, not\ \neg p, not\ b, \neg a, not\ a, \neg p, not\ p\}.$$

The supports of p are computed as follows: from the only rule for p we conclude that the supports of p are the supports of $not\ q$; since $not\ q$ is a revisable then one of its supports is $\{not\ q\}$; as $\neg q \notin WFSX_p(P)$, there are no other supports of q. Thus the only support of p is $\{not\ q\}$.

The supports of $\neg p$ are: from the only rule for $\neg p$ conclude that the supports of $\neg p$ are the supports of $not\ a$; since $not\ a$ is a revisable then one of its supports is $\{not\ a\}$; since $\neg a \in WFSX_p(P)$ supports of $\neg a$ are also supports of $not\ a$; from the only rule for $\neg a$ conclude that the supports of $\neg a$ are the supports of $not\ b$; likewise $not\ q$ above, the only support of $not\ b$ is $\{not\ b\}$. Thus $\neg p$ has two supports, namely $\{not\ a\}$ and $\{not\ b\}$.

Proposition 21. *A literal L belongs to the* WFSX$_p$ *of a program P iff it has at least one support $SS(L)$.*

Definition 22. A *contradiction support* of a program P is a support of $\perp$ in the program obtained from P by adding to it integrity rules of the form $\perp \leftarrow L, \neg L$ for every objective literal L in the language of P.

N.B. From now on, unless otherwise stated, when we refer to a program we mean the program obtained by adding to it all such rules.

Example 12. The contradiction supports of P in example 8 are the unions of pairs of supports of p and $\neg p$.

Thus, according to the supports of in example 11, P has two contradiction supports, namely $\{not\ q, not\ a\}$ and $\{not\ q, not\ b\}$.

Contradiction supports are sets of revisables true in the $WFSX_p$ involved in some support of contradiction (i.e. $\perp$)[10].

Having defined the sets of revisables that together support some literal, it is easy to produce sets of revisables such that, if all become undefined, the truth of that literal would necessarily become ungrounded. To cope with indissociability, these sets are closed under indissociable literals.

Definition 23. A pre-removal set of a literal L belonging to the $WFSX_p$ of a program P is a set of literals formed by the union of some nonempty subset from each $SS(L)$.

A *removal set* (RS) of L is the closure under indissociable literals of a pre-removal set of L.

If the empty set is a $SS(L)$ then the only $RS(L)$ is, by definition, the empty set. Note that a literal not belonging to $WFSX_p(P)$ has no RSs defined for it.

[10] Note the close relationship between the SSs of $\perp$ and the sets of nogoods of Truth Maintenance Systems.

In view of considering minimal changes to the WF Model, we next define those RSs which are minimal in the sense that there is no other RS contained in them.

Definition 24 Minimal removal set. In a program P, $RS_m(L)$ is a *minimal removal set* iff there exists no $RS_i(L)$ in P such that $RS_m(L) \supset RS_i(L)$. We represent a minimal RS of L in P as $MRS_P(L)$.

A *contradiction removal set* of program P is a minimal removal set of the (reserved) literal $\perp$, i.e. a CRS of P is a $MRS_P(\perp)$.

Example 13. Consider program P of example 3. The only support of $\perp$ is $\{not\ a\}$. Thus the only pre-removal set of $\perp$ is also $\{not\ a\}$. Since $Ind(\{not\ a\}) = \{not\ b\}$, the only contradiction removal set is $\{not\ a, not\ b\}$.

Example 14. The removal sets of $\perp$ in the program of example 8 are:

$$\begin{array}{ll} RS_1 = \{not\ q\} & RS_2 = \{not\ q, not\ a\} \\ RS_3 = \{not\ q, not\ b\} & RS_4 = \{not\ a, not\ b\} \end{array}$$

Thus RS_1 and RS_4 are contradiction removal sets. Note that these correspond exactly to the revisions of minimal non-contradictory submodels of example 8.

It is important to guarantee that contradiction removal sets do indeed remove contradiction.

Lemma 25. *Let P be a contradictory program with contradiction removal set CRS. Then $P \cup IR(CRS)$ is non-contradictory.*

Now we state[11] that this process concurs with the intended revisions above:

Theorem 26 Correctness of CRSs.

1. *Let R be a nonempty CRS of a contradictory program P. Then $\langle M, R\rangle$ is a MNS of P, where $M = \mathrm{WFSX}(P \cup IR(R))$.*
2. *If $\{\}$ is a CRS of a program P then P is unrevisable.*
3. *Let $\langle M, R\rangle$ be a MNS, with $R \neq \{\}$, of a contradictory program P. Then R is a CRS of P.*
4. *Let P be a contradictory program with CRSs $R_1, \ldots, R_n$. The sceptical revised program of P is*

$$P \cup \left\{ L \leftarrow not\ L \mid\ not\ L \in \bigcup_{1 \le i \le n} R_i \right\}.$$

[11] The proof of correctness can be found in [2].

Thus in order to compute the minimal and sceptical submodels, one starts by computing all supports of $\perp$. Although the definition of support requires one to know a priori the paraconsistent $WFSX_p$, an alternative definition exists such that this is not required. This definition is based on a top-down derivation procedure. Computing all supports of $\perp$ is like computing all the derivations for $\perp$ in $WFSX_p$.

If $\{\}$ is a support of $\perp$ then the program is unrevisable. Otherwise, after having all supports of $\perp$, the rest follows by operations on these sets. For such operations one can rely on efficient methods known from the literature. For example the method of [15] for finding minimal diagnosis can be applied for finding CRSs given the supports. Finally, a minimal revised program is obtained by adding to P one inhibition rule for each element of a CRS, and the sceptical revision is obtained as the union of all such minimally revised programs.

Example 15. Consider the hiking/swimming program of part I, and let $Rev = \{not\ rain,\ not\ cold\_water\}$.

The supports of $\perp$ are $\{not\ rain\}$ and $\{not\ rain, not\ cold\_water\}$. Thus its removal sets are $\{not\ rain\}$, and $\{not\ rain, not\ cold\_water\}$. The only CRS is $\{not\ rain\}$, so the only MRP of P, and its sceptical revised program is $P \cup IR(rain)$, whose $WFSX$ is $\{not\ cold\_water, swimming\}$. This result coincides with the WFS_{Opt} calculated in part I.

Example 16. Recall program P of example 9, whose $WFSX_p$ is:

$$\{c, \neg a, not\ a, not\ b, a, not\ \neg a\}.$$

The supports of $\perp$ result from the union of supports of a and supports of $\neg a$. As the only rule for $\neg a$ is a fact, its only support is $\{\}$. Supports of a are the supports of $not\ b$, and supports of $not\ b$ are the supports of c. Again, as the only rule for c is a fact, its only support is $\{\}$.

Thus the only support of $\perp$ is $\{\}$, and so P is unrevisable.

5 Equivalence between Avoidance and Removal

In this section we discuss the equivalence between the approaches of contradiction avoidance described in part I [1] and contradiction removal described in this part.

The need for semantics more sceptical than $WFSX$ can be seen as showing the inadequacy of the latter for certain problems. The equivalence results show that this is not the case since, by providing a revision process, $WFSX$ can deal with the same problems as the more sceptical semantics WFS_{Opt}, and give the same results.

The advantages of using $WFSX$ plus the revision process reside mainly on its simplicity compared to avoidance, and on the existence of top-down procedures for it. Moreover, as pointed out in [5], the top-down procedures for $WFSX$ can be obtained by simple modifications of procedures for WFS [14, 16, 10, 3]. The specialized revision process corresponding to $\mathcal{POS}$ has been sucessfully applied to a wide variety of classical non-monotonic reasoning domains [9, 11, 13].

The revision procedure can be implemented as a preprocessor of programs[12], and the maintenance of non-contradiction might benefit from existing procedures for Truth Maintenance Systems.

In order to prove the main equivalence theorems, we begin by presenting two important lemmas. These lemmas state that avoiding a hypothesis in contradiction avoidance is equivalent to adding an inhibition rule for that hypothesis in contradiction removal.

Lemma 27. *If $P \cup H$ is a complete scenario wrt Opt of a program P with avoidance set S then $P' \cup H$ is a complete scenario of $P' = P \cup IR(S)$.*

Lemma 28. *If $P' \cup H$ is a complete scenario of $P' = P \cup IR(R)$, and $R \subseteq Opt$, then $P \cup H$ is complete wrt Opt.*

Theorem 29 Quasi-complete scenarios and MNSs. *$P \cup H$ is a quasi-complete scenario wrt Opt of program P with avoidance set S iff*

$$\langle \mathrm{WFSX}(P \cup IR(S)), S \rangle$$

is a MNS of P with revisables Opt.

This theorem states that assuming hypotheses maximally and avoiding the contradiction, corresponds to minimally introducing inhibition rules, and then computing the *WFSX*.

Theorem 30 Sceptical revision and WFS_{Opt}. *$P \cup H$ is the WFS_{Opt} of a program P with avoidance set S iff $\langle \mathrm{WFSX}(P \cup IR(S)), S \rangle$ is the sceptical submodel of P with revisables Opt.*

From this theorem it follows that the role of optatives in contradiction avoidance is the same as the role of revisables in contradiction removal. Thus the discussion about special criteria for automatically inferring optatives from a program, applies directly to the issue of finding special criteria for inferring revisables from the program.

References

1. J. J. Alferes and L. M. Pereira. Contradiction: when avoidance equal removal. Part I. In R. Dyckhoff, editor, *4th Int. Ws. on Extensions of Logic Programming*, 1993.
2. José Júlio Alferes. *Semantics of Logic Programs with Explicit Negation*. PhD thesis, Universidade Nova de Lisboa, October 1993.

[12] The implementation of such a preprocessor for the revision process above exists, and is available on request.

3. W. Chen and D. H. D. Warren. A goal–oriented approach to computing well–founded semantics. In K. Apt, editor, *Int. Joint Conf. and Symp. on Logic Programming*, pages 589–603. MIT Press, 1992.
4. K. Eshghi and R. Kowalski. Abduction compared with negation by failure. In G. Levi and M. Martelli, editors, *6th Int. Conf. on Logic Programming*. MIT Press, 1989.
5. L. M. Pereira and J. J. Alferes. Well founded semantics for logic programs with explicit negation. In B. Neumann, editor, *European Conf. on AI*, pages 102–106. John Wiley & Sons, Ltd, 1992.
6. 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*, pages 105–119. MIT Press, 1991.
7. 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, *5th Portuguese AI Conf.* Springer-Verlag, 1991.
8. L. M. Pereira, J. J. Alferes, and J. N. Aparício. Contradiction removal semantics with explicit negation. In *Applied Logic Conf.* ILLC, Amsterdam, 1992.
9. 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.* MIT Press, 1991.
10. L. M. Pereira, J. N. Aparício, and J. J. Alferes. A derivation procedure for extended stable models. In *Int. Joint Conf. on AI.* Morgan Kaufmann Publishers, 1991.
11. L. M. Pereira, J. N. Aparício, and J. J. Alferes. Logic programming for nonmonotonic reasoning. In *Applied Logic Conf.* ILLC, Amsterdam, 1992.
12. L. M. Pereira and M. Calejo. A framework for Prolog debugging. In R. Kowalski, editor, *5th Int. Conf. on Logic Programming.* MIT Press, 1988.
13. L. M. Pereira, C. Damásio, and J. J. Alferes. Diagnosis and debugging as contradiction removal. In L. M. Pereira and A. Nerode, editors, *2nd International Workshop on Logic Programming and NonMonotonic Reasoning*, pages 316–330. MIT Press, 1993.
14. T. Przymusinski. Every logic program has a natural stratification and an iterated fixed point model. In *8th Symp. on Principles of Database Systems.* ACM SIGACT-SIGMOD, 1989.
15. R. Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32:57–96, 1987.
16. D.S. Warren. The XWAM: A machine that integrates prolog and deductive databases. Technical report, SUNY at Stony Brook, 1989.

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