A paraconsistent semantics detecting contradiction support

Carlos Viegas Damásio[1,2] and Luís Moniz Pereira[2]

[1] Universidade Aberta
Rua da Escola Politécnica 141-147
1250 Lisboa, Portugal
cd@di.fct.unl.pt
[2] CITIA, Departamento de Informática, U. Nova de Lisboa,
2825 Monte da Caparica, Portugal
lmp@di.fct.unl.pt

Abstract. A

1 Introduction

Logic programming still holds great potential and already has shown wide adequateness for representing knowledge [4]. Nevertheless, a mechanism for explicitly declaring the falsity of propositions is not available in the restricted language of normal logic programs. The importance of extending logic programming with an explicit form of non-classical negation, "¬", alongside the usual default or implicit one, has been stressed for use in deductive databases, knowledge representation, and non-monotonic reasoning. This has been pointed out by several authors [25, 16, 19, 37, 27], and an enhanced language and semantics [16, 31, 27, 39] have been proposed. Extended logic programming was thus born. A recent study of this explicit form of negation (and its strong form) compared with classical negation can be found in [?, ?].

However, the introduction of explicit negation permits and requires being able to reason with, or at least detect, contradictory information. Indeed, information is not only normally incomplete but contradictory as well. As remarked in [29, 39] there are three main ways of dealing with inconsistent information in logic programs:

Explosive approach: If the program is contradictory then every formula is derived from it. This corresponds to the usual approach in mathematical logic, and of several of the semantics for extended logic programs [31, 16, 27].

Belief revision approach: The program is revised in order to regain consistency. This is the view adopted by some authors in the logic programming community [29, 28, ?, ?, 1]. It does not necessarily require an explicit paraconsistent semantics: the procedural program transforming revision operators suffice.

Paraconsistent approach: Accept contradictory information into the semantics and perform reasoning tasks that take it into account[6, 25, 26, 23, 24, 38, 39, 1]. This is the approach we will further explore in this paper.

The first approach only makes sense when dealing with mathematical objects. For instance, if we have a large knowledge base being maintained or updated by different agents, it is natural to encounter inconsistencies in the database. Most of the time, this inconsistency is local to some part of the knowledge base and shouldn't affect other, independent, information. But if we adopt the explosive approach, when a single contradiction is found we must discard the entire knowledge base. This is uneconomical.

Sometimes the contradictory information can be due to a specification error, and we'd like to fix it through debugging. In other situations the information provided is in itself contradictory, and is not to be corrected. In the former case, we rely on belief revision techniques. In the latter, a paraconsistent deductive mechanism is necessary. However, even to perform belief revision we need, in any case, to detect inconsistencies and the reasons supporting them. Thus, paraconsistent reasoning is, at least implicitly, an intermediate step to attain belief revision.

The idea of introducing paraconsistent reasoning in logic programming is fairly recent. Since paraconsistent reasoning seems to be fundamental for understanding human cognitive processes, it has been study in philosophical logic by several authors [7, 5, 32, 3, 30]. Their intuitions and results have been brought to the logic programming setting mainly by Blair, Pearce, Subrahmanian, and Wagner [6, 25, 26, 23, 24, 38, 39]. The introduction of a non-classical explicit form of negation in logic programming led other researchers to address this issue as well, namely Przymusinski, Sakama and ourselves, with respect to extensions of well-founded and of answer sets semantics [31, 33, 34, 1, ?].

However, with the exception of few works [33, 34, 41], the problem of detecting support on contradiction has been overlooked or not properly captured. It is also desirable to have some control on the propagation of inconsistency [41, ?]. A major achievement of this paper is to present the techniques for dealing with these issues in an integrated fashion. The starting point is our paraconsistent well-founded semantics with explicit negation ($WFSX_{\mathrm{p}}$) [1, ?].

The plan of the paper is as follows. We start by motivating the use of paraconsistency and the detection of contradiction supported conclusions with recourse to examples. Next we overview $WFSX_{\mathrm{p}}$ and present its embedding into WFS. We address in Section 4 the problem of detecting support on contradiction by relating it to $WFSX_{\mathrm{p}}$ intrinsic properties. Afterwards, we show how to implement two new constructs in the language of extended logic programs in order to gain explicit control of contradiction propagation. We finish by drawing some conclusions.

2 Motivating Examples

Since paraconsistency is present in our everyday life, the rationale of our semantics is to glean the maximum number of conclusions, but not all, from contradictory information. This gives one the information required to decide what to do, because all, even if incompatible, scenarios are considered. Contradictory information alerts us about potential problems, and is up to us to take the right decision. An interesting case in point is that of taxonomies with incomplete information. It is a natural example because our knowledge of the animal world is rather limited, and new species are discovered regularly:

Example 1. Consider the following simple-minded common-sensical rules for identifying birds and mammals:

- Oviparous warm-blooded animals having a bill are birds;
- Hairy warm-blooded animals are mammals;
- Birds are not mammals and vice-versa;
- Birds fly;
- Mammals nurse their offspring.

This chunk of knowledge can be represented by the following extended logic program rules:

$$bird(X) \leftarrow bill(X), warm\_blood(X), oviparous(X).$$
$$mammal(X) \leftarrow hair(X), warm\_blood(X).$$
$$\neg mammal(X) \leftarrow bird(X). \qquad flies(X) \leftarrow bird(X).$$
$$\neg bird(X) \leftarrow mammal(X). \qquad nurses(X) \leftarrow mammal(X).$$

If the information regarding cats and ducks is correctly filled in one gets the expected results. We just add to the program the set of facts, with the obvious abbreviations:

$$hair(c) \qquad warm\_blood(c). \qquad bill(d). \qquad warm\_blood(d). \qquad oviparous(d).$$

Relevant conclusions are:

$$\{mammal(c), nurses(c), \neg bird(c), not\, bird(c), not\, flies(c)\} \cup$$
$$\{bird(d), flies(d), \neg mammal(d), not\, mammal(d), not\, nurses(d)\}$$

In your trip to Australia you now discover there are creatures named platypus which lay eggs, have warm-blood, sport a bill, and are hairy! You obtain stupendous contradictions from the program containing the facts:

$$hair(p). \qquad warm\_blood(p). \qquad bill(p). \qquad oviparous(p).$$

The model entails the following new conclusions:

$$\{mammal(p), \neg mammal(p), not\, mammal(p), not\, \neg mammal(p)\} \cup$$
$$\{nurses(p), not\, nurses(p), not\, \neg nurses(p)\} \cup$$
$$\{bird(p), \neg bird(p), not\, bird(p), not\, \neg bird(p))\} \cup$$
$$\{flies(p), not\, flies(p), not\, \neg flies(p)\}$$

The remarkable points about this example are manifold. First, contradictory information can coexist with safe one without interfering with each other; in particular, we must not relinquish the information about dogs and ducks. Second, we can detect a contradiction on both the *mammal* and the *bird* predicates and, since its consequences are propagated we are aware that the knowledge about platypus regarding their nursing and flying abilities is doubtful because supported on contradiction. Third, the correct information is covered by the program's model: platypus are mammals, do not fly, and nurse their progeny. Finally, it is unsound to have a heuristic rule to the effect of dropping all objective (or default) knowledge regarding platypus. We want to retain that they nurse their descendants but discard the $fly(p)$ conclusion. Compare with the other semantics, which insist that platypus fly and nurse they children.

The above taxonomy example clearly identifies another desirable advantage and motivation for our semantics. Assume that the above knowledge is to be revised in order to consistently incorporate the new information regarding platypus. By having all contradiction supported conclusions initially available we can mark the desirable and undesirable ones, program revision (or declarative debugging if you prefer) can then be performed and problematic rule instances pinpointed, namely, in this example

$$bird(p) \leftarrow bill(p), warm\_blood(p), oviparous(p)$$

We have indeed devised such declarative debugging techniques for logic programs on the basis of our paraconsistent semantics and contradiction removal methods. The reader is referred to [?, ?, ?] for the definitions and techniques. It is manifest from this discussion that conclusions supported on contradiction are desirable and should not be avoided or discarded, as they contain the necessary information for identifying and fixing the problem.

Another use of paraconsistency, specially catered by our semantics, is to evaluate the pros and cons of two possible situations, and take the appropriate measures for each of the contingencies. This is particularly relevant when representing medical knowledge.

Example 2. A patient arrives at a hospital urgency with the following symptoms and signs[3]:

- Sudden epigastric pain;
- Abdominal tenderness;
- Signs of peritoneal irritation.

These symptoms are common to a perforation of a peptic ulcer and to an acute pancreatitis. The former requires surgery and the latter solely therapeutic treatment. The (approximate) rules for diagnosing these two conditions are:

- If a patient has sudden epigastric pain, abdominal tenderness, and signs of peritoneal irritation then he has a perforation of a peptic ulcer or an acute pancreatitis[4];

[3] We thank Doctor of Medicine António Dias Ribeiro for providing us this example. Any mistakes are entirely our own.

[4] He can also has a myocardial infarction, but for the sake of simplicity we ignore this situation.

– Furthermore, if he has high amylase levels then a perforation of a peptic ulcer can be exonerated; vice-versa, if he exhibits Jobert's manifestation then pancreatitis can be exonerated;
– Under either situation the patient should not be nourished, and should take H_2 antagonists.

We represent this scenario with the following extended logic program:

$$perforation \leftarrow sudden\_pain, abd\_tenderness, per\_irritation, not\, high\_amylase$$
$$pancreatitis \leftarrow sudden\_pain, abd\_tenderness, per\_irritation, not\, jobert.$$

$$surgery\_indication \leftarrow perforation. \qquad \neg surgery\_indication \leftarrow pancreatitis.$$
$$anesthesia \leftarrow surgery\_indication.$$

$$\neg nourish \leftarrow perforation. \qquad H_2\_antagonist \leftarrow perforation.$$
$$\neg nourish \leftarrow pancreatitis. \qquad H_2\_antagonist \leftarrow pancreatitis.$$

Now assume that the physician is uncertain regarding the amylase levels and existence of Jobert's manifestation. Cautiously, he adds the information $\neg high\_amylase$ and $\neg jobert$ to the system, besides the other known symptoms $sudden\_pain$, $abd\_tenderness$, and $per\_irritation$.

He can conclude that the patient suffers from pancreatitis and perforation of a peptic ulcer. In either case, the sick person should not be fed and should take H_2 antagonists. Anesthesia is recommended but relies on contradictory information, namely $surgery\_indication$ and $\neg surgery\_indication$ (and therefore $not\, anesthesia$ is also entailed). In such a problematic cases the patient is usually operated on, regardless.

An additional motivation for our approach, and an important one, is that we want to execute queries in a goal-oriented fashion. That being the case, first it is not desirable to pay an extra price for having paraconsistency. This is indeed possible: our procedures are the same for the paraconsistent and for the non-paraconsistent cases [?]. Second, since we are evaluating queries in a goal-oriented fashion we do not want to test and check consistency of the entire knowledge base: contradictions are dealt with at "run-time" instead of at "compile-time", with the advantage of simplifying knowledge-base updating. If global consistency is to be enforced, then allowing paraconsistency helps us to identify the reasons behind non-intended conclusions. Restoring consistency can then be carried out in a computationally more demanding second stage (contradiction removal). Furthermore, since a precise and declarative meaning of programs is provided by $WFSX_{\mathrm{p}}$, one should not be reluctant in having consequences and conclusions based on contradiction, as long as they can be distinguished. Moreover, as we have noted, keeping contradiction can be important for knowledge-representation because it proffers important information about contradiction supported conclusions. Our semantics is not an *ad hoc* approach to paraconsistency, but respects tha above caveats.

3 Overview of $WFSX_{\mathrm{p}}$

As usual, for the sake of simplicity and without loss of generality, we will restrict the discussion to (possibly infinite) propositional, or to ground, programs. A non-ground program stands for its fully instantiated version, i.e. the set of all ground instances of its rules. The alphabet of the language of the programs is the set of atoms At. Atoms can be negated by juxtaposing them with the explicit negation symbol "$\neg$" thereby obtaining the explicitly negated literals. The explicit complement of a set $A = \{a_1, a_2, \ldots\}$ is $\neg A = \{\neg a_1, \neg a_2, \ldots\}$. The set of objective literals is $OLit = At \cup \neg At$. Default literals are of the form $not\, a$ and $not\, \neg a$, where a is an atomic proposition in the language's alphabet. The set of all literals is $Lit = OLit \cup not\, OLit$, where default negation of a set of objective literals stands for the set comprised of the default negation of each one.

Definition 1. An extended logic program is a set of rules of the form

$$L_0 \leftarrow L_1, \ldots, L_m, not\, M_1, \ldots, not\, M_n \qquad (m, n \geq 0)$$

where L_i $(0 \leq i \leq m)$ and M_j $(1 \leq j \leq n)$ are objective literals.

Most semantics for extended logic programs view default negation and explicit negation as unrelated. To overcome this situation a new semantics for extended logic programs was proposed in [27, 2]. Well-founded Semantics with Explicit Negation (WFSX for short) embeds a "coherence principle" providing the natural missing link between both negations: if $\neg L$ holds then $not\, L$ should too (similarly, if L then $not\, \neg L$). More recently, a paraconsistent extension of this semantics ($WFSX_{\mathrm{p}}$) has been proposed in [1, ?] via an alternating fixpoint definition that we now recapitulate.

We begin by recalling the definition of Gelfond-Lifschitz Γ operator, used in the alternating fixpoint definition of WFS [14], $WFSX$, and $WFSX_{\mathrm{p}}$.

Definition 2. [15] Let P be an extended program, I a two-valued interpretation. The GL–transformation $\frac{P}{I}$ is the 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 $\Gamma_P I = \mathcal{M}_{\frac{P}{I}}$.

To impose the coherence requirement Alferes and Pereira resort to the semi-normal version of an extended logic program.

Definition 3. [27, 2] 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 with respect to 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 4. [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. assigns meaning to every extended logic program.

Definition 5. [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 and Pereira 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 true because, by coherence, they would also entail $not\,L$ and $not\,\neg L$. Therefore $WFSX_{\mathrm{p}}$ generalizes $WFSX$, as it does not impose this additional condition. Furthermore, for normal logic programs, it coincides with WFS.

Example 3. Let P be the extended logic program:

$$a \leftarrow not\,b. \qquad b \leftarrow not\,c. \qquad b \leftarrow not\,\neg c. \qquad c. \qquad \neg c. \qquad d \leftarrow not\,d.$$

The sequence for determining the least fixpoint of $\Gamma\Gamma_s$ of program P is:

$$\begin{aligned}
I_0 &= \{\} \\
I_1 &= \Gamma\Gamma_s\{\} &&= \Gamma\{a, b, c, \neg c, d\} &&= \{c, \neg c\} \\
I_2 &= \Gamma\Gamma_s\{c, \neg c\} &&= \Gamma\{a, d\} &&= \{a, b, c, \neg c\} \\
I_3 &= \Gamma\Gamma_s\{a, b, c, \neg c\} &&= \Gamma\{d\} &&= I_2
\end{aligned}$$

Thus $WFM_p(P) = \{a, b, c, \neg c, not\,a, not\,\neg a, not\,b, not\,\neg b, not\,c, not\,\neg c, not\,\neg d\}$.

One of the distinguishing features of $WFSX_{\mathrm{p}}$ is that it does not enforce default consistency, i.e. a and $not\,a$ can be simultaneously true, in contradistinction to all other semantics. In the above example this is the case for literals a, b, c and $\neg c$. It is due to the adoption of the coherence principle: for instance, because a and $\neg a$ hold then, by coherence, $not\,\neg a$ and $not\,a$ must hold too. We will see in the next section how can one profit from this seemingly awkward behaviour.

Next we consider a useful program transformation from $WFSX_{\mathrm{p}}$ to normal programs under WFS semantics which preserves the paraconsistent well-founded model, though not the other PSM_ps. This is achieved at the cost of doubling the program size and computation. The main idea is to use a variant of the doubled program construction of [9, 40], which doubles the number of rules: one takes care of computing the Γ_s part and the other of the Γ part:

Definition 6. Let P be an extended logic program. Normal logic program P^{T-TU} is derived from P by replacing each rule with two new ones obtained from it. First, every atom A ($\neg A$) is replaced by A^p (A^n) and:

- In the first new rule all default negated occurences of objective literals are subscripted with TU, and all other objective literals with T;
- In the second new rule all default negated occurences of objective literals are subscripted with T and all other objective literals with TU. Furthermore, if A^p (A^n) is the head of the new rule then $not\, A^n$ $(not\, A^p)$ is added to its body.

The difference to the doubled program construction is in the introduction of the extra default literal in the body of rules of the second rule form. Note the similarities of the second form with the semi-normal program for P. The two subscript distinct program parts are computationally linked via the program divisions on default negated literals.

Example 4. Let P be the extended logic program:

$$a \leftarrow \neg c, not\, b. \qquad b \leftarrow \neg a, b, not\, c, not\, \neg b. \qquad \neg a.$$

The resulting $T - TU$ transformed program is:

$$\begin{array}{ll} a^p_T \leftarrow c^n_T, not\, b^p_{TU}. & a^p_{TU} \leftarrow c^n_{TU}, not\, b^p_T, not\, a^n_T. \\ b^p_T \leftarrow a^n_T, b^p_T, not\, c^p_{TU}, not\, b^n_{TU}. & b^p_{TU} \leftarrow a^n_{TU}, b^p_{TU}, not\, c^p_T, not\, b^n_T, not\, b^n_T. \\ a^n_T. & a^n_{TU} \leftarrow not\, a^p_T. \end{array}$$

Rules for literals subscripted with TU are used to derive the literals obtained at the Γ_s steps, and the ones subscripted with T the ones obtained at the $\Gamma\Gamma_s$ steps. With this idea in mind the following result is more or less straightforward:

Theorem 7. *Let P be an extended logic program. Then the following equivalence holds for an arbitrary atom a:*
- $a \in WFM_{\mathrm{p}}(P)$ *iff* $a^p_T \in WFM(P^{T-TU})$;
- $\neg a \in WFM_{\mathrm{p}}(P)$ *iff* $a^n_T \in WFM(P^{T-TU})$;
- $not\, a \in WFM_{\mathrm{p}}(P)$ *iff* $not\, a^p_{TU} \in WFM(P^{T-TU})$;
- $not\, \neg a \in WFM_{\mathrm{p}}(P)$ *iff* $not\, a^n_{TU} \in WFM(P^{T-TU})$.

To conclude this overview of $WFSX_{\mathrm{p}}$, we have proven elsewhere [?] that appropriate extensions of Dix's strong and weak properties for the paraconsistent version are all obeyed by $WFSX_{\mathrm{p}}$. All strong principles, with the exception of *Cautious Monotony*, are verified by $WFSX_{\mathrm{p}}^{\mathrm{g}}$. However, two natural new versions of *Cautious Monotony* for general logic programs were set forth by us, and we have proven $WFSX_{\mathrm{p}}$ satisfies them. The weak principles were devised especially for normal logic programming semantics, and therefore it is natural that with an extended language they have to be modified to reflect the more general framework. With the exception of *Modularity*, *Equivalence*, *Isomorphy*, *Weak model property* and *M_P-extension*, all other properties had to be generalized for the paraconsistent extended logic programming case. All such new properties are verified by $WFSX_{\mathrm{p}}$ [?].

4 Detecting Support On Contradiction

One main advantage of $WFSX_{\mathrm{p}}$ with respect to other paraconsistent semantics is in surpassing the others limited facility in detecting the support of conclusions on contradiction. Our first effort is to clarify what we mean by support on contradiction, something which is most of the times neglected. Our definition is declarative and independent of the underlying semantics. We only assume the language of extended logic programs.

Definition 8. Let P be an extended logic program and $C = \{c, \neg c \mid c, \neg c \in P\}$, the set of the contradictory facts in P. We say that a literal L depends on contradiction iff there is $S \subseteq C$ such that

$$SEM(P) \cap \{L, \neg L, not\, L, not\, \neg L\} \neq SEM(P - S) \cap \{L, \neg L, not\, L, not\, \neg L\}$$

The rationale is clear-cut. We say that L depends on contradiction iff by removing some subset of the contradictory literals in the program the truth-value of L gets changed. The main result regarding $WFSX_{\mathrm{p}}$ is afforded by the next theorem:

Theorem 9. *Let P be an extended logic program with paraconsistent well-founded model M such that for every objective literal L we have at least $L \in M$ or $not\, L \in M$. If there is a rule with true body for a literal L and L depends on contradiction then both L and $not\, L$ are in M.*

Dependence is also an operational concept, related to the underlying proof procedures for $WFSX_{\mathrm{p}}$ described in [1, ?]. Ignoring default negation for the sake of simplicity, the idea expressed in Theorem 9 is that if L is true and all ways of proving L (i.e. derivations for L) use at least one literal L' for which L' and $\neg L'$ hold. For details consult [?].

The converse of Theorem 9 is also true, but more generally so: if L and $not\, L$ belong to the model (total or otherwise) then L depends on contradictory information, i.e. L is derivable from inconsistent antecedents.

The above theorem justifies why in some situations we have a literal and its default negation belonging simultaneously to the program's model. Contrarily to what might be expected this conveys very important information.

Example 5. Consider the program:

$$\begin{array}{lll} innocent \leftarrow \neg guilty. & charged. & man. \\ \neg guilty \leftarrow charged, not\, guilty. & \neg charged. & \end{array}$$

It has the following paraconsistent well-founded model:

$$\begin{array}{c} \{charged, \neg charged, \neg guilty, innocent, man\} \cup \\ not\, \{charged, \neg charged, guilty, \neg guilty, innocent, \neg innocent, \neg man\} \end{array}$$

For this program, objective literal conclusions in our semantics coincide with the usual ones provided by several semantics. However, if we look at the default negated literals, we notice that $not\, innocent$ and $not\, \neg guilty$ are true. Though $innocent$ and $\neg guilty$ are true but $\neg innocent$ and $guilty$ are not entailed, we nevertheless know that $innocent$ and $\neg guilty$ depend on contradictory information, even if not themselves contradictory.

A similar interpretation applies to the examples of Section 2.

Sakama and Inoue provide two semantics for detecting support on contradiction [33, 34]. However they are not able to propagate the knowledge about the support on contradiction through negation by default, as the following example shows:

Example 6. Recall the model of the program of Example 3:

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

This means that c and $\neg c$ are contradictory and that a and b depend on contradictory information. Literal d is undefined. Clearly the truth of a depends on the contradictory pair of literals c and $\neg c$: by removing at least one of them a is no longer true. Sakama's suspicious well-founded model [33] and Sakama and Inoue's suspicious stable models [34] are incapable of detecting this dependency of a.

Another approach to the detection of support on contradiction can be found in [41]. As for the previous semantics, these authors are not capable of dealing with the propagation of contradiction support through default negation.

Example 7. Consider the following paraconsistent logic program:

$$a \leftarrow not\, b. \qquad b \leftarrow not\, c, not\, \neg c. \qquad c. \qquad \neg c.$$

This program has the well-founded structure $< \{b, c, \neg c\}, \{\{b, c\}, \{b, \neg c\}\} >$. Accordingly we have $not\, a$, $not\, c$ and $not\, \neg c$ true in that structure. One of the interesting points of this semantics is that neither c nor $\neg c$ hold in the above program. Now, suppose we remove one of c or $\neg c$. In the resulting programs the associated well-founded structures we have $not\, b$ and a true. This means that the conclusion $not\, a$ in the original program depends on the contradictory literals c and $\neg c$, as per Definition 8. However, the approach followed in [41] is not capable of detecting this situation. For more details consult [41].

Our semantics is able to detect the support on contradictory information in the above examples since the model is total, as this is guaranteed by Theorem 9. In some non-total situations, it might happen that the truth of a literal depends on contradictory information, but $WFSX_{\mathrm{p}}$ is not able to detect it, at least via Theorem 9. This can occur only if there are undefined literals in the model. Theorem 12 below addresses this situation, so that $WFSX_{\mathrm{p}}$ covers all the cases. Let us see what the problem is.

Example 8. Consider the extended logic program:

$$a \leftarrow not\, b. \qquad a \leftarrow c. \qquad b \leftarrow not\, b. \qquad c. \qquad \neg c.$$

In this program the truth of a depends on the contradictory fact $\neg c$. But $not\, a$ is not entailed by the WFM_p model of the program, since b is undefined. Sakama's suspicius well-founded semantics handles this program properly.

We need to equip $WFSX_{\mathrm{p}}$ semantics with additional mechanisms to handle the non-total cases. Failure to detect support on contradiction by Theorem 9 for non-total models always hinges on some literal true in the model but with at least one rule with undefined body. This remark motivates the following definition:

Definition 10. Let P be an extended logic program and M its WFM_p. Program ΔP is obtained from P by deleting all the rules with a true head and an undefined body. A rule has an undefined body iff there is a literal in the body that does not belong to M, and for each body literal its default complement does not belong to M.

Now, such rules can be removed without shrinking the semantics. Moreover, whenever P is a normal logic program or an extended one with consistent well-founded model we have $WFM_p(P) = WFM_p(\Delta P)$. This property does not hold for contradictory extended logic programs:

Example 9. Consider the program of Example 8. The corresponding Δ program is obtained by deleting the rule $a \leftarrow not\, b$. The well-founded model of the transformed program is $\{a, not\, a, not\, \neg a, not\, \neg b, c, \neg c, not\, c, not\, \neg c\}$, which is distinct from the original program.

However, one can easily show that in general we have $\Delta P \subseteq P$ and $WFM_p(P) \subseteq WFM_p(\Delta P)$, i.e. as the program monotonically reduces the corresponding model monotonically increases.

The idea is to apply the Δ operator iteratively till the least fixpoint is reached. In each step the undefined rules which can prevent the propagation of the contradiction support are removed. At the end one obtains the desired property of support on contradiction detection, since all rules which could influence the application of Theorem 9 have been removed.

Definition 11. Let P be an extended logic program. Construct the following transfinite sequence of programs $\{P_\alpha\}$:

$$\begin{aligned} P_0 &= P \\ P_{\alpha+1} &= \Delta P_\alpha \\ P_\delta &= \bigcap \{P_\alpha \mid \alpha < \delta\} \text{ for limit ordinal } \delta \end{aligned}$$

There exists a smallest ordinal λ for the sequence above, such that P_λ is the least fixpoint of Δ. We can now define $WFM_p^s(P) = WFM_p(P_\lambda)$, and the main result is readily obtained:

Theorem 12. *Let P be an extended logic program and let $M = WFM_p^s(P)$. A literal L and its default complement $not\,L$ are in M iff there is a rule with true body for L and L depends on contradictions.*

Example 10. The sequence originated by program P of Example 8 is:

$$\begin{array}{ll} P_0 = P & WFM_p(P_0) = \{a, not\,\neg a, not\,\neg b, c, \neg c, not\,c, not\,\neg c\} \\ P_1 = P_0 - \{a \leftarrow not\,b\} & WFM_p(P_1) = WFM_p(P_0) \cup \{not\,a\} \\ P_2 = P_1 & WFM_p(P_2) = WFM_p(P_1) = WFM_p^s(P) \end{array}$$

An important remark is that the construction of Definition 11 is not continuous. As a result, more than ω steps may be necessary to obtain the least fixpoint. This behaviour is illustrated in Example 11.

Example 11. Let P be the following extended logic program, where variables range over the natural numbers:

$$\begin{array}{llllll} p(0) \leftarrow c. & p(X+1) \leftarrow not\,p(X). & s \leftarrow not\,r. & r \leftarrow p(X). & c. & u \leftarrow not\,u. \\ p(0) \leftarrow u. & p(X+1) \leftarrow u. & s \leftarrow u. & & \neg c & \neg u \leftarrow not\,\neg u \end{array}$$

The tranfinite sequence of programs and associated models $M_i = WFM_p(P_i)$ is:

$$\begin{array}{llll} P_0 & = P & M_0 & = \{p(0), not\,\neg p(0), c, \neg c, not\,c, not\,\neg c, r, not\,\neg r\} \\ P_1 & = P_0 - \{p(0) \leftarrow u\} & M_1 & = M_0 \cup \{not\,p(0), p(1), not\,\neg p(1)\} \\ P_2 & = P_1 - \{p(1) \leftarrow u\} & M_2 & = M_1 \cup \{not\,p(1), p(2), not\,\neg p(2)\} \\ & \vdots & & \vdots \\ P_\omega & = P_0 - \{p(\_) \leftarrow u\} & M_\omega & = \{p(\_), not\,p(\_), not\,\neg p(\_)\} \cup \\ & & & \{c, \neg c, not\,c, not\,\neg c, r, not\,r, not\,\neg r, s, not\,\neg s\} \\ P_{\omega+1} & = P_\omega - \{s \leftarrow u\} & M_{\omega+1} & = M_\omega \cup \{not\,s\} \end{array}$$

The $WFM_p^s(P)$ is $\{p(\_), not\,p(\_), not\,\neg p(\_)\} \cup \{c, \neg c, not\,c, not\,\neg c, r, not\,r, not\,\neg r, s, not\,\neg s, not\,s\}$,. Mark that all literals are simultaneously true and false by default. According to Theorem 12 this means that We conclude that all literals depend on some contradictory literal. Since c is the single contradictory literal, all other literals depend on c.

5 Blocking Propagation of Contradiction

In some situations it is desirable to detect contradictions and avoid any propagation of inconsistency based on them. For instance, in our medical example of Section 2 a physician when faced with the contradiction $\neg surgery\_indication$ and $surgery\_indication$ he opts for the latter. However, our language of extended logic programs does not provide any form of reasoning based on queries "Does x depend on some contradiction ?" and "Is x contradictory ?".

These two kinds of reasoning have been proposed in [41]. This three-valued framework for paraconsistent logic programs includes explicit negation, default negation, and disjunction. Two introspective modal operators $\mathbf{C}$ and $\mathbf{C_d}$ are also provided, which can be employed to detect, respectively, whether some literal is inconsistent on its own or whether, alternatively, it is entailed by inconsistent premises. We have shown in Example 7 that the semantics doe not handle well programs with default negation. Another difference to our approach is that contradictory literals and literals which depend on them are false by default. One has to use the modalities to check if some literal is contradictory or depends on contradiction. This means that contradiction is regarded harmful and turns difficult the reading of logic programs (one cannot immediately see if some literal is entailed or not). Last but not least, the authors discard the coherence principle.

A different approach has been proposed in [?], where two modal operators $\mathbf{B}$ and $\mathbf{B^C}$ are included to represent, respectively, possibly inconsistent beliefs or just consistent ones. A first-order language extended with the modal operators is used. The author is concerened with the axiomatization of these operators and applies them to some knowledge representation problems.

We adopt the ideas of [41] in our semantics, including the terminology, and we will show how these two operators can be implemented in $WFSX_p$ by means of a program transformation into WFS, without recourse to modalities.

Example 12. [41] A federal regulation requires the floor of a sausage plant to be kept clean. The only way to satisfy this is by constantly washing the floor. A state regulation requires the floor of a sausage plant not be wet. Violations will be fined. This situation is encoded with the following extended logic program:

$$penalty \leftarrow watering. \qquad penalty \leftarrow \neg watering. \qquad watering. \qquad \neg watering.$$

According to the authors, the most appropriate information regarding *penalty* is that *penalty* is depends on inconsistent premises, so that one may want to investigate and appeal the penalty. To achieve this, the rule "$appeal \leftarrow \mathbf{C_d} penalty$" is introduced in the program. Then, of course, *appeal* and $\mathbf{C}watering$ hold. Mark, as we have noticed before, that neither *watering* nor $\neg watering$ nor *penalty* are entailed.

Definition 13. Instropective programs are obtained by augmenting the language of extended logic programs with the *introspective operators* $\mathbf{C}L$ and $\mathbf{C_d}L$, where L is any objective literal. The occurence of introspective operators is allowed only in the body of rules. Default negations of these operators are also permitted.

In the previous section we have identified the conditions for detection of a contradictory conclusion and also of dependence on contradiction. Recall that to test if an objective literal L is contradictory it suffices to check whether L and $\neg L$ belong to the program's model; to test if L is supported on contradiction it is enough to check that L and $not\, L$ are both entailed. We will first simply equate $\mathbf{C}L$ with "L and $\neg L$ hold" and $\mathbf{C_d}$ with "L and $not\, L$ hold." This is a meta-condition rather difficult to implement. Our approach to the problem is to make use of the P^{T-TU} program transformation to capture the intended meaning of the introspective operators. However a completely satisfactory solution is not attained, except for total models. We will discuss the problems with our solution further on in this section.

Definition 14. Let P be an introspective program. Normal logic program P^{int} is derived from P by replacing each rule with two new ones obtained from it. Every occurence of an introspective operator is kept as in the original program. The remaining translation of rules is identical to the P^{T-TU} transformation.

Additionally, the following rules are added to P^{int}:

- If $\mathbf{C}A$ occurs in the body of some rule or query then include $\mathbf{C}A \leftarrow A_T^p, A_T^n$;
- If $\mathbf{C}\neg A$ occurs in the body of some rule or query then include $\mathbf{C}\neg A \leftarrow A_T^p, A_T^n$;
- If $\mathbf{C_d}A$ occurs in the body of some rule or query then include $\mathbf{C_d}A \leftarrow A_T^p, not\, A_{TU}^p$;
- If $\mathbf{C_d}\neg A$ occurs in the body of some rule or query then include $\mathbf{C_d}\neg A \leftarrow A_T^n, not\, A_{TU}^n$.

For all purposes, literals of the form $\mathbf{C}L$ and $\mathbf{C_d}L$ in P^{int} are regarded as new atoms.

First, remark that the evaluation of the introspective operators in T and TU rules is identical. This accomplishes the desired blocking of propagation of inconsistencies. An introspective operator holds in a T rule iff it holds in a TU rule. Second, the bodies of the rules which implement the new operators capture the two conditions previously discussed. The semantics of the introspective programs is obtained straightforwardly.

Definition 15. Let P be an introspective program genreating the normal logic program P^{int}. The introspective well-founded model of P designated by $WFM_i(P)$, is obtained from the well-founded model of P^{int} as follows:

- $a \in WFM_i(P)$ iff $a_T^p \in WFM(P^{int})$;
- $\neg a \in WFM_i(P)$ iff $a_T^n \in WFM(P^{int})$;
- $not\, a \in WFM_i(P)$ iff $not\, a_{TU}^p \in WFM(P^{int})$;
- $not\, \neg a \in WFM_i(P)$ iff $not\, a_{TU}^n \in WFM(P^{int})$.

The truth (falsity) of an introspective operator can be tested by checking if it (its default negation) belongs to $WFM(P^{int})$.

Example 13. Consider the program of Example 12. Its associated transformed program is (with the obvious abbreviations):

$$
\begin{array}{lll}
p_T^p \leftarrow w_T^p. & w_T^p. & a_T^p \leftarrow \mathbf{C_d}p. \\
p_{TU}^p \leftarrow w_{TU}^p, not\, p_T^n. & w_{TU}^p \leftarrow not\, w_T^n. & a_{TU}^p \leftarrow \mathbf{C_d}p, not\, a_T^n. \\
 & & \\
p_T^p \leftarrow w_T^n. & w_T^n. & \mathbf{C_d}p \leftarrow p_T^p, not\, p_{TU}^p. \\
p_{TU}^p \leftarrow w_{TU}^n, not\, p_T^n. & w_{TU}^n \leftarrow not\, w_T^p. & \mathbf{C}w \leftarrow w_T^p, w_T^n.
\end{array}
$$

The well-founded model of the above normal logic program is:

$$\{p_T^p, w_T^p, w_T^n, a_T^p, a_{TU}^p, \mathbf{C_d}p, \mathbf{C}w\} \cup not\, \{p_{TU}^p, p_T^n, p_{TU}^n, w_{TU}^p, w_{TU}^n, a_T^n, a_{TU}^n\}$$

Therefore, we obtain the following model for the original program:

$$\{p, w, \neg w, a\} \cup not\,\{p, \neg p, w, \neg w, \neg a\}$$

Notice the differences to the resulting model of [41] presented in Example 12. The literals are derived irrespectively of whether they are contradictiory (or depend on contradiction) or not. Nevertheless, the information regarding the problematic literals is provided by means of the introspective operators. These operators can then be used to detect the contradiction and block its propagation whenever desired. By blocking the contradiction we mean that a and $not\,a$ are not simultaneously true. If we would have substituted $\mathbf{C_d}p$ by p in the rule for a then a and $not\,a$ would be entailed. By introducing the introspective operator this is no longer the case: we have retained a solely.

Even though the introspective operators inherently have a two-valued semantics it is possible in some cases to have these operators undefined in the introspective well-founded model. The same behaviour can occur for the original $\mathbf{C}$ and $\mathbf{C_d}$ operators of [41]. This feature is illustrated in the next example.

Example 14. Let P be the introspective program containing the fact $\neg a$ and the rule $a \leftarrow not\,\mathbf{C}a$. The introspective well-founded model of P is simply $\{\neg a, not\,a\}$. The $\mathbf{C}a$ operator gets undefined in the auxiliar P^{int} program; any other value leads to a contradiction: if $\mathbf{C}a$ were true then the body of the rule for a would be false, and therefore there would not be any support for the truth of a, and hence $\mathbf{C}_a$ would be false; if $\mathbf{C}a$ were false then a would be entailed and therefore $\mathbf{C}a$ would be true.

Some combinations of the primitive introspective operators are useful for knowledge representation. The reading of the negations of the new operators is immediate: $not\,\mathbf{C}L$ means that L is not contradictory, though it might depend on contradiction; $not\,\mathbf{C_d}L$ simply means that L is safe, i.e. neither is it contradictory nor does it depends on contradiction. The conjunction $not\,\mathbf{C}L \wedge \mathbf{C_d}L$ means that L is not contradictory but depends on contradictory premises. Finally, the disjunction $\mathbf{C}L \vee not\,\mathbf{C_d}L$ means that either L is contradictory or it is safe, thereby discarding the case of support of L on contradiction.

For those programs where it is needed the construction of Definition 11 to determine the dependencies on contradiction, the results might be unexpected: the integration of the introspective operators in our language destroys the monotonicity of the construction of the previous section. The transfinite sequence of programs as per Definition 11 monotonically decreases when we substitute the WFM_p by WFM_i in the Δ operator. However, we have no more the corresponding inclusion $WFM_i(\Delta P) \subseteq WFM_i(P)$. For the problems involved see the next example.

Example 15. Consider the introspective program P:

$$\begin{array}{lllll} a \leftarrow not\,\mathbf{C_d}b. & b \leftarrow not\,c. & c \leftarrow d. & d. & u \leftarrow not\,u. \\ a \leftarrow u. & & c \leftarrow u. & \neg d. & \neg u \leftarrow not\,\neg u. \end{array}$$

The tranfinite sequence of programs and associated models $M_i = WFM_i(P_i)$ is:

$$P_0 = P \qquad M_0 = \{a, c, d, \neg d\} \cup not\,\{\neg a, b, \neg b, \neg c, d, \neg d\}$$
$$P_1 = P_0 - \{a \leftarrow u, c \leftarrow u\} \quad M_1 = \{b, c, d, \neg d\} \cup not\,\{a, \neg a, b, \neg b, c, \neg c, d, \neg d\}$$

We have $not\,a$ true, which is not expected in the original program. Literal a should be undefined.

A better mechanism for propagating consistency upwards is required for introspective programs. Nevertheless, when the introspective well-founded model is total, i.e. there are no literals and no introspective operators undefined, the intended results are obtained. This is always the case when there are no default negations in the program, as secured by Theorem 9.

For programs without default negation we conjecture a relation to the construction of [41]. Basically, all literals which are not contradictory neither depend on contradiction coincide in both approaches. The same applies to the introspective operators. For the remaining literals, we have them entailed in our paraconsistent well-founded model while in the minimal para-model of [41] they are not entailed. If this conjecture is true, we have given a way of obtaining the minimal para-model of a paraconsistent program.

The integration of the techniques presented here and in the previous section are the subject of our current research. This is an open problem in need of attention. We may proceed by defining an appropriate notion of stratification with respect to the $\mathbf{C}$ and $\mathbf{C_d}$ operators. At least for these programs we can guarantee the expected results.

6 Comparisons and Conclusions

The study of extended logic programming can be traced back to the works of Blair and Subrhamanian [6] on paraconsistent logic programs. The introduction of default negation and introspective operators is not addressed. Several equivalent accounts have been proposed by Wagner [38, 39] and have its own roots in the works on paraconsistent constructive logics [22, 3]. In fact, most semantics seem to coincide, apart syntactical differences, with the class of Generalized Horn Programs [6].

A different line of work has been followed by Fitting. In [11] he provides a Kripke/Kleene semantics for logic programs under a richer range of logics than the classical by resorting to bilattices. This work was continued in [12] for a similar generalization of well-founded semantics. However, the models assigned to logic programs under both semantics are always consistent. Therefore, there is no need of introspective operators to be introduced in the language.

With the introduction of default negation a Pandora's box opens. Two different approaches are identifiable: the coherence view and the weak negation view. The former adopts the coherence principle [27], relating the two forms of negation, as basic and provides a localized "explosion" of consequences when faced with contradiction; our $WFSX_{\mathrm{p}}$ is the only representative of this "school"; the latter view sustain, more or less firmly, the total independence of an atom A from its explicit negation $\neg A$, and all other semantics embrace it, sometimes blindly. The coherence principle also appears disguisedly in logic LA [13].

Wagner's liberal reasoning is the basic inference relation for the weak negationist view, with the exception of his credulous, conservative, and skeptical forms of reasoning. The differences lie, as for ordinary normal logic programming semantics, in the treatment of infinite negative recursions. The well-founded based semantics [31, 33] assign the logical value *undefined* to literals involved in such recursions. Przymusinski's semantics has an explosive behaviour in face of contradiction, while Sakama's extended well-founded semantics does not.

Those of the weak negation persuasion have proposed several non-explosive semantics [23, 24, 34] which are variants of the Answer Sets semantics [16]. Weak Answer Sets and Paraconsistent Stable Models are related to Almukdad and Nelson's logic system N^- [3] as shown in [24]. To overcome the problem of non-existing models for some extended logic programs Sakama defined the Semi-Stable Models semantics [34], but it suffers from problems in the treatment of "*undefined*" literals.

All the referred semantics do not cater for the problem of detecting support on contradiction, and therefore ignore introspective operators. These new features were first touched by Sakama and Inoue, for paraconsistent extensions of well-founded semantics [33] and answer-sets [34]. As we have shown their notions of support do not cope well with default negation. Furthermore, introspective operators are not introduced in the language. For the same purposes and for well-founded based semantics, our WFM_p^s model presented in this paper is much more adequate. We have shown how one can detect the support on contradiction for arbitrary extended logic programs.

The introduction of the introspective operators of [41] in $WFSX_{\mathrm{p}}$ brings new issues. Our approach is distinct from [41]. There it is provided a full-fledged paraconsistent semantics for introspective programs allowing disjunction. The operational details are ignored. By means of our P^{int} program transformation we show how can we use the well-founded model for providing a semantics for introspective programs (without disjunction). These results are possible due to the properties of $WFSX_{\mathrm{p}}$, in particular its embedding in *WFS* first presented in this paper. We conjecture that we obtain similar results to the ones described in [41] for introspective programs without occurences of default negation. The introduction of default negation in introspective programs is not very well explored in [41], and one cannot compare the approaches. As we have shown, the semantics proposed by the authors also does not handle well detection of support on contradiction in paraconsistent programs with default negation. Furthermore, the authors do not discuss what is the appropriate semantics and its corresponding properties. An advantage of our semantics is the adoption of the coherence principle and a liberal way of propagating consequences of contradictory literals.

However, our introspective well-founded model may not give the intended results whenever the model contains undefined literals. This happens whenever there are support on contradictions which are not properly propagated. A solution of the kind of the one presented in Section 4 does not work, as we have shown. This points future directions of our work, which we intend to explore.

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. *Reasoning with Logic Programming.* Springer–Verlag, 1995. In print.
3. A. Almukdad and D. Nelson. Constructible falsity and inexact predicates. *JSL*, 49:231–233, 1984.
4. C. Baral and M. Gelfond. Logic programming and knowledge representation. *J. Logic Programming*, 19/20:73–148, 1994.
5. N. D. Belnap. A useful four-valued logic. In G. Epstein and J. M. Dunn, editors, *Modern Uses of Many-valued Logic*, pages 8–37. Reidel, 1977.
6. H. A. Blair and V. S. Subrahmanian. Paraconsistent logic programming. *Theoretical Computer Science*, 68:135–154, 1989.
7. N. Costa. On the theory of inconsistency formal system. *Notre Dame Journal of Formal Logic*, 15:497–510, 1974.
8. 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 LogIn '92*, pages 241–329. DeGruyter, 1995.
9. W. Drabent and M. Martelli. Strict completion of logic programs. *New Generation Computing*, 9(1):69–79, 1991.
10. M. Fitting. A Kripke-Kleene semantics for logic programs. *Journal of Logic Programming*, 2(4):295–312, 1985.
11. M. Fitting. Bilattices and the semantics of logic programming. *Journal of Logic Programming*, 11:91–116, 1991.
12. M. Fitting. Well-founded semantics, generalized. *Journal of Logic Programming*, 12:71–84, 1992.
13. J. Fox, P. Krause, and S. Ambler. Arguments, contradictions and practical reasoning. In B. Neumann, editor, *ECAI'92*, pages 623–627. John Wiley & Sons, 1992.
14. A. V. Gelder. The alternating fixpoint of logic programs with negation. In *8th Symposium on Principles of Database Systems.* ACM SIGACT-SIGMOD, 1989.
15. 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. MIT Press, 1988.
16. M. Gelfond and V. Lifschitz. Logic programs with classical negation. In Warren and Szeredi, editors, *ICLP'90*, pages 579–597. MIT Press, 1990.
17. M. Kifer and E. Lozinskii. A logic for reasoning with inconsistency. *Journal of Automated Reasoning*, 8:179–215, 1992.
18. M. Kifer and V. Subrahmanian. Theory of generalized annotated logic programming and its applications. *Journal of Logic Programming*, 12:335–367, 1992.
19. R. Kowalski and F. Sadri. Logic programs with exceptions. In Warren and Szeredi, editors, *7th International Conference on Logic Programming.* MIT Press, 1990.
20. P. Krause, S. Ambler, M. Elvang-Goransson, and J. Fox. A logic of argumentation for reasoning under uncertainty. *Computational Intelligence*, 11(1):113–131, 1995.

21. J. Minker and C. Ruiz. Semantics for disjunctive logic programs with explicit and default megation. *Fundamenta Informaticae*, 20(3/4):145–192, 1994.
22. D. Nelson. Constructible falsity. *JSL*, 14:16–26, 1949.
23. D. Pearce. Reasoning with Negative Information, II: hard negation, strong negation and logic programs. In D. Pearce and H. Wansing, editors, *Nonclassical logic and information processing*, LNAI 619, pages 63–79. Springer–Verlag, 1992.
24. D. Pearce. Answer sets and constructive logic, II: Extended logic programs and related nonmonotonic formalisms. In L. Pereira and A. Nerode, editors, *LPNMR'93*, pages 457–475. MIT Press, 1993.
25. 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.
26. D. Pearce and G. Wagner. Logic programming with strong negation. In P. Schroeder-Heister, editor, *ELP'91*, pages 311–326. LNAI 475, Springer–Verlag, 1991.
27. 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.
28. 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, Washington, USA, June 1991. MIT Press.
29. S. G. Pimentel and W. L. Rodi. Belief revision and paraconsistency in a logic programming framework. In A. Nerode, W. Marek, and V. S. Subrahmanian, editors, *LPNMR'91*, pages 228–242. MIT Press, 1991.
30. G. Priest, R. Routley, and J. Norman. *Paraconsistent logics*. Philosophia Verlag, 1988.
31. T. Przymusinski. Extended stable semantics for normal and disjunctive programs. In Warren and Szeredi, editors, *ICLP'90*, pages 459–477. MIT Press, 1990.
32. N. Rescher and R. Brandom. *The logic of inconsistency*. Basil Blackwell, 1980.
33. C. Sakama. Extended well–founded semantics for paraconsistent logic programs. In *Fifth Generation Computer Systems*, pages 592–599. ICOT, 1992.
34. C. Sakama and K. Inoue. Paraconsistent Stable Semantics for extended disjunctive programs. *Journal of Logic and Computation*, 5(3):265–285, 1995.
35. V. S. Subrahmanian. Paraconsistent disjunctive deductive databases. *Theoretical Computer Science*, 93:115–141, 1992.
36. S. Toulmin. *The uses of arguments*. Cambridge University Press, 1958.
37. 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.
38. G. Wagner. Reasoning with inconsistency in extended deductive databases. In L. M. Pereira and A. Nerode, editors, *LPNMR'93*, pages 300–315. MIT Press, 1993.
39. G. Wagner. Vivid logic: Knowledge-based reasoning with two kinds of negation. *Lecture Notes on Artificial Intelligence*, 764, 1994.
40. M. Wallace. Tight, consistent, and computable completions for unrestricted logic programs. *Journal of Logic Programming*, 15:243–273, 1993.
41. J.-H. You, S. Ghosh, L.-Y. Yuan, and R. Goebel. An introspective framework for paraconsistent logic programs. Technical report, D.C.S. of the University of Alberta, Canada, 1995.