A Survey of Paraconsistent Semantics for Logic Programs

Carlos Viegas Damásio and Luís Moniz Pereira

Abstract

In this chapter we motivate the use of paraconsistency, and survey the most salient paraconsistent semantics for (extended) logic programs. Most of the semantics are accompanied by their multi-valued model theory. The survey also presents new results regarding the embedding of part of these semantics into normal logic programs under Well-Founded Semantics [32], Partial Stable Model Semantics (or stationary semantics) [72], and Stable Model Semantics [34]. The conclusions provide a discussion of the relative merits and distinguishing features of paraconsistent logic programming semantics. We believe a comprehensive coverage of the topic as it stands at present is attained here. The reader is assumed to have a adequate knowledge of the semantics of normal logic programs.

1 Introduction

Our contribution to this volume consists in giving a logic programmer's view on handling program inconsistency. The semantics we cover will touch several aspects of *implementing* reasoning in the presence of contradiction. Logic programming has already shown a wide applicability for representing knowledge [10]. Also, the most important non-monotonic formalisms, for instance Default Logic [75] and Autoepistemic logics [53, 54], have a counterpart semantics on the logic programming side.[1] Moreover, logic programming has turned out to be vehicle for implementing and exploring other important aspects of Artificial Intelligence such as updates and belief revision. Therefore, it is not strange that a lot of work in the logic programming community has been carried out to understand the integration of paraconsistent reasoning with logic programming, in preparation for an applicational and implementational rôle of great potential, now emerging.

Until recently, a mechanism for explicitly declaring the falsity of propositions was not available in the restricted language of normal logic programs. The importance of extending logic programming with an explicit form of non-classical negation, "¬", beside the usual default one, has lately been stressed, for use in deductive databases, knowledge representation, and non-monotonic reasoning. This need has been recognized by several authors [59, 35, 44, 88, 61], which have proposed an enhanced language and semantics [35, 71, 61, 91]. Extended logic programming came thus to be born. A recent study of this explicit form

[1] For an overview of Modal Logics and Default Logics based knowledge representation formalisms, see the Chapters by Meyer and van der Hoek and Schaub in this volume.

of negation (and its strong form) compared with classical negation can be found in [5].

However, the introduction of explicit negation enables 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 [66, 91] there are three main ways of dealing with inconsistent information in logic programs (see also the introduction of Meyer and van der Hoek's Chapter in this volume):

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 [71, 35, 61].

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 [66, 62, 41, 64, 2, 7, 3]. It does not necessarily require an explicit paraconsistent semantics: the procedural program transforming revision operators suffice. This corresponds to a particular case of potential-contradictions view of handling inconsistency, as defined in the introductory chapter by Besnard and Hunter in this volume.

Paraconsistent approach: Accept contradictory information into the semantics and perform reasoning tasks that take it into account. This is the approach followed by at least a dozen semantics which we will describe in this paper, and corresponds to the actual-contradictions view of Besnard and Hunter (see their chapter in this volume).

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 for attaining 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 studied in philosophical logic by several authors [18, 38, 11, 39, 76, 6, 68]. The reader is referred to Hunter's Chapter in this volume for a more detailed discussion on the logical properties of several paraconsistent logics. Their intuitions and results have been brought

to the logic programming setting mainly by Blair, Pearce, Subrahmanian, and Wagner [13, 59, 60, 56, 57, 90, 91]. 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 both well-founded and answer sets semantics (see further on for references).

In this survey we emphasize the relationships between the diverse semantics, rather than providing a complete analysis of each one. Their prominent features will be singled out by a large body of examples. We try to reduce most of the semantics to either well-founded or stable model semantics. This means that they can be readily implemented, by means of the program transformations given here, using the existing implementations, for the two most important approaches to semantics of normal logic programs, skeptical and credulous.

Furthermore, the description of most of the semantics reported in this chapter are complemented with a short analysis of the underlying multi-valued logic theory. The most important properties of these logics are pointed out, for the sake of comparison. We avoid incursions into a full discussion of the logics. We believe that this presentation can give to the outsider a more complete view of the work carried out in logic programming, from different perspectives.

The plan of the chapter is as follows. We start by motivating the need for a paraconsistent semantics, with recourse to examples. Next we begin by discussing the semantics for definite extended logic programs, i.e. those without the occurrence of default negation. Then we cover non-monotonic semantics for extended logic programs with default negation based on the well-founded semantics. In a subsequent section we engage in a similar study, but for answer sets based semantics. We proceed by dissecting the semantics advanced for detecting support on contradiction. The semantics covered in these first sections all adhere to the actual-contradictions view. We finish our detailed overview in Section 7 with the works on logic programming which follow, or enable, the potential-contradictions view.

Unfortunately we cannot consider in this survey, in the same detail, all extant semantics for paraconsistent logic programs (extended or otherwise). Notable cases are the annotated logic programs extensions in [42, 43], with the exception of [13]. Fitting's semantics [28, 29] also will not be addressed in great detail. Still, in Section 8, we briefly describe the most important features of these and other related works.

Finally, we collect, in the last section, the main conclusions of our comparative study.

To help the reader plan his journey we exhibit in Figure 1 the dependencies among sections and subsections of this chapter.

Several reading paths are possible. In Table 1 we outline some possibilities, besides the advised one comprehending all sections. Sections 1, 2, 3, 8 and 9 are mandatory. The others can be traversed according to the reader's interest.

Concerning the semantics defined for the more general class of disjunctive extended logic programs, we study only their disjunction-free fragment.

Table 1. Some reading paths

Core	1 + 2 + 3 + 8 + 9
Bird's eye view of the topic	Core + 4.1 + 4.2 + 5.1
Well-founded based semantics	Core + 4.1 + 6.1 + 7.2
Stable-models based semantics	Core + 5 + 6.2 + 7.2
$WFSX_{\mathrm{p}}$ based semantics	Core + 4.1 + 4.2 + 6 + 7.2 + 7.3
Support on Contradiction	Core + 4.1 + 4.2 + 5.1 + 6
Blocking Contradiction	Core + 4 + 5.1 + 6 + 7
Actual Contradictions View	Core + 4 + 5 + 6 + 7
Potential Contradictions View	Core + 4 + 6.3 + 7

2 Motivating Examples

Paraconsistency is present in our everyday life. We support this statement with two examples of realistic situations. We also describe the main problems and requirements of a paraconsistent semantics. For the syntax used check Definition 1 and Definition 19. The reader acquainted with the syntax of first-order classical logic will certainly understand the examples.

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:

$$
\begin{aligned}
&bird(X) \leftarrow bill(X), warm\_blood(X), oviparous(X).\\
&mammal(X) \leftarrow hair(X), warm\_blood(X).\\
&\neg mammal(X) \leftarrow bird(X).\\
&\neg bird(X) \leftarrow mammal(X).\\
&flies(X) \leftarrow bird(X).\\
&nurses(X) \leftarrow mammal(X).
\end{aligned}
$$

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

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

Relevant conclusions are that cats are mammals, cats nurse their offspring, cats are not birds and do not fly. On the other hand, ducks are birds, ducks fly, ducks are not mammals and do not nurse.

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

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

According to the above program it is expectable that semantics for extended logic programs provide the information that platypuses are mammals and are not mammals, platypuses are birds and are not birds, platypuses fly and nurse.

The remarkable points about this example are manifold. First, contradictory information can coexist with safe information without interfering with each other; in particular, we must not relinquish the information about cats and ducks after introducing the characteristics of platypuses. Second, we should detect a contradiction about both the *mammal* and the *bird* predicates. Furthermore, since consequences of these predicates are to be propagated, we should be aware that the knowledge about platypuses regarding their nursing and flying abilities is open to doubt because supported on contradiction. Third, any correct information should be covered by the program's model: platypuses are mammals, do not fly, and nurse their progeny. Finally, it is unsound to introduce a heuristic rule to the effect of dropping all objective (or default) knowledge regarding platypuses. We want to retain that they nurse their descendants but also to discard the $fly(p)$ conclusion.

The above taxonomy example clearly identifies another desirable property of a semantics for extended logic programs. Assume the above knowledge is to be revised in order to consistently incorporate the new information regarding platypuses. All contradiction supported conclusions should be available so that the desirable and undesirable ones can be identified. Program revision (or declarative debugging if you prefer) may then be performed, and problematic rule instances pinpointed, namely in this example

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

Indeed, we have devised such declarative debugging techniques for logic programs on the basis of our paraconsistent semantics and contradiction removal methods. The reader is referred to [63, 64, 65] 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 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. An emergency patient arrives at a hospital with the following symptoms and signs[2]:

- 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[3];
- Furthermore, if he has high amylase levels then a perforation of a peptic ulcer can be eliminated; vice-versa, if he exhibits Jobert's manifestation then pancreatitis can be eliminated;
- Under either situation the patient should not be fed, and should take H_2 antagonists.

This scenario can be represented with the following extended logic program:

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

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

$$\neg nourish \leftarrow perforation.$$
$$\neg nourish \leftarrow pancreatitis.$$

$$H_2\_antagonist \leftarrow perforation.$$
$$H_2\_antagonist \leftarrow pancreatitis.$$

Notice that the rules for diagnosing perforation and pancreatitis make use of the default negation operator. If *high_amylase* is inserted in the program then the rule for *perforation* is not applicable. A similar reasoning is applicable to the second one regarding Jobert's manifestation.

Now assume that the physician is uncertain regarding the amylase levels and existence of Jobert's manifestation. He either adds nothing to the program or

[2] We thank Doctor of Medicine António Dias Ribeiro of Hospital dos Covões, Coimbra, for providing us with this example. Any mistakes in coding it are entirely our own.

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

he adds the information $\neg high\_amylase$ and $\neg jobert$ to the system, in addition to the other symptoms $sudden\_pain$, $abd\_tenderness$, and $per\_irritation$. In our opinion, the same results should be obtainable.

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 $\neg surgery\_indication$. In such a problematic cases the patient is usually operated on, regardless. Therefore, it should be possible in some situations to test whether a conclusion depends on a contradiction.

Another motivation for the use of paraconsistency, and an important one, is that queries should be executable in a goal-oriented fashion. That being the case, first it is not desirable to pay an extra price for allowing paraconsistency. Second, when queries are evaluated in a goal-oriented fashion it is not recommended 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 should help in identifying the reasons behind non-intended conclusions. Restoring consistency can then be carried out in a computationally more demanding second stage of contradiction removal. Furthermore, since a precise and declarative meaning of programs should be provided by the underlying semantics, one should not be reluctant in having consequences and conclusions based on contradiction, as long as they can be distinguished from the other ones. Moreover, as we have noted, keeping contradiction can be important for knowledge-representation because it proffers significative information about contradiction-supported conclusions.

3 Definite Extended Logic Programs

Next we introduce the language of definite extended logic programs without default negation (or monotonic extended logic programs) and a semantics which is a common denominator to almost all other semantics examined in this survey. It is related to Blair and Subrahmanian's generalized Horn programs [13], Wagner's logic programs with strong negation [90, 91], and Almukdad and Nelson's paraconsistent constructive system N^- [6].

3.1 Language and Semantics

As usual, for the sake of simplicity and without loss of generality, we will restrict the discussion to (possibly infinite) propositional 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 $OLit = At \cup \neg At$.

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

$$L_0 \leftarrow L_1, \ldots, L_m. \qquad (m \geq 0)$$

where L_i $(0 \leq i \leq m)$ are objective literals. If $m = 0$ the rule $L_0 \leftarrow$ is said to be a fact and will be denoted simply by L_0.

One natural idea is to view definite extended logic programs in such a way that explicitly negated literals are simply envisaged as new atoms. As remarked in [83] this approach was put forward in [51].

Definition 2. Let P be a definite extended logic program. The model $\mathcal{M}_P$ of the program is obtained as follows:

1. Transform program P into a definite logic program $P^{\neg}$ by substituting every occurrence of objective literals a and $\neg a$, respectively, by a^p and a^n.
2. Let $M^{\neg}$ be the minimal model of $P^{\neg}$, which can be computed by the ordinary T_P operator [25].
3. Then, to obtain the model $\mathcal{M}_P$, reverse the mapping of the first step by transforming $a^p \in M^{\neg}$ $(a^n \in M^{\neg})$ into $a \in \mathcal{M}_P$ $(\neg a \in \mathcal{M}_P)$.

It is obvious that this semantics is monotonic, since it relies on classical logic to determine the meaning of a program.

Example 3. [61, 4] Consider the classical birds example:

$$\begin{array}{l} flies(X) \leftarrow bird(X). \\ \neg flies(X) \leftarrow penguin(X). \\ bird(X) \leftarrow penguin(X). \end{array}$$

Suppose the facts $bird(tweety)$ and $penguin(fred)$ are added to the above rules. The model $\mathcal{M}_P$ of this program is:

$$\{flies(tweety), bird(tweety)\} \cup$$
$$\{flies(fred), \neg flies(fred), bird(fred), penguin(fred)\}$$

Notice that in the corresponding classical theory every literal is entailed by the program, because of the "ex falso quodlibet" principle.

If an arbitrary set of new facts or rules is added to the program, the corresponding model will be a superset of the above. This behaviour is justified by the monotonicity of the semantics.

Recall that Besnard and Hunter, in the introductory chapter of this volume, identify three schemas for detecting the presence of contradiction in a theory: C-scheme, A-scheme and N-scheme. The C-scheme says that inconsistency arises when all formulas are inferred. In the A-scheme a set of elements of the language

is used to represent absurdity. Finally, the N-scheme relies on the existence of a negation operator for capturing the notion of contradiction: the simultaneous truth of a formula A and its negation $\neg A$ represents an inconsistency.

As can be seem from the above example, we face a contradiction whenever $\mathcal{M}_P$ contains a literal L and its explicit negation $\neg L$, corresponding to the adoption of the N-scheme presented. However, from this contradiction we do not conclude everything nor do we have a special proposition identifying this contradiction on this particular objective literal: the C-scheme and A-scheme are not enforced. Even though the latter two schemes are not inbuilt into the semantics, the user can explicitly program them, as shown in the next examples. The semantics is compatible with these schemas.

Example 4. Let us recall the first example of the introduction of this volume. Consider the language $\mathcal{L} = \{rain, snow, sun\}$ and the relation C such that $C(T) = \mathcal{L}$ iff $\{rain, snow\} \subseteq T$. In all other cases $C(T) = T$. Such a relation can be implemented by the following definite extended logic program containing theory T and the following rules:

$$rain \leftarrow rain, snow. \quad snow \leftarrow rain, snow. \quad sun \leftarrow rain, snow.$$

In fact one can drop the first two rules, as they are void.

Example 5. Consider Besnard and Hunter's illustrating example of the combination of the A and N-schemes. The language is $\mathcal{L} = \{coloured, \neg coloured, solid, \neg solid, fool_c, fool_s\}$. Recall that the literals $fool_c$ and $fool_s$ represent, respectively, contradiction on literal *coloured* and *solid*. The inference relation should obey:

$$\begin{aligned} C(\{coloured, \neg coloured\}) &= \{coloured, \neg coloured, fool_c\} \\ C(\{solid, \neg solid\}) &= \{solid, \neg solid, fool_s\} \end{aligned}$$

The logic programming rules implementing these constraints are:

$$fool_c \leftarrow coloured, \neg coloured \qquad fool_s \leftarrow solid, \neg solid$$

All other examples can be worked out in a similar fashion. We can now state our first main result. The most important semantics surveyed here coincide on the class of definite extended logic programs.

Theorem 3. *Let P be a definite extended logic programs. Apart from syntactical differences, all the semantics reported in [13, 66, 78, 57, 3, 4, 80], and Wagner's logic programs with strong negation [91] and with liberal reasoning [90] are isomorphic to $\mathcal{M}_P$.*

The above theorem fully characterizes the several semantics of definite extended logic programs in terms of the semantics given in Definition 2.

3.2 Model-theoretic semantics

In order to get a better understanding of the underlying "logic", we now provide a model-theoretical characterization of the semantics. We resort to Belnap's four-valued logic [11] depicted in Figure 2. The first use of this semantics for providing the meaning of paraconsistent logic programs was reported in [13]. We discuss this approach in detail in Section 3.3. The truth-values $\bot$, **f**, **t**, $\top$ stand, respectively, for *undetermined*, *false*, *true*, and *overdefined* (or *contradictory*).

We will resort to the type of representation of Figure 2 to synthetically characterize the several logics to be discussed in this chapter. First, two orderings on the truth-space are defined: the truth-ordering and the knowledge ordering. In the former, along the **t**-axis, we have $\mathbf{f} \prec_t \bot$, $\mathbf{f} \prec_t \top$, $\bot \prec_t \mathbf{t}$ and $\top \prec_t \mathbf{t}$. In the latter, along the **k**-axis, we have $\bot \prec_k \mathbf{f}$, $\bot \prec_k \mathbf{t}$, $\mathbf{f} \prec_k \top$ and $\mathbf{t} \prec_k \top$.

Our interpretation of the connectives "$\wedge$", "$\vee$", and "$\neg$" is the one described in [11] and can be found in Tables 2 to 4. The conjunction operation corresponds to the meet in the truth-ordering while the disjunction operation reflects the join in the same ordering. The negation operation is obtained by flipping the diagram along the $\bot/\top$ vertical axis.

Table 2. Truth table for negation

A	$\bot$	**f**	**t**	$\top$
$\neg A$	$\bot$	**t**	**f**	$\top$

Table 3. Truth table for conjunction

$\wedge$	$\bot$	**f**	**t**	$\top$
$\bot$	$\bot$	**f**	$\bot$	**f**
f	**f**	**f**	**f**	**f**
t	$\bot$	**f**	**t**	$\top$
$\top$	**f**	**f**	$\top$	$\top$

We differ from Belnap's logic in the definition of the consequence relation and in the associated implication connective (presented in Table 5).

We define the propositional language of the logic as usual, the notion of interpretation mapping propositional symbols into truth-values (in this case $\bot$, **f**, **t** and $\top$) and its generalization to arbitrary formulas (truth-valuation). We

Table 4. Truth table for disjunction

$\vee$	$\bot$	**f**	**t**	$\top$
$\bot$	$\bot$	$\bot$	**t**	**t**
f	$\bot$	**f**	**t**	$\top$
t	**t**	**t**	**t**	**t**
$\top$	**t**	$\top$	**t**	$\top$

Table 5. Truth table for implication

$\leftarrow$	$\bot$	**f**	**t**	$\top$
$\bot$	**t**	**t**	**f**	**f**
f	**t**	**t**	**f**	**f**
t	**t**	**t**	**t**	**t**
$\top$	**t**	**t**	**t**	**t**

designate this logic $\mathcal{FOUR}$. Interpretations are ordered by the usual extension to sets of literals of the knowledge ordering among literals.

The translation of definite extended logic programs into $\mathcal{FOUR}$ logic theories is immediate. We make use of the notion of *designated truth-values* from many-valued logics [76, 87], to define the consequence relation.

Definition 4. Let I be a $\mathcal{FOUR}$ interpretation and $\hat{I}$ the corresponding truth-valuation. Let F be an arbitrary propositional formula. We say that I satisfies F, denoted by $I \models_4 F$, if and only if $\hat{I}(F) \in \{\mathbf{t}, \top\}$.

The set $\{\mathbf{t}, \top\}$ forms the set of designated truth-values. An interpretation I is a model of a theory iff it satisfies all the formulas in the theory. As usual, $I \not\models_4 F$ stands for "I does not satisfy F."

Note that the implication operator always evaluates to either **f** or **t**. The whole point of the above definition is to ensure that the following equivalences are valid:

Theorem 5. *Let I be a $\mathcal{FOUR}$ interpretation. Then,*

$$I \models_4 A \wedge B \quad \textit{iff } I \models_4 A \textit{ and } I \models_4 B$$
$$I \models_4 A \vee B \quad \textit{iff } I \models_4 A \textit{ or } I \models_4 B$$
$$I \models_4 A \leftarrow B \textit{ iff } I \models_4 A \textit{ or } I \not\models_4 B$$

To further characterize our logic we need to clarify the notion of equivalence. In a multi-valued setting one can define (at least) two natural notions of equivalence, one based on the truth-value assignment and another on the consequence relation.

Definition 6. Let F and G be two formulas of the language of $\mathcal{FOUR}$. We say that $F \equiv_4 G$ iff $\hat{I}(F) = \hat{I}(G)$ for every interpretation I. On the other hand, the equivalence $F \models\!\models_4 G$ holds whenever for every interpretation I we have $I \models_4 F$ iff $I \models_4 G$. Otherwise, $F \models\!\models_4 G$ is false.

Theorem 7. *Let F and G be two formulas of the language of $\mathcal{FOUR}$. Then,*

$$\textit{If } F \equiv_4 G \textit{ then } F \models\!\models_4 G$$

Note that the above theorem is valid for an arbitrary many-valued logic, whenever $\equiv$ is defined as truth-value equality and $\models$ is expressed by means of a set of designated truth-values. The two forms of equivalence will be used only at the meta-language level. Unless stated otherwise, when we talk about equivalence we mean the weaker form $\models\!\models$.

The equivalences true in $\mathcal{FOUR}$ are similar to the ones holding in classical logic. In fact, one can easily show that the associative, distributive, commutative, idempotent, absorption, double negation, zero/one, and De Morgan's laws all hold in the logic $\mathcal{FOUR}$. Also, the Deduction Theorem and Modus Ponens hold as well.

However, the logic defined is not the classical one. For instance, the excluded middle and the law of contradiction are not valid (and hence the C-scheme is not complied with). Furthermore, the implication is not definable in terms of the other connectives and the contraposition laws are not obeyed[4]. Modus Tollens and Disjunctive Syllogism fail. Interestingly, all axioms of propositional logic hold with the single exception of

$$(A \rightarrow B) \rightarrow ((A \rightarrow (\neg B)) \rightarrow (\neg A))$$

corresponding to the introduction rule for negation of the natural deduction calculus. The negation of $\mathcal{FOUR}$ is not intuitionistic, since both the elimination and the introduction rules for negation of intuitionistic logic are not obeyed. Clearly, the logic presented here is neither da Costa's C_ω system [18], since excluded middle is not satisfied, nor Belnap's original logic [Belnap, 1977], since Modus Ponens is a sound rule in $\mathcal{FOUR}$. For more details consult Anthony Hunter's chapter in this volume.

After this small detour we provide the desired relationship of logic $\mathcal{FOUR}$ to definite extended logic programs. The following theorem expresses the correspondence between the consequence relation and the truth-assignment of propositional symbols.

Theorem 8. *Let A be a propositional symbol and I an interpretation in a language containing A. Then,*

[4] For a definition of these properties see Tables 15 and 16 in the last section of this chapter.

$$I \models_4 A \text{ and } I \models_4 \neg A \text{ iff } \hat{I}(A) = \top$$
$$I \models_4 A \text{ and } I \not\models_4 \neg A \text{ iff } \hat{I}(A) = \mathbf{t}$$
$$I \not\models_4 A \text{ and } I \models_4 \neg A \text{ iff } \hat{I}(A) = \mathbf{f}$$
$$I \not\models_4 A \text{ and } I \not\models_4 \neg A \text{ iff } \hat{I}(A) = \bot$$

The results of Theorem 8 are synthesized in Figure 3. A literal is L entailed by an interpretation I iff $\hat{I}(L)$ maps to $\mathbf{t}$ or $\top$. The explicit complement of L, i.e. $\neg L$, holds iff $\hat{I}(L)$ maps to $\mathbf{f}$ or $\top$. In the remaining of this chapter we will present the logics as in Figure 3, conveying the information relating the consequence relation with the truth-valuation function.

The above theorem paves the way to defining an isomorphism between $\mathcal{FOUR}$ interpretations and sets of objective literals on the logic programming side. The relationship is given by Theorem 9.

Theorem 9. *Let P be a definite extended logic program. $\mathcal{M}_P$ is the model of P as per Definition 2 iff the following $\mathcal{FOUR}$ interpretation I_P is the least $\mathcal{FOUR}$ model of P with respect to the knowledge ordering. Let A be an arbitrary proposition in the language of P. Interpretation I_P is given by:*

$$\hat{I}(A) = \bot \text{ iff } A \notin \mathcal{M}_\mathcal{P} \text{ and } \neg A \notin \mathcal{M}_\mathcal{P}$$
$$\hat{I}(A) = \mathbf{f} \text{ iff } A \notin \mathcal{M}_\mathcal{P} \text{ and } \neg A \in \mathcal{M}_\mathcal{P}$$
$$\hat{I}(A) = \mathbf{t} \text{ iff } A \in \mathcal{M}_\mathcal{P} \text{ and } \neg A \notin \mathcal{M}_\mathcal{P}$$
$$\hat{I}(A) = \top \text{ iff } A \in \mathcal{M}_\mathcal{P} \text{ and } \neg A \in \mathcal{M}_\mathcal{P}$$

Example 6. Consider the definite extended logic program of Example 3. The least $\mathcal{FOUR}$ model I_P of P is given by the mapping:

$$I_P\,(flies(tweety)) = \mathbf{t} \qquad I_P\,(flies(fred)) = \top$$
$$I_P\,(bird(tweety)) = \mathbf{t} \qquad I_P\,(bird(fred)) = \mathbf{t}$$
$$I_P\,(penguin(tweety)) = \bot \quad I_P\,(penguin(fred)) = \mathbf{t}$$

There are other models of the program which convey unsupported information, for instance:

$$I_P\,(flies(tweety)) = \mathbf{t} \qquad I_P\,(flies(fred)) = \top$$
$$I_P\,(bird(tweety)) = \mathbf{t} \qquad I_P\,(bird(fred)) = \top$$
$$I_P\,(penguin(tweety)) = \mathbf{f} \quad I_P\,(penguin(fred)) = \mathbf{t}$$

This model is not minimal, as required by Theorem 9. Clearly, knowledge regarding non-penguins and non-bird entities is not expressed in the program. Therefore, it is unexpected to have $\neg penguin(tweety)$ and $\neg bird(fred)$ entailed by the program's model.

3.3 Blair & Subrahmanian's generalized Horn programs

We present in this section the logic programming-based semantics defined in [13]. Historically, this is the first semantics for logic programs taking into account the problems of handling contradiction in a coherent manner, and resorts to a many-valued logic. The authors emphasize the need for a paraconsistent logic programming semantics as a means to produce a declarative semantics for arbitrary sets of clauses. They introduced the four-valued logic described in the previous subsection and made use of the knowledge ordering $\preceq_k$. The main contribution to our discussion is in the definition of a monotonic fixpoint operator with respect to $\preceq_k$, providing a direct way to compute the least model I_P of a definite extended logic program. The negation operation is also adopted and defined by them according to Table 2. We will restrict the comparisons and definitions to the ground case.

Definition 10. [13] A generalized Horn program (GHP) is a set of gh-clauses, i.e. those of the form $A_0 : \mu_0 \Leftarrow A_1 : \mu_1 \& \ldots \& A_n : \mu_n$ where $A_0, \ldots, A_n$ are objective literals and $\mu_0, \ldots, \mu_n$ are truth values (or annotations) from $\mathcal{T} = \{\bot, \mathbf{f}, \mathbf{t}, \top\}$.[5]

In the original definition, a GHP is a finite set of possibly non-ground gh-clauses. The definition of interpretation is, as usual, a function from the Herbrand base into the set of truth-values $\mathcal{T}$ forming a complete lattice, ordered by the extension to interpretations of the order among literals $\preceq_k$. The notion of satisfaction of a formula by an interpretation is again as usual, with the exception of annotated literals. An interpretation I satisfies $A : \mu$ iff $I(A) \succeq_k \mu$, and I satisfies $\neg A : \mu$ iff it satisfies $A : \neg\mu$. Thus, every negated annotated literal $\neg A : \mu$ can be replaced by $A : \neg\mu$ everywhere in the program. The semantics of a generalized Horn program G is given by its least model under ordering $\preceq_k$. This model can be determined by a continuous operator on the complete lattice of interpretations, which is the natural generalization of T_P to GHPs.

Definition 11. [13] Suppose G is a GHP. Then T_G^{GHP} is a mapping from the Herbrand interpretations of G to the Herbrand interpretations of G defined by:

$$T_G^{\mathrm{GHP}}(I)(A) = lub\{\mu \mid A : \mu \Leftarrow B_1 : \mu_1 \& \ldots \& B_k : \mu_k \in G \text{ and } I \models B_1 : \mu_1 \& \ldots \& B_k : \mu_k\}.$$

In other words, the truth value μ of A is the least upper bound of all the μs of gh-clauses for A with body satisfied in I. Note that $lub\{\} = \bot$.

[5] In fact, the authors only consider well-annotated literals in the above rules. A literal $A : \mu$ is well-annotated iff μ is $\mathbf{f}$ or $\mathbf{t}$. Their results easily carry over to this more general syntax [43].

Example 7. Consider the following program G:

$$p(a) : \mathbf{t} \Leftarrow q(a) : \mathbf{f} \;\&\; r(a) : \mathbf{t}$$
$$p(b) : \mathbf{t} \Leftarrow q(b) : \mathbf{f} \;\&\; r(b) : \mathbf{t}$$
$$p(c) : \mathbf{t} \Leftarrow q(c) : \mathbf{f} \;\&\; r(c) : \mathbf{t}$$

$$q(a) : \mathbf{t} \Leftarrow \qquad q(b) : \mathbf{f} \Leftarrow \qquad q(c) : \mathbf{f} \Leftarrow$$
$$r(a) : \mathbf{t} \Leftarrow \qquad r(a) : \mathbf{f} \Leftarrow$$
$$r(b) : \mathbf{f} \Leftarrow \qquad r(c) : \mathbf{t} \Leftarrow$$

The computation of the least fixpoint proceeds as follows:

	$T_G^{\mathrm{GHP}} \uparrow^0$	$T_G^{\mathrm{GHP}} \uparrow^1$	$T_G^{\mathrm{GHP}} \uparrow^2$	$=$	$T_G^{\mathrm{GHP}} \uparrow^3$
$p(a)$	$\bot$	$\bot$	$\bot$	$=$	$\bot$
$p(b)$	$\bot$	$\bot$	$\bot$	$=$	$\bot$
$p(c)$	$\bot$	$\bot$	$\mathbf{t}$	$=$	$\mathbf{t}$
$q(a)$	$\bot$	$\mathbf{t}$	$\mathbf{t}$	$=$	$\mathbf{t}$
$q(b)$	$\bot$	$\mathbf{f}$	$\mathbf{f}$	$=$	$\mathbf{f}$
$q(c)$	$\bot$	$\mathbf{f}$	$\mathbf{f}$	$=$	$\mathbf{f}$
$r(a)$	$\bot$	$\top$	$\top$	$=$	$\top$
$r(b)$	$\bot$	$\mathbf{f}$	$\mathbf{f}$	$=$	$\mathbf{f}$
$r(c)$	$\bot$	$\mathbf{t}$	$\mathbf{t}$	$=$	$\mathbf{t}$

The relationship to definite extended logic programs is made via the following definition:

Definition 12. Let G be a GHP and consider an arbitrary gh-clause of Q of the form $A_0 : \mu_0 \Leftarrow A_1 : \mu_1 \& \ldots \& A_n : \mu_n$. The body of the gh-clause is transformed into a conjunction *Body*, of objective literals, by replacing each annotated literal $A_i : \mathbf{t}$ ($A_i : \mathbf{f}$) by A_i ($\neg A_i$). Annotated literals $A_i : \top$ are replaced by the conjunction $A_i \wedge \neg A_i$. Literals $A_i : \bot$ are deleted since they are satisfied in any interpretation.

The definite extended logic program Q^{GHP} is constructed from each of the gh-clauses as follows. If $\mu_0 = \bot$ we add nothing to Q^{GHP}. If $\mu_0 = \mathbf{f}$ ($\mu_0 = \mathbf{t}$) we add the rule $\neg A_0 \leftarrow \mathit{Body}$ ($A_0 \leftarrow \mathit{Body}$). Otherwise, i.e. $\mu_0 = \top$, we add both of these rules to the program.

Example 8. Let G be the following GHP:

$$a : \bot \Leftarrow b : \mathbf{t}. \quad a : \mathbf{t} \Leftarrow b : \mathbf{f} \;\&\; c : \top \quad b : \mathbf{f} \Leftarrow c : \mathbf{t} \quad c : \top \Leftarrow d : \bot.$$

The resulting transformed extended logic program G^{GHP} is:

$$a \leftarrow \neg b, c, \neg c. \qquad \neg b \leftarrow c. \qquad c. \qquad \neg c.$$

The transformation of Definition 12 is very similar to the one appearing in [91], ours being defined for all generalized Horn programs. Essentially, Wagner does not consider the case of body literals annotated with $\bot$.

Theorem 13. *Let G be a GHP, $\mathcal{G} = lfp(T_G^{\mathrm{GHP}})$ and $\mathcal{M} = \mathcal{M}_{G^{\mathrm{GHP}}}$. The following equivalence between $\mathcal{G}$ and $\mathcal{M}$ holds:*

$$\begin{array}{l} \mathcal{G}(A) = \bot \textit{ iff } A \notin \mathcal{M} \textit{ and } \neg A \notin \mathcal{M} \\ \mathcal{G}(A) = \mathbf{f} \textit{ iff } A \notin \mathcal{M} \textit{ and } \neg A \in \mathcal{M} \\ \mathcal{G}(A) = \mathbf{t} \textit{ iff } A \in \mathcal{M} \textit{ and } \neg A \notin \mathcal{M} \\ \mathcal{G}(A) = \top \textit{ iff } A \in \mathcal{M} \textit{ and } \neg A \in \mathcal{M} \end{array}$$

Example 9. Consider the translation of the GHP of Example 7:

$$\begin{array}{llllll} p(a) \leftarrow \neg q(a), r(a). & q(a). & \neg q(b). & \neg q(c). & & \\ p(b) \leftarrow \neg q(b), r(b). & r(a). & \neg r(a). & \neg r(b). & r(c). & \\ p(c) \leftarrow \neg q(c), r(c). & & & & & \end{array}$$

The model is $\{p(c), q(a), \neg q(b), \neg q(c), r(a), \neg r(a), \neg r(b), r(c)\}$, correspondong to $T_G^{\mathrm{GHP}} \uparrow^3$ of Example 7.

Of course, the converse question naturally arises: are we able to transform definite extended logic programs into generalized Horn programs ? The answer is yes. The result follows from the previous theorem since it is clear that GHPs without literals annotated with $\bot$ or $\top$ are in one-to-one correspondence with definite extended logic programs. As an immediate corollary we realize that every GHP can be transformed into an equivalent GHP without occurrences of literals annotated with $\bot$ or $\top$, corresponding to the well-annotated programs of [13]. Furthermore, operator T_G^{GHP} can be readily applied in the computation of the least $\mathcal{FOUR}$ interpretation I_P of a definite extended logic program by inverting the GHP-transformation of Definition 12.

What is the operational mechanism necessary to implement Blair and Subrahmanian's semantics? Most of their article is devoted to the study of a SLD-like proof procedure for their semantics. They provide complex definitions and very weak results regarding its soundness and completeness. By translating GHPs into logic programs a derivation procedure based on SLD is readily implementable.

3.4 Wagner's logic programs with strong negation

Wagner has been a supporter of the introduction of explicit negation and constructive based paraconsistent logics in logic programming [59, 60, 88, 89, 90, 91]. His main motivating works are Nelson's constructive logic N with "strong negation" [55], Belnap's system B [11] and Levesque's vivid reasoning research programme
[46, 47]. In his book [91] several proposals for extending logic programming with strong negation in Nelson's sense are explored, under the form of "Vivid Rule Knowledge Bases." One such system corresponds, as we shall see, to what we previously named definite extended logic programs.

Definition 14. The language of logic programs with strong negation consists of the logical operator symbols $\wedge$, $\vee$, $\sim$ and 1, standing for conjunction, disjunction, strong negation, and the verum, respectively, where the latter represents a tautology.

A logic program with strong negation is a set of clauses of the form $l \leftarrow F$ where l is an atom a or its strong negation $\sim a$, and F an arbitrary formula.

The author assumes the language to have a finite number of atoms, with no function symbols. As usual and without loss of generality, we will restrict the discussion to the ground instantiation of a logic program with strong negation. Furthermore, we assume that the premise F of an implicational formula $l \leftarrow F$ is a conjunction of literals. As shown by Wagner [91], every program in the full syntax can be transformed into an equivalent one in this more restricted format.

His definition of the intended semantics for this class of programs is quite simple and is based on the notion of partial models:

Definition 15. A partial Herbrand interpretation $\mathcal{M} = \langle M^t, M^f \rangle$ is a pair of sets of atoms, where M^t contains the true atoms and M^f the false ones. A partial interpretation gives rise to both a model relation ($\models$) and a countermodel one ($=\!\!|$):

$$\begin{array}{lll}
\mathcal{M} \models a & \text{iff} & a \in M^t \\
\mathcal{M} \models F \wedge G & \text{iff} & \mathcal{M} \models F \text{ and } \mathcal{M} \models G \\
\mathcal{M} \models F \vee G & \text{iff} & \mathcal{M} \models F \text{ or } \mathcal{M} \models G \\
\mathcal{M} \models \sim F & \text{iff} & \mathcal{M} =\!\!| F
\end{array}$$

$$\begin{array}{lll}
\mathcal{M} =\!\!| a & \text{iff} & a \in M^f \\
\mathcal{M} =\!\!| F \wedge G & \text{iff} & \mathcal{M} =\!\!| F \text{ or } \mathcal{M} =\!\!| G \\
\mathcal{M} =\!\!| F \vee G & \text{iff} & \mathcal{M} =\!\!| F \text{ and } \mathcal{M} =\!\!| G \\
\mathcal{M} =\!\!| \sim F & \text{iff} & \mathcal{M} \models F
\end{array}$$

The attentive reader will notice the similarities of the above definition with Levesque's logic of implicit belief (see Meyer and Hoek's chapter in this volume). However, Wagner does not define a belief operator in his language.

The definition of model for logic programs with strong negation is obtained by describing the intended meaning of the implication rule operator:

Definition 16. A partial Herbrand interpretation $\mathcal{M}$ is called a model of a logic program Π with strong negation, symbolically $\mathcal{M} \models \Pi$, if, for all $l \leftarrow F \in \Pi$, $\mathcal{M} \models F$ implies $\mathcal{M} \models l$.

With this model definition, Wagner showed that every logic program with strong negation has a least model $\mathcal{M}_\Pi$ with the property that a formula F is entailed in every model of Π iff it is entailed by $\mathcal{M}_\Pi$. The semantics of his programs is provided by this least model $\mathcal{M}_\Pi$. This is not surprising and the attentive reader will notice immediately the similarities of the above definitions with the semantics for definite extended logic programs of Section 3.1. In fact the following natural correspondence holds:

Theorem 17. *Let Π^s be the extended logic program obtained from a logic program with strong negation Π by substituting every literal of the form $\sim A$ by $\neg A$. Then, $\mathcal{M}_\Pi \models a$ iff $a \in \mathcal{M}_{\Pi^s}$ and $\mathcal{M}_\Pi \models\!\sim a$ iff $\neg a \in \mathcal{M}_{\Pi^s}$.*

After model-theoretically defining his semantics, Wagner tries to give it a proof-theoretic interpretation. Wagner characterizes a special class of logic programs, which he designates well-founded and semi-well-founded, by showing that $\mathcal{M}_\Pi \models F$ iff $\Pi \models_{N^-} F$, where F is an implication free formula and $\models_{N^-}$ is the paraconsistent system obtained from Nelson's logic N by dropping the axiom schema $\varphi \rightarrow (\sim \varphi \rightarrow \psi)$ [6], i.e. adopting a non-explosive behaviour in face of contradiction (not everything follows from a contradiction, i.e. dropping the C-scheme). In fact, the adequateness of $\mathcal{M}_\Pi$ with respect to logic N^- is known to hold for arbitrary definite extended logic programs:

Theorem 18. [56, 57] *Let Π be a logic program with strong negation, and L a literal in the language of Π. Then L belongs to $\mathcal{M}_\Pi$ iff $\Pi \models_{N^-} L$.*

Therefore, for the case of definite extended logic programs, the objective literal consequences of $\mathcal{M}_\Pi$ are the consequences of Nelson's paraconsistent constructive logical system N^-. Therefore, the logic provided in Section 3.2 is equivalent to the restriction of Almukdad and Nelson's system N^- to the language of definite extended logic programs! Other results relating ordinary logic programming semantics with constructive logics can be found in [15, 16, 17].

4 Extended Logic Programs: WFS based semantics

Since definite extended logic programming is monotonic in character, a non-monotonic reasoning mechanism is needed to increase the expressive power of the language. As usual, the default negation operator "*not*" is employed for that purpose.

We start by extending the language of extended logic programs with the default negation symbol, applicable to objective literals only. Thus, 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 19. 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.

With the introduction of default negation a Pandora's box opens. We address in this section only three-valued extended logic programming semantics, and in the next section the two-valued ones. Two paraconsistent extensions of Well-Founded Semantics are presented, namely Sakama's extended well-founded semantics and our own paraconsistent Well-Founded Semantics with Explicit Negation [3, 20], $WFSX_{\mathrm{p}}$. We conclude by studying Wagner's Vivid Knowledge Bases under liberal reasoning [90, 91].

4.1 Sakama's Extended Well-founded Semantics and Przymusinski's Extended Stable Semantics

The first attempt to introduce explicit negation into well-founded semantics was made by Przymusinski in [71], motivated by the work of Gelfond and Lifschitz in answer sets semantics [35]. His semantics also views explicitly negated literals simply as new atoms. However Przymusinski's Extended Stable Semantics, by definition, discards all inconsistent models, i.e. those which contain both A and $\neg A$ for some atom A. This corresponds to the adoption of the C-scheme and the N-scheme discussed in the introduction of this volume. Sakama dropped the C-scheme in his Extended Well-Founded Semantics [78], thereby obtaining a paraconsistent version of *WFS* for extended logic programs (*EWFS*). The original presentation of his semantics is based on Przymuskinski's constructive definition of *WFS* in terms of the Θ operator [69]. We prefer the following more declarative and simpler definition, which is equivalent to the one in [78] and supercedes the one of Definition 2.

Definition 20. Let P be an extended logic program. The extended well-founded model M_P of the program is obtained as follows:

1. Transform program P into a normal logic program $P^{\neg}$ by replacing all occurrences of literals A, $\neg A$, $not\, A$ and $not\, \neg A$ by, respectively, A^p, A^n, $not\, A^p$, and $not\, A^n$.
2. To obtain the model M_P undo the mapping of the previous step with respect to the resulting $WFM(P^{\neg})$.

Example 10. Consider the following extended logic program:

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

The extended well-founded model of this program is $\{\neg a, not\, \neg b\}$. Mark that even though $\neg a$ belongs to the model we do not have $not\, a$ and hence b neither.

Probably one of the most interesting aspects of Sakama's paper is the (tentative) model-theoretical characterization of his *EFWS*. The author uses logic **VII** of Ginsberg [36, 37] to define the model-theory for his version of *WFS*.

Logic **VII** was introduced by Ginsberg to represent default assumptions (see Figure 4).

The reading of the 7 truth-values $\{\bot, \mathbf{df}, \mathbf{dt}, *, \mathbf{f}, \mathbf{t}, \top\}$ is, respectively, *undefined*, *false by default*, *true by default*, *don't care by default*, *false*, *true*, and *contradictory*. The consequence relation $\models_7$ is defined, again, in terms of the set of designated truth-values $\{\mathbf{t}, \top\}$. For details consult Definition 4. The underlying language of logic theories is extended with the *not* operator. The definition of the two negation operations is given in Table 6.

Once again, the $\neg$ operation flips the diagram around the vertical $\bot/\top$ axis. Clearly, default negation is not an extension of classical negation since we have $not\, \mathbf{f} = \mathbf{dt}$ and $not\, \mathbf{t} = \mathbf{df}$. We believe that there is no natural reading for the default negation operator, namely what regards the mapping of the more definite

Table 6. Truth table for negations in Logic **VII**

A	$\bot$	**df**	**dt**	$*$	**f**	**t**	$\top$
$\neg A$	$\bot$	**dt**	**df**	$*$	**t**	**f**	$\top$
not A	$\bot$	**t**	**f**	$\top$	**df**	**dt**	$*$

truth-values (**f**,**t**, and $\top$) to their default negations (**dt**, **df**, and $*$). In fact, one cannot say that *not* , as defined by Sakama, is a true negation if we stick to the following three fundamental properties:

Definition 21. [37] A negation operator "$\sim$" in a bilattice should satisfy the following three properties:

1. If $A \leq_k B$ then $\sim A \leq_k \sim B$;
2. If $A \leq_t B$ then $\sim B \leq_t \sim A$;
3. It is always the case that $A \equiv \sim\sim A$;

If the double negation condition (3) is not verified then the negation operator is said to be weak [28].

The intuition is that negations should reverse the degree of truth but preserve that of knowledge. The double negation law is dispensable; consider for instance intuitionistic logic. Obviously, "*not*", as defined in Table 6 does not satisfy the first mandatory property.

The conjunction and disjunction operators are, as usual, the meet and join in the truth-ordering. Implication is two-valued and defined (as in our logic $\mathcal{FOUR}$) in order to guarantee validity of

$$I \models_7 A \leftarrow B \text{ iff } I \models_7 A \text{ or } I \not\models_7 B$$

Several theorems of classical logic are not valid in **VII**. Namely the excluded middle and the contraposition laws are not verified for both forms of negation, the law of contradiction is not valid for explicit negation (but holds for default negation), and De Morgan's laws do not hold for *not* . The implication connective $A \leftarrow B$ is not definable in terms of the classical relation $A \vee \neg B$ (or $A \vee not\, B$). Furthermore, Modus Tollens is invalid for both forms of negation but nevertheless Disjunctive Syllogism still holds in **VII** for default negation. Another important property which is not valid is the coherence principle of [61, 4]:

$$\text{If } I \models_7 \neg F \text{ then } I \models_7 not\, F$$

for any formula F.

We defer the discussion of this principle to the next section. Other fundamental equivalences of classical logic are true in **VII**, namely the associative, distributive, commutative, idempotent, absorption, double negation, and zero/one laws. Also, the Deduction Theorem and Modus Ponens hold. Furthermore, the two negations *not* and $\neg$ are interchangeable, i.e.

$$not\,\neg A \models\!\models_7 \neg not\,A$$

One can easily conclude that **VII** is neither classical logic, intuitionistic logic, nor da Costa's C_ω system.

Another interesting aspect of logic **VII** is difference in the two notions of equivalence ($\equiv_7$ and $\models\!\models_7$). It is immediate that

$$\begin{array}{c} A \vee (B \wedge C) \not\equiv_7 (A \vee B) \wedge (A \vee C) \\ A \wedge (B \vee C) \not\equiv_7 (A \wedge B) \vee (A \wedge C) \\ A \wedge not\,A \not\equiv_7 \mathbf{f} \end{array}$$

This means that the distributive laws and the contradiction law for default negation are not valid under the stronger form of equivalence $\equiv_7$. For all other properties mentioned above the two forms of equivalence coincide.

The relationship between the extended well-founded model and a **VII** interpretation is given by the following theorem.

Theorem 22. [78] *Given the extended well-founded model M_P of a program P construct a* **VII** *interpretation as follows, where A is an arbitrary atom:*

- *If A and $\neg A$ belong to M_P then $I(A) = \top$;*
- *else, if $A \in M_P$, respectively $\neg A \in P$, then $I(A) = \mathbf{t}$, respectively $I(A) = \mathbf{f}$;*
- *else, if $not\,A$ and $not\,\neg A$ belong to M_P then $I(A) = *$;*
- *else, if $not\,A \in M_P$, respectively $not\,\neg A \in M_P$, then $I(A) = \mathbf{df}$, respectively $I(A) = \mathbf{dt}$;*
- *otherwise, $I(A) = \bot$.*

Then, interpretation I is a **VII** *model of program P. The relation is patent in Figure 4.*

However this seven-valued logic is not the natural logic for *EWFS*. We believe that a nine-valued does a better job since, according to *WFS*, a literal is assigned one of three logical values: false, undefined, or true.

This means that one can assign nine different combinations of logical values to the pair of literals $A/\neg A$, which correspond to the following nine interpretations in extended well-founded semantics:

$$\begin{array}{ccccc} \{\} & \{A\} & \{\neg A\} & \{A, \neg A\} & \{notA\} \\ & \{A, not\neg A\} & \{\neg A, notA\} & \{notA, not\neg A\} & \{not\neg A\} \end{array}$$

But if we have nine combinations and seven logical values some grouping must take place in logic **VII**. Indeed, the interpretations $\{A\}$ and $\{\neg A\}$ are grouped, respectively, with $\{A, not\,\neg A\}$ and $\{\neg A, not\,A\}$, corresponding to **VII**'s logical values *true* and $false$. This seems to us undesirable, and can lead to odd behaviour as shown in the following example.

Example 11. Consider the following two extended logic programs:

$$P_1 = \left\{ \begin{array}{l} b \leftarrow not\, a \\ \neg a \end{array} \right\} \qquad\qquad P_2 = \left\{ \begin{array}{l} b \leftarrow not\, a \\ a \leftarrow not\, a \\ \neg a \end{array} \right\}$$

In P_1, a and b are assigned, by Sakama's extended well-founded semantics, the logical values false and true, respectively. In the second program a is still assigned the truth value false, but "oddly" now b is no longer true: it has become default true. Semantically, nothing should have changed since a kept its truth value.

4.2 Paraconsistent Well-Founded Semantics with Explicit Negation

Both semantics in the previous section see default negation and explicit negation as unrelated. To overcome this situation a new semantics for extended logic programs was proposed in [61, 4]. 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 $not\, \neg L$). In Example 10, literal $\neg a$ is true but $not\, a$ is not entailed under *EWFS*. With *WFSX* the more intuitive model $\{\neg a, b, not\, a, not\, \neg b\}$ is obtained. More recently, a paraconsistent extension of this semantics ($WFSX_{\mathrm{p}}$) has been proposed in [3, 19] 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* [31], *WFSX*, and $WFSX_{\mathrm{p}}$.

Definition 23. [34] Let P be an extended program, I a two-valued interpretation. The GL–transformation produces the reduct $\frac{P}{I}$, i.e. 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 24. [61, 4] The semi-normal version P_s of a program P is obtained from P by adding to the (possibly empty) *Body* of each rule $L \leftarrow Body$ the default literal $not\, \neg L$, where $\neg L$ is the complement of L 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 25. [3] *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 26. [3]
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 $not\,\neg L$. This once more corresponds to the simultaneous use of the C-scheme and N-scheme. Therefore $WFSX_{\mathrm{p}}$ generalizes $WFSX$, as it does not impose this additional condition. Furthermore, for normal logic programs, it coincides with WFS.

Example 12. Let P be the extended logic program P:

$$P = \left\{ \begin{array}{lll} c \leftarrow not\,b & a & d \leftarrow not\,d \\ b \leftarrow a & \neg a & \end{array} \right\}$$

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

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

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

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, $\neg a$, b and 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 return to this issue later and see its usefulness for detecting dependencies in contradictory information.

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 [24, 92] (see also [9]), which doubles the number of rules of the program: one takes care of computing the Γ_s part and the other the Γ part:

Definition 27. 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 13. 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:

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

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 28. *Let P be an extended logic program. Then the following equivalence holds for an arbitrary atom a:*

- $a \in WFM_{\rm p}(P)$ *iff* $a^p_T \in WFM(P^{T-TU})$;
- $\neg a \in WFM_{\rm p}(P)$ *iff* $a^n_T \in WFM(P^{T-TU})$;
- $not\, a \in WFM_{\rm p}(P)$ *iff* $not\, a^p_{TU} \in WFM(P^{T-TU})$;
- $not\, \neg a \in WFM_{\rm p}(P)$ *iff* $not\, a^n_{TU} \in WFM(P^{T-TU})$.

Now we address a main criticism leveled at the original definition of well-founded semantics with explicit negation regarding its model theory. In [61, 4] the definition of the logical implication operation is not truth-functional and literals a and $\neg a$ are viewed almost as separate entities: though $\neg a$ entails a false, nothing else follows for other truth values. Furthermore, in some cases the head of a rule can be assigned the truth value **f**, the body **u**, and the rule ($\mathbf{f} \leftarrow \mathbf{u}$) be satisfied, namely when the head of the rule is made explicitly false

by some other rule, a phenomenon known as overriding (see Example 14 below). This is due to the coherence principle, and it might be considered awkward in a three-valued logic. In other cases, where coherence does not intervene, the rule is unsatisfiable, thus showing the non truth-functional character of the model definition used.

Example 14. Consider the following extended logic program

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

The model of this program according to *WFSX* and $WFSX_{\mathrm{p}}$ is $\{not\, \neg a, \neg b, not\, b\}$, i.e. the undefinedness of a in the rule for b is overriden by the explicit falsity of b. Compare with the model obtained with *EWFS* where the set of conclusions is simply $\{not\, \neg a, \neg b\}$.

In [20, 19] we have provided a nine-valued truth-functional model theory for $WFSX_{\mathrm{p}}$, based on Ginsberg's bilattices concept [37], which is displayed in Figure 5.

The set of truth-values is $\{\bot, \mathbf{df}, \mathbf{dt}, \mathbf{f}, \mathbf{t}, \mathbf{I}, \mathbf{II}, \mathbf{III}, \mathbf{IV}\}$. The consequence relation $\models_9$ is obtained as per Definition 4 by letting the designated truth-values be **t**, **II** and **I**. By $\bot$ we mean no opinion (or undecidedness or undefinedness), and it corresponds to the bottom element of the lattice according to the knowledge ordering. The two additional logical values correspond to default falsity (**df**) and default truth (**dt**). We have also the classical **t** and **f** values, and four degrees of contradictory information. The truth value **IV** should be understood as contradictory belief, i.e. **df** and **dt**. The higher truth values **II** and **III** in the knowledge ordering can be understood as truth plus contradictory belief and falsity plus contradictory belief, respectively. Finally, **I** is the contradictory truth-value.

It might be controversial to say that **IV** is contradictory. It rather depends on the interpretation of the *not* operator on the logic programming side. Some works [74, 79, 80] interpret $not\, F$ as the belief in the falsity of F, formally $\mathcal{B}\neg F$. Consequently, $not\, a$ and $not\, \neg a$ respectively entail $\mathcal{B}\neg a$ and $\mathcal{B}a$, i.e. in truth-vale **IV** one simultaneously believes in the falsity of $\neg a$ and in the truth of a. This form of inconsistency is known in the paraconsistency literature as epistemic contradiction, which does not raise any problem from a conceptual point of view [76, 68, 42]. In other works [33, 73, 14, 2] the reading of the default negation operator is $\neg\mathcal{B}$ and therefore the simultaneous truth of $not\, a$ and $not\, \neg a$ is understood as $\neg\mathcal{B}a$ and $\neg\mathcal{B}\neg a$, i.e. in truth-value **IV** one does not believe in neither the truth of a nor the falsity of a, which is not inconsistent unless one insists on the logic programming side that a is two valued.

We adhere to the former interpretation of the *not* operator, which is in accordance with our intuitive reading of literals, and allows for three-valued interpretations of logic programs.

Disjunction and conjunction are defined, respectively, as the meet and the join in the truth-ordering (from left to right in the figure, **f** being the bottom element and **t** the top one). We define the two negation operators according to the truth-table depicted in Table 7.

Table 7. Truth table for negations in logic $\mathcal{NINE}$

A	$\bot$	**df**	**dt**	**f**	**t**	**I**	**II**	**III**	**IV**
$\neg A$	$\bot$	**dt**	**df**	**t**	**f**	**I**	**III**	**II**	**IV**
$not\, A$	$\bot$	**t**	**f**	**t**	**f**	**I**	**I**	**I**	**I**

The rationale is that $\neg$ finds the degree of knowledge conviction about the negation of a proposition, in exchanging the roles of what counts for and what counts against it, with respect to the $\bot/\top$ axis. The *not* operator determines if the negation of a proposition is at least believed. It is immediate that $\neg$ is a negation operator, and that *not* is a weak negation operator, in the sense of Definition 21. Finally, rule implication is two-valued and defined as before: $A \leftarrow B$ is **f** iff B has one of the designated truth-values and A has none of them; otherwise, $A \leftarrow B$ is **t**.

A remarking feature of $\mathcal{NINE}$ is the set of classical truths which are still obeyed. In particular, all of the associative, distributive, commutative, idempotent, absorption, zero/one, and De Morgan's laws hold in the logic $\mathcal{NINE}$. The double negation law for explicit negation is obeyed. Even though double negation law fails for *not* the intuitionistic property $A \rightarrow not\, not\, A$ is a theorem of $\mathcal{NINE}$. Also, the Deduction Theorem and Modus Ponens hold. Interesting too are the properties which are not valid: the excluded middle principle, the contradiction law, the contraposition law and the classical definition of implication in terms of $\neg$ or of *not*. In fact, the first two properties are not desired in a paraconsistent logic for well-founded based semantics. Furthermore, Modus Tollens and Disjunctive Syllogism are not sound rules with respect to $\mathcal{NINE}$. We finally should mention that the negation symbols *not* and $\neg$ are interchangeable, i.e. $not \neg A \models\!\models_9 \neg not\, A$. Obviously, the coherence principle, if $I \models_9 \neg L$ then $I \models_9 not\, L$, is valid in $\mathcal{NINE}$. As for the logics discussed before, $\mathcal{NINE}$ does not coincide with any of the classical, intuitionistic, or C_ω systems.

The adequateness of $\mathcal{NINE}$ as the underlying model theory for $WFSX_{\mathrm{p}}$ is provided by the following theorem:

Theorem 29. [20, 19] *Let P be an extended logic program and M a partial paraconsistent stable model of P. Let I be the $\mathcal{NINE}$-interpretation and A an arbitrary atom in the language of P:*

$$
\begin{array}{lll}
I(A) = \bot & \text{iff} & A \notin I \text{ and } \neg A \notin I \text{ and not } A \notin I \text{ and not } \neg A \notin I \\
I(A) = \mathbf{df} & \text{iff} & A \notin I \text{ and } \neg A \notin I \text{ and not } A \in I \text{ and not } \neg A \notin I \\
I(A) = \mathbf{dt} & \text{iff} & A \notin I \text{ and } \neg A \notin I \text{ and not } A \notin I \text{ and not } \neg A \in I \\
I(A) = \mathbf{f} & \text{iff} & A \notin I \text{ and } \neg A \in I \text{ and not } A \in I \text{ and not } \neg A \notin I \\
I(A) = \mathbf{t} & \text{iff} & A \in I \text{ and } \neg A \notin I \text{ and not } A \notin I \text{ and not } \neg A \in I \\
I(A) = \mathbf{I} & \text{iff} & A \in I \text{ and } \neg A \in I \text{ and not } A \in I \text{ and not } \neg A \in I \\
I(A) = \mathbf{II} & \text{iff} & A \in I \text{ and } \neg A \notin I \text{ and not } A \in I \text{ and not } \neg A \in I \\
I(A) = \mathbf{III} & \text{iff} & A \notin I \text{ and } \neg A \in I \text{ and not } A \in I \text{ and not } \neg A \in I \\
I(A) = \mathbf{IV} & \text{iff} & A \notin I \text{ and } \neg A \notin I \text{ and not } A \in I \text{ and not } \neg A \in I
\end{array}
$$

Then, I is a $\mathcal{NINE}$ model of P.

To conclude, we have proven elsewhere [19] that appropriate extensions of Dix's desirable strong and weak properties of a semantics [23, 22] for the paraconsistent case are all obeyed by $WFSX_{\mathrm{p}}$. All strong principles, with the exception of *Cautious Monotony*, are verified by $WFSX_{\mathrm{p}}$. 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 adapted for the paraconsistent extended logic programming case.

4.3 Wagner's Vivid Knowledge Bases under liberal reasoning

Similarly to what was done in the previous section for definite extended logic programs with "*not*", Wagner introduces alternatively a non-monotonic negation operator, "$-$", which he names weak negation. The meaning of the weak negation operator is defined by extending the model and countermodel relations of Definition 15 induced by a partial interpretation with the following two rules:

$$\mathcal{M} \models -F \quad \text{iff} \quad \mathcal{M} \not\models F \qquad\qquad \mathcal{M} \models\!\!\!| - F \quad \text{iff} \quad \mathcal{M} \models\!\!\!| F$$

Notice that this rendering of weak negation has the flavour of "negation by failure" since $-F$ is true if we don't have F in a given partial interpretation $\mathcal{M}$. But as it stands some problems are immediately foreseeable. First, in every partial interpretation we have $\mathcal{M} \models a \vee -a$, i.e. weak negation is "two-valued". Furthermore, it is impossible to have simultaneously a and $-a$ true, i.e. $\mathcal{M} \models -(a \wedge -a)$.

Definition 30. [91] A vivid knowledge base (*VKB*) is a set of clauses of the form $l \leftarrow F$ where l is an atom a or its strong negation $\sim a$, and F an arbitrary formula constructed from the language defined by the logical operator symbols $\wedge$, $\vee$, $\sim$, $-$ and 1.

It is possible to transform an arbitrary *VKB* into a set of rules of the form $l \leftarrow E$, where E is a conjunction of literals (atoms or strong negation of atoms) and weakly negated literals. Vivid knowledge bases of this form Wagner calls extended deductive databases *XDBs* [90, 91]. So, without loss of generality, we will restrict our discussion to these.

Like two-valued normal logic programs *XDBs* do not have a single minimal model. Because of this, Wagner tries to define the consequences of a *XDB* proof-theoretically. Let X be a *XDB*, E a set (conjunction) of literals and weakly negated literals, and l a literal:

$$\begin{array}{ll}
(1) & X \vdash_l 1 \\
(--) & X \vdash_l --l \text{ if } X \vdash_l l \\
(\wedge) & X \vdash_l E \text{ if } \forall e \in E : X \vdash_l e \\
(-\wedge) & X \vdash_l -E \text{ if } \exists e \in E : X \vdash_l -e \\
(l) & X \vdash_l l \text{ if } \exists (l \leftarrow E) \in X : X \vdash_l E \\
(-l) & X \vdash_l -l \text{ if } \forall (l \leftarrow E) \in X : X \vdash_l -E
\end{array}$$

This form of reasoning is called liberal since the truth or weak falsity of $\tilde{l}$ (the strong negation complement of l) does not affect the truth or weak falsity of l. Unfortunately some syntactic restrictions must be enforced so that the inference relation can be defined in such a recursive manner (besides assuming that the language does not have function symbols).

Example 15. Consider the *XDB* $X = \{a \leftarrow b \wedge c; b \leftarrow a; c \leftarrow 1; d \leftarrow -d\}$. For X we have the following infinite derivations:

$$\begin{array}{ccc}
X \vdash_l a & X \vdash_l -a & X \vdash_l d \\
| & | & | \\
X \vdash_l b \wedge c & X \vdash_l -(b \wedge c) & X \vdash_l -d \\
| & | & | \\
X \vdash_l b \text{ and } X \vdash_l c & X \vdash_l -b \text{ or } X \vdash_l -c & X \vdash_l d \\
| & | & \vdots \\
X \vdash_l a \text{ and } X \vdash_l 1 & X \vdash_l -a \text{ or } X \vdash_l -1 & \\
| & | & \\
X \vdash_l a & X \vdash_l -a & \\
\vdots & \vdots &
\end{array}$$

To avoid unending derivations and guarantee that a unique preferred model is obtained Wagner imposes syntactic restrictions on theories such that the intended conclusions of the *XDB* can be defined proof-theoretically: well-foundedness and weak well-foundedness. The former discards programs with positive loops or loops through weak negation. The latter allows positive loops while rejecting negative loops over weak negation. For the case of weakly well-founded *XDBs* Wagner defines the concept of perfect model, which is the immediate translation of Przymusinski's perfect model notion [70] into the *XDB* setting. Given this notion, Wagner then defines a new inference system (let us designate it by

$\vdash_l^{loop}$) which is adequate with respect to the perfect model $\mathcal{M}_X$, i.e. $\mathcal{M}_X \models F$ iff $X \vdash_l^{loop} F$. Inference system $\vdash_l^{loop}$ is $\vdash_l$ plus loop-checking, failing all positive cycles. Furthermore, for a well-founded *XDB* we have $X \vdash_l^{loop} F$ iff $X \vdash_l F$.

The translation of weakly well-founded *XDBs* to logic programs is immediate, via the usual isomorphism.

Definition 31. Let X be an *XDB*. Normal logic program P^l is constructed from X by substituting every occurrence of 1, a, $\sim a$, $-a$, and $- \sim a$, respectively by true, a^p, a^n, $not\, a^p$, and $not\, a^n$, where a is an arbitrary atom.

It is immediately recognizable that an *XDB* X is well-founded iff P^l is acyclic [8] and weakly well-founded iff P^l is locally stratified [70]. Therefore P^l is locally stratified and has a unique perfect model which is equivalent to the well-founded model of P^l [69]. It is not difficult to see that the following correspondence holds:

Theorem 32. *Let X be a weakly well-founded* XDB. *Then the following equivalences hold, where a is an atom in the language of X:*

$$\begin{array}{ll} \mathcal{M}_X \models a & \textit{iff } a^p \in WFM(P^l) \\ \mathcal{M}_X \models -a & \textit{iff } not\, a^p \in WFM(P^l) \\ \mathcal{M}_X \models \sim a & \textit{iff } a^n \in WFM(P^l) \\ \mathcal{M}_X \models - \sim a & \textit{iff } not\, a^n \in WFM(P^l) \end{array}$$

The main conclusion is that Wagner arrived at a semantics which is a restricted case of Sakama's extended well-founded semantics. This is most interesting because Wagner started from paraconsistent constructive logics, thus formally justifying some of the ideas behind extended logic programming semantics. Unfortunately, the non-weakly well-founded programs are not taken care of, and we still have to find out if there are intuitions in the mathematical logic field which might justify three-valued extended logic programming semantics. A suggestion by Wagner consists in extending partial interpretations with two more sets, M^{df} and M^{dt} where the truth of $-a$ and $- \sim a$ are checked by testing, respectively, if $a \in M^{df}$ and $a \in M^{dt}$. The only condition required according to Wagner is default consistency:

$$M^{df} \cap M^t = \{\} \qquad \text{and} \qquad M^{dt} \cap M^f = \{\}$$

Then a partial interpretation $\langle M^t, M^f \rangle$ in the sense of Wagner would have an equivalent 4-place interpretation N with $N^t = M^t$, $N^f = M^f$, $N^{dt} = At - M^f$ and $N^{df} = At - M^t$, where At is the set of atoms of the underlying language. Coherence is expressed by the condition:

$$M^t \subseteq M^{dt} \qquad \text{and} \qquad M^f \subseteq M^{df}$$

But default consistency and coherence entail $M^t \cap M^f = \{\}$, and so no form of paraconsistency is allowed when we have both.

Thus his liberal semantics foregoes the coherence principle instead of default consistency, since it must allow $M^t \cap M^f \neq \{\}$. For instance, if $a \leftarrow 1$ and $\sim a \leftarrow 1$ simultaneously belong to an XDB then the weak negations $-a$ and $- \sim a$ are not entailed. In general, the coherence principle could be obeyed only if there is no pair of literals a and $\sim a$ true in the model. This might make sense in Wagner's setting since there is not the need to override "undefined" literals mentioned earlier (cf. Example 14) as by construction there are no undefined atoms! But for our three-valued based $WFSX_{\mathrm{p}}$ semantics this overriding seems desirable, and even intuitive. However, by our previous results, if coherence is not desired then a simple program transformation suffices to impede it: the $P^{\neg}$ transformation of Definition 20. So, on the one hand we have a semantics with an interpretation of weak negation as pure "negation by failure" which does not assign semantics to every program and cannot obey coherence. On the other, we have a semantics which assigns meaning to every extended logic program, with an intuitive coherence principle which can be disabled if wanted. We opt for the second approach and so drop the default consistency axiom instead, i.e. we think coherence is more fundamental than default consistency: if we allow one form of paraconsistency, on objective literals, why not allow it on subjective ones too ?

5 Extended Logic Programs: AS based semantics

The other semantics mainstream for extended logic programs originates in the Answer Sets semantics. Its main feature is two-valuedness, i.e. either L or $not\, L$ holds. According to Wagner's terminology this makes it a weak negation interpretation of default negation. In contrats to Wagner's liberal reasoning none of the semantics overviewed in this section are restricted to the class of locally stratified programs. We first discuss a direct generalization of Answer Sets Semantics [35], independently proposed by several authors: Stable Environments [66], Weak Answer Sets [57] and Paraconsistent Stable Models [80]. Next we treat Sakama and Inoue's Semi-Stable Models, which assign meaning to every extended logic program, and coincide with the Paraconsistent Stable Models whenever the latter exist. Finally, we briefly present the OD-semantics of Grant and Subrahmanian [40].

5.1 Paraconsistent Stable Models

We start by presenting the generalization of Answer Sets to the paraconsistent case proposed independently by [66, 57, 80]. The semantics is defined for the disjunctive case, but we restrict the discussion to disjunctive free programs. For the purpose of simplifying further comparisons, we use the terminology of [80], namely the term Paraconsistent Stable Models[6].

In what follows interpretations are now sets of objective literals.

[6] We would prefer the term "Paraconsistent Answer Sets".

Definition 33. Let P be an extended logic program and I an interpretation. I is a paraconsistent stable model (shortly, p-stable model) of P iff $I = \Gamma_P I$.

Example 16. Consider program $\{\neg a; a \leftarrow not\, b\}$. This program has the single p-stable model $\{a; \neg a\}$, coinciding with the paraconsistent well-founded model.

The relationship to *WFS* is immediate and again established through the program transformation $P^{\neg}$ of Definition 20.

Theorem 34. *Let P be an extended logic program. Then I is a p-stable model of P iff $(I \cup not\, \{L \mid L \notin I\})^{\neg}$ is a partial stable model of $P^{\neg}$ under* WFS *semantics.*

As remarked in [80], Paraconsistent Stable Models are incomparable to Answer Sets. However the authors provide a relationship between these two semantics in the same article. It uses the program P_{tr}, obtained by adding to an extended logic program P a set of axioms of the form $N \leftarrow L, \neg L$ (where L and N are arbitrary objective literals), thus "implementing" the trivialization rule of Answer Set semantics. The p-stable models of P_{tr} are in one-to-one correspondence with the answer sets of P. This corresponds to the adoption of the C-scheme and N-scheme described in the introduction of this volume. By dropping the C-scheme in Answer Sets Semantics one gets Paraconsistent Stable Models (or Stable Environments or Weak Answer Sets).

In [61, 4] it was shown that *WFSX*, and therefore $WFSX_{\mathrm{p}}$, are a good approximation to Answer Sets semantics, i.e. the (paraconsistent) well-founded model is contained in the intersection of all answer sets (whenever these are defined). Obviously, if the paraconsistent well-founded model has a pair of contradictory literals L and $\neg L$ then if an answer set exists it must be the set of all literals (Lit). What is important to notice is that the paraconsistent well-founded model and p-stable models are, in general, incomparable.

Example 17. Consider the extended logic program P:

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

Its paraconsistent well-founded is $\{\neg a, b, not\, a, not\, \neg b\}$. The same program has exactly two p-stable models, namely $\{a, \neg a\}$ and $\{\neg a, b\}$. The first p-stable model is incomparable to the above well-founded model. Notably, the coherence principle is not enforced by Sakama and Inoue, and hence the disparate results.

This example shows that $WFSX_{\mathrm{p}}$ cannot be used as an approximation to p-stable models. However the following interesting property holds:

Proposition 35. *Let P be an extended logic program. If I is a p-stable model of P then there is a paraconsistent partial stable model of P containing I.*

Example 18. Consider again the program of Example 17. A PSM_p of P is given by the interpretation $\{a, \neg a, b, not\, a, not\, \neg a,\ not\, b, not\, \neg b\}$. Notice that p-stable model $\{a, \neg a\}$ is contained in it and that the only other p-stable model coincides with the paraconsistent well-founded model of the program.

By means of simple arguments it can be shown that the $P^{\neg}$ program under *WFS* gives better limits to the intersection of p-stable models than the original program P under $WFSX_{\mathrm{p}}$ semantics. Whenever the set of all literals of P is not an answer set the following inclusions hold:

$$\begin{array}{ccc} gfp(\Gamma_P\Gamma_P) & \subseteq & gfp(\Gamma_P\Gamma_{P_s}) \\ \bigcup\!| & & \bigcup\!| \\ \bigcup \text{p-stable models} & & \bigcap \text{answer sets} \\ \bigcup\!| & & \bigcup\!| \\ \bigcap \text{p-stable models} & & \bigcup \text{answer sets} \\ \bigcup\!| & & \bigcup\!| \\ lfp(\Gamma_P\Gamma_P) & \subseteq & lfp(\Gamma_P\Gamma_{P_s}) \end{array}$$

In the case that *Lit* is an answer set the inclusion $\bigcup$answer sets $\subseteq gfp(\Gamma_P\Gamma_{P_s})$ is no longer valid.

The interesting point is that the inclusions pertaining to the p-stable models are related to the fixpoints of $\Gamma\Gamma$, while the ones for answer sets are related to the fixpoints of $\Gamma\Gamma_s$. The justification for this behaviour relies on the fact that p-stable models do not obey coherence whereas answer sets do (with the exceptional case of programs having *Lit* as its single answer set). Recall once more that the inner Γ_s step enforces the coherence principle. The inclusions for answer sets semantics are destroyed only in the case of non-coherent answer sets, i.e. for the set of all literals.

For some programs p-stable models do not exist, due to its semantics' answer sets basis. For instance, a program for which p-stable models do not exist is $\{a \leftarrow not\, a\}$. Paraconsistent Stable Models inherits other difficulties of Answer Sets, namely failure of *Cumulativity*, *Rationality*, and *Relevance* [23, 22]. On the bright side, this semantics has important known relations with constructive logics [56, 57, 58], besides those with default theories and autoepistemic logics of Stable Models (see [50, 9, 10]). We should also mention that Paraconsistent Stable Models coincide with Wagner's perfect model whenever the latter is defined (see Section 4.3).

Sakama and Inoue provide as well a model theory which is an extension of $\mathcal{FOUR}$, incorporating default negation (see Figure 6). In their logic **IV** the connectives $\wedge$, $\vee$, $\leftarrow$ and $\neg$ are interpreted as in logic $\mathcal{FOUR}$ of Section 3.2, and therefore satisfy the same properties. The truth-table for *not* is presented in Table 8.

It is clear that default negation defined as in logic **IV** is neither a negation nor a weak negation in the sense of Definition 21. Surprisingly, if we analyse the properties of *not* with respect to the consequence relation induced by the designated truth-values **t** and $\top$, we conclude that *not* behaves like classical negation !

[pth]

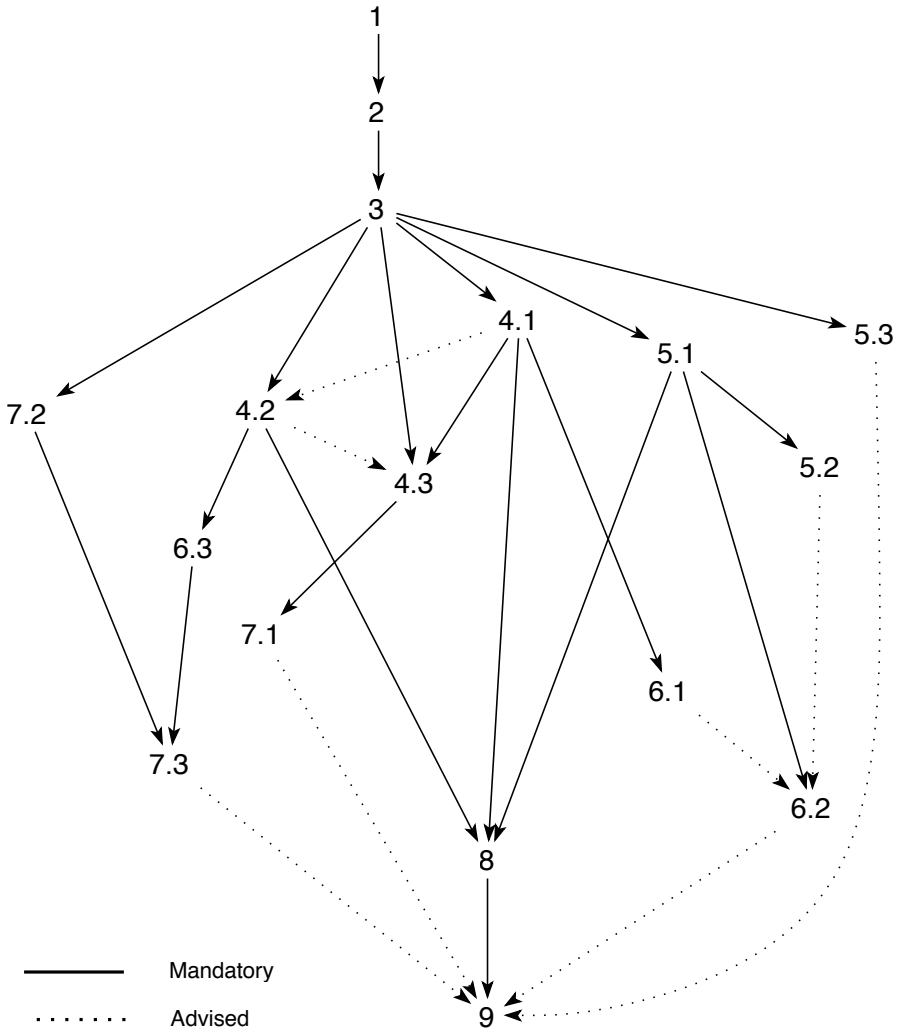

Fig. 1. Dependencies among sections

Table 8. Truth table for default negation

A	$\bot$	**f**	**t**	$\top$
not A	$\top$	**t**	**f**	$\bot$

[ht]

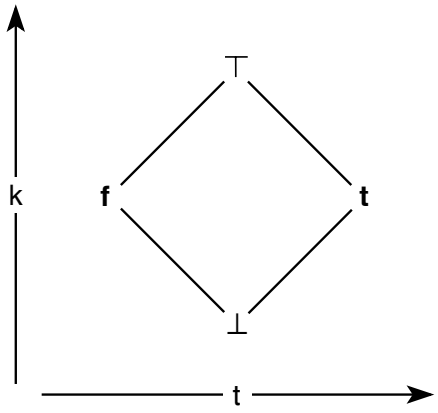

Fig. 2. Belnap's truth-space

[ht]

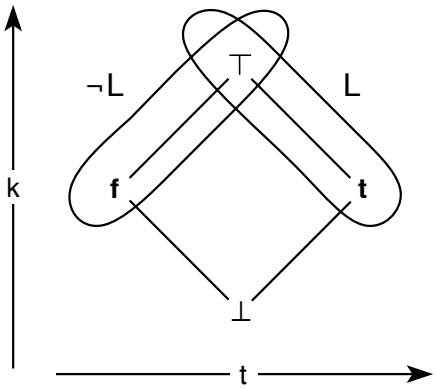

Fig. 3. Correspondence between the consequence relation and truth-assignment

Let us explain this result and its consequences. First, the designation of "two-valued" negation for the *not* of stable models and answer sets is fully justified. Our conclusion agrees with the known result that the stable models of a normal logic program are classical models of the corresponding propositional theory. Second, one cannot simply classify an unary operator as a negation by looking at its truth-table and associated knowledge and truth-orderings. The set of designated truth-values has a primary influence in the set of consequences derivable from the multi-valued logic. Last but not least, this allows us to explain

[ht]

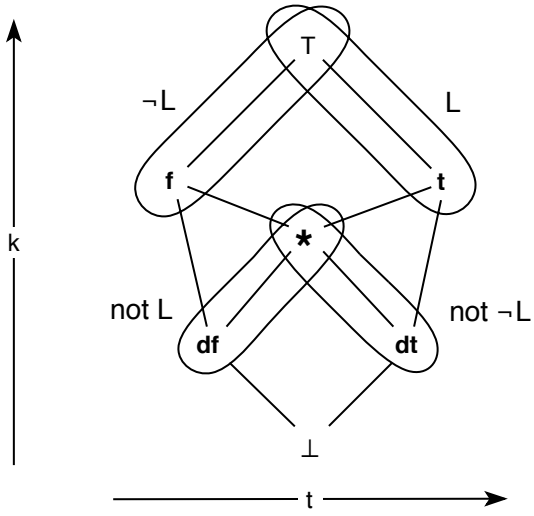

Fig. 4. Ginsberg's Logic **VII**

[ht]

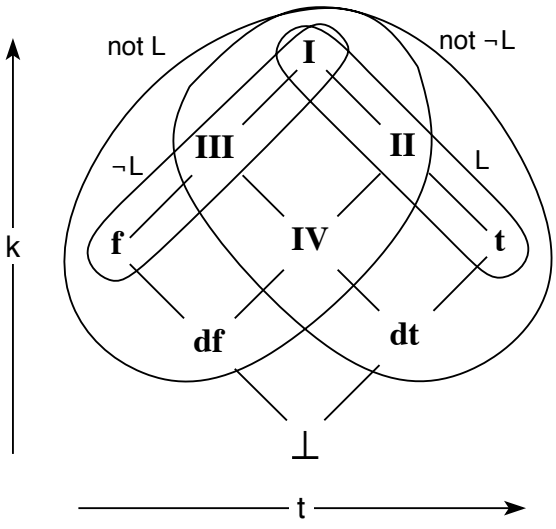

Fig. 5. Logic $\mathcal{NINE}$.

[ht]

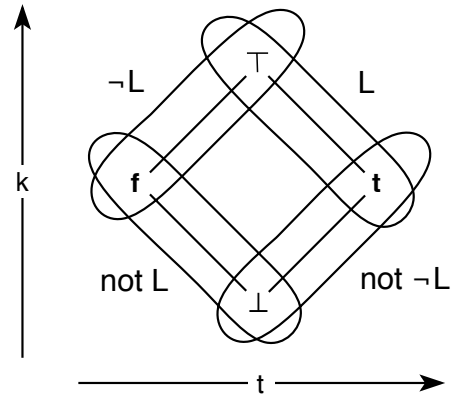

Fig. 6. Sakama and Inoue's logic **IV**

a frequent misunderstanding of logic programming semantics: even though the underlying logic may be monotonic the resulting semantics might not. In fact, the whole point of logic programming semantics is to single out a particular model (or models). The process involved may (desirably) destroy the monotonicity of the underlying model-theory. For instance, the models of the completion of a normal logic program are classical ones, but as we know the resulting semantics of negation by failure is non-monotonic.

5.2 Semi-Stable Models

To overcome those situations where p-stable models do not exist, Sakama and Inoue define the notion of Semi-Stable Models [80]. The main idea consists in translating (disjunctive) extended logic programs into positive disjunctive extended logic programs, and then to use a semantics for this restricted type of programs. Again, for comparability, we will limit the discussion to the case of non-disjunctive extended logic programs.

Definition 36. [80] Let P be an extended logic program. Its epistemic transformation is defined as the positive disjunctive extended logic program P^{κ} obtained from P by replacing each rule of the form:

$$L_0 \leftarrow L_1, \ldots, L_m, not\, L_{m+1}, \ldots, not\, L_n \qquad (0 \leq m \leq n)$$

with the following not-free rules in P^{κ}:

$$\lambda \vee \mathbf{K}L_{m+1} \vee \ldots \vee \mathbf{K}L_n \leftarrow L_1, \ldots, L_m.$$

$$L_0 \leftarrow \lambda.$$
$$\leftarrow \lambda, L_j.\ (m+1 \leq j \leq n)$$

Each already not-free rule in P is included in P^κ as it is.

To each disjunctive rule in P^κ a different λ is associated. The $\mathbf{K}L_j$ literals are new atoms in the language of P^κ, with the reading "L_j is believed", while *not* L_j has the meaning "L_j is disbelieved". These $\mathbf{K}L_j$ literals can be shared by different rules in the program. To give semantics to such P^κ programs the meaning of disjunction in the heads and of integrity constraints must be defined. Sakama and Inoue use the split program construction to achieve this:

Definition 37. [80] Given a positive extended disjunctive program P, a split program is defined as the positive extended logic program obtained from P by replacing each disjunctive rule

$$L_1 \vee \ldots \vee L_l \leftarrow L_{l+1}, \ldots, L_m$$

with the following extended rules (called split rules):

$$L_i \leftarrow L_{l+1}, \ldots, L_m \text{ for every } L_i \in S$$

where S is some arbitrarily chosen non-empty subset of $\{L_1, \ldots, L_l\}$ for each disjunctive rule. Then a p-possible model is defined as the (unique) p-stable model of any split program P. Different choices for the S engender all the split programs.

Example 19. Consider the extended logic program of Example 17. Its corresponding positive disjunctive program P^κ is:

$$\begin{array}{lllll} \lambda_1 \vee \mathbf{K}b. & a \leftarrow \lambda_1. & \leftarrow \lambda_1, b. & \neg a. \\ \lambda_2 \vee \mathbf{K}a. & b \leftarrow \lambda_2. & \leftarrow \lambda_2, a. & \end{array}$$

The program P^κ has nine split programs but only five have p-stable models. The corresponding p-possible models are:

$$p_1 = \{\lambda_1, a, \mathbf{K}a, \neg a\},$$
$$p_2 = \{\lambda_1, a, \mathbf{K}a, \mathbf{K}b, \neg a\},$$
$$p_3 = \{\mathbf{K}a, \mathbf{K}b, \neg a\},$$
$$p_4 = \{\lambda_2, b, \mathbf{K}b, \neg a\} \text{ and}$$
$$p_5 = \{\lambda_2, b, \mathbf{K}b, \mathbf{K}a, \neg a\}.$$

Given this bunch of p-possible models, Sakama and Inoue introduce a preference criteria to select some of them, which are then used to define the Semi-Stable models of an extended logic program:

Definition 38. [80] Let P be an extended logic program and $\mathcal{I}_{P^\kappa}$ the set-inclusion minimal p-possible models of P^κ[7]. An interpretation $I^\kappa \in \mathcal{I}_{P^\kappa}$ is said to be maximally canonical iff there is no interpretation $J^\kappa \in \mathcal{I}_{P^\kappa}$ such that $\{\mathbf{K}L \mid \mathbf{K}L \in J^\kappa \text{ and } L \notin J^\kappa\} \subseteq \{\mathbf{K}L \mid \mathbf{K}L \in I^\kappa \text{ and } L \notin I^\kappa\}$. The semi-stable models of P are the maximally canonical interpretations in $\mathcal{I}_{P^\kappa}$ with the λ_i literals removed.

Example 20. Consider again program P of Example 19. The minimal p-possible models of P^κ are $p_1 = \{\lambda_1, a, \mathbf{K}a, \neg a\}$, $p_3 = \{\mathbf{K}a, \mathbf{K}b, \neg a\}$ and $p_4 = \{\lambda_2, b, \mathbf{K}b, \neg a\}$. The semi-stable models are $\{a, \mathbf{K}a, \neg a\}$ and $\{b, \mathbf{K}b, \neg a\}$ corresponding, respectively, to the p-possible models p_1 and p_4. In this case, the semi-stable models are isomorphic to the p-stable models of the program.

Example 21. Let P be the program $\{\neg a; b \leftarrow not\, b\}$. Clearly, P has no p-stable models, and the unique semi-stable model $\{\neg a, \mathbf{K}b\}$.

Two minimization processes are used to define the semi-stable models of an extended logic program. The first one is, as usual in disjunctive logic programming semantics, the selection of the minimal models. The second one is specific to Sakama and Inoue's semantics.

The authors show that p-stable models are the canonical interpretations in $\mathcal{I}_{P^\kappa}$, i.e. the interpretations such that for every $\mathbf{K}L \in I^\kappa$ then $L \in I^\kappa$ or, equivalently, for every $L \notin I^\kappa$ then $\mathbf{K}L \notin I^\kappa$. Suppose, for simplicity, that $not\, a$ is the single default negated literal in the body of a rule. The canonical condition guarantees that if a is false then $\mathbf{K}a$ is false, and therefore if the body is entailed the head entailed too, i.e. $not\, a$ is true. If a is true then the integrity constraints do not let the λ-literal be true, and therefore the head of the rule cannot be entailed. Essentially we have the following two conditions, where I is an interpretation:

- If $a \notin I$ then $not\, a$ is true, by the canonical condition;
- If $a \in I$ then $not\, a$ is false, by the integrity constraints on λ-literals.

It is clear that the second condition holding is always desirable. However, when p-stable models do not exist, the canonical condition is not satisfied. So the idea is to accept as models of programs the interpretations which least violate the canonical condition, those defined as the Semi-Stable Models. Notice that a canonical interpretation is also maximally canonical, and is also smaller than any other non-canonical interpretation, giving rise to the Paraconsistent Stable Models in that case.

A first "problem" of Semi-Stable Models is their non compliance with the coherence principle. The trivial example is program $\{a \leftarrow not\, a; \neg a\}$, having the unique semi-stable model $\{\neg a\}$ in contrast with the paraconsistent well-founded model $\{a, \neg a, not\, a, not\, \neg a\}$.

[7] This corresponds to the set-inclusion minimal models of P^κ viewed as a positive disjunctive logic program.

The treatment of undefined literals in Semi-Stable Models is also in some situations counter-intuitive, as illustrated in the following example:

Example 22. Consider the extended logic program $\{b \leftarrow not\, d; c \leftarrow d; d \leftarrow not\, c\}$. Its unique semi-stable model is $\{b, not\, \neg b, not\, \neg c, not\, d, not\, \neg d\}$. Thus, c is "undefined" but $not\, d$ is true and therefore b is also true. The semantics undefines only some of the literals involved in a negative loop. Notice first that we have an undefined value implying a false one, which seems strange for a normal logic program[8].

Second, assume we add the rule "$a \leftarrow b, not\, a$" to the above program. Now there are two semi-stable models: $\{not\, \neg a, b, not\, \neg b, not\, \neg c, not\, d, not\, \neg d\}$ and $\{not\, a, not\, \neg a, not\, b, not\, \neg b, not\, \neg c, not\, \neg d\}$. So, because of a rule "unrelated" to d, we must now accept a model where c and d are both undefined. We would expect instead the first semi-stable model to be the single result in a "well-behaved" semantics.

The treatment of "undefined" literals in Semi-Stable models seems a bit *ad hoc* and program-dependent. This semantics does not obey any of Dix's principles of *cumulativity*, *rationality*, or *relevance* [22, 23]. We would also expect that a paraconsistent semantics for arbitrary extended logic programs based on Answer Sets would be defined in terms of the maximal partial stable models (or preferred extensions) of program $P^{\neg}$.

Finally, it should be remarked that a nine-valued logic exists, **IX**, that provides a model-theory for Semi-Stable Models. Logic **IX** is isomorphic to our logic $\mathcal{NINE}$, but with a different interpretation of default negation. Its Hasse Diagram is presented in Figure 7.

The interpretation of these truth-values is as follows: **bf** and **bt** denote *believed false* and *believed true*, and are in accordance with the interpretation of the **K** literals; **b**$\top$, **fcb**, and **tcb** denote different degrees of contradictory information, the former means *believed contradictory* and **fcb** and **tcb** denote, respectively, *true with contradictory belief* and *false with contradictory belief*; the remaining four values correspond to Belnap's ones, namely undefined, false, true and contradictory.

As for $\mathcal{NINE}$, it is possible to define the truth-tables for the negation operators "$\neg$", "*not*" and for the implication sign. The first remark is that both "$\neg$" and "$\leftarrow$" have truth tables isomorphic to our connectives under logic $\mathcal{NINE}$, which supports the naturalness of both approaches (and also conjunction, disjunction and satisfaction). The essential difference is in the *not* operator. The most natural way of defining Sakama's *not* operator appears in Table 9, which verifies the double negation law $not\, not\, L = L$ and the De Morgan laws with respect to the consequence relation $\models_{IX}$ defined by the set of designated truth-values $\{\mathbf{t}, \mathbf{tcb}, \top\}$.

[8] $WFSX_{\mathrm{p}}$ has a similar behaviour, but only in those situations where we have an explicitly negated fact overriding an undefined value. This never occurs in normal logic programs.

[ht]

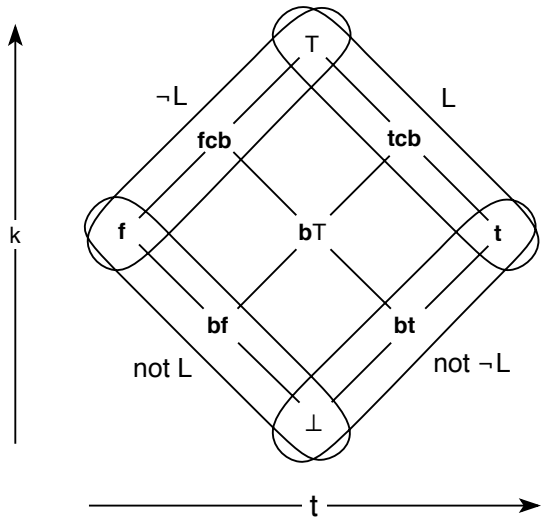

Fig. 7. Sakama and Inoue's logic **IX**

Table 9. Truth table for default negation in logic **IX**

A	$\bot$	**bf**	**bt**	**f**	**t**	**b**$\top$	**fcb**	**tcb**	$\top$
not A	$\top$	**tcb**	**fcb**	**t**	**f**	**b**$\top$	**bt**	**bf**	$\bot$

Furthermore, the associative, commutative, idempotent, distributive, absorption, zero/one, double negation, and De Morgan laws all hold in logic **IX**. The two forms of negation are interchangeable. As usual, the Deduction Theorem and Modus Ponens hold. The excluded middle principle is not obeyed by both forms of negation. The contradiction law is obeyed by default negation under the weaker form of equivalence relation (but not the stronger one $\equiv_{IX}$) while it is invalid for explicit negation. Modus Tollens is invalid for both negations and Disjunction Syllogism holds for default one but not for the explicit rendition. As we have referred before, coherence is not verified by this logic. An interesting remark is that *not* under consequence relation $\models_{IX}$ behaves as the three-valued Lukasiewickz's negation operator [49]. However, the definition of the implication operator is rather different.

Finally, the most important of Fitting's axioms (cf. Definition 21) for negation operators in interlaced bilattices is not obeyed in **IX**: the negation operator should reverse truth but never knowledge. Sakama's negation operator reverses

both at the same time. For instance we have **bf** $\leq_k$ **tcb** and **bf** $\leq_t$ **tcb** but *not* **tcb** $\leq_k$ *not* **bf** and *not* **tcb** $\leq_t$ *not* **bf**. Notice this is not due to our way of defining the truth table. It follows from the satisfaction of the double negation and De Morgan laws. This discussion also stresses the originality of $\mathcal{NIN}\mathcal{E}$'s *not* operator, which we have shown to be a weak negation operator in the sense of Fitting.

Finally, and for the sake of completeness, we present the mapping between the semi-stable models of a program and corresponding **IX** interpretations.

Theorem 39. [80] *Let* $\mathcal{L}^\kappa = OLit \cup \{\mathbf{K}L | L \in OLit\}$ *and* I^κ *a subset of* $\mathcal{L}^\kappa$. *Define a* **IX** *interpretation* I *from* I^κ *as follows, where* A *is an atom in the language of the program:*

$$\begin{aligned} I(A) = lub_k\{\ x \mid\ & x = \mathbf{t} \text{ if } A \in I^\kappa, \\ & x = \mathbf{f} \text{ if } \neg A \in I^\kappa, \\ & x = \mathbf{bt} \text{ if } \mathbf{K}A \in I^\kappa, \\ & x = \mathbf{bf} \text{ if } \mathbf{K}\neg A \in I^\kappa, \\ & x = \bot \text{ otherwise}\} \end{aligned}$$

If I^κ *is a semi-stable model of* P *then* I *is a* **IX** *model of* P.

The above theorem supports the terminology used for the several truth-values, in particular, the interpretation of the **K** literals in the semi-stable models.

5.3 Over-determined semantics

In [40] is presented a semantics for extended logic programs, incorporating ideas from [13] and from stable models. According to its authors the following three issues should be addressed by extended logic programs:

Reasoning with Inconsistency: A small inconsistency should not cause the entire database to be rendered useless;

Reasoning with the Law of Excluded Middle: The statement $A \vee \neg A$ must always be considered true. Hence, if we can reason that A implies B and $\neg A$ implies B then we should be able to conclude B;

Reasoning with Cases: Again, if we know that B or C is true, and we know that B implies D, and C implies D, then we should conclude D.

Even though the first and last conditions are are accepted by most paraconsistent semantics, the enforcement of the law of excluded middle for explicit negation is more controversial. No other semantics in this survey enforce such a requirement for explicit negation. The authors motivating example is the following:

Example 23. [40] Consider the following extended logic program:

$$masters(X) \leftarrow credits(X,Y), has\_thesis(X), exam(X,a), Y \geq 30.$$
$$masters(X) \leftarrow credits(X,Y), has\_thesis(X), exam(X,b), Y \geq 30.$$
$$masters(X) \leftarrow credits(X,Y), \neg has\_thesis(X), exam(X,a), Y \geq 36.$$
$$credits(jim, 36).$$
$$exam(jim, a).$$

As usual, we understand a non-ground program standing for its instantiated version. The first two rules say that any student who has completed a thesis and passed a comprehensive exam with either an A or a B grade, and has taken at least thirty credits of courses, is eligible for a master's degree. The third rule says that students who haven't done a thesis, but have taken 36 or more credits of courses and got an A on the comprehensive exam are also eligible for a master's degree. Jim has taken 36 credits in course work and got an A in the exam. Clearly, none of the semantics presented in Section 3 entail $masters(jim)$, which would be justified by the excluded middle law.

For handling these situations, and while being tolerant to inconsistency, the authors delimit the truth-space of logic $\mathcal{FOUR}$ to the upper three truth-values in the knowledge ordering: **f**, **t** and $\top$. The undetermined truth-value is discarded, thereby enforcing the law of the excluded middle. The notion of an interpretation I being a model of a formula F is the obvious restriction to be imposed on $\mathcal{FOUR}$, and is denoted by $I \models_{od} F$. An interpretation I is an OD-model of a definite extended logic program iff, for every rule R in P, $I \models_{od} R$.

Definition 40. [40] Let P be a definite extended logic program. The over-determined semantics for P is the set of all disjunctions of objective literals d, such that for every OD-model I of P we have $I \models_{od} d$.

Obviously, the over-determined semantics obeys the authors' three caveats stated at the beginning of this section. It should also be clear that every conclusion of the semantics given in Section 3 is again a conclusion of the OD-semantics.

Example 24. In the program of Example 23 $masters(jim)$ is an OD-consequence. An OD-model I of the program must entail $has\_thesis(jim)$ or $\neg has\_thesis(jim)$. In the first case I entails $masters(jim)$ by the first program rule. In the second, I derives $masters(jim)$ by application of the third rule. Thus, $masters(jim)$ is true in all OD-models, and hence an OD-consequence of the program of Example 23.

The extension of the OD-semantics for programs with default negation is achieved by means of the usual Gelfond-Lifschitz program division construction (cf. Definition 23).

Definition 41. [40] Let P be an extended logic program and X a set of objective literals. Define operator $\mathbf{O}_P$ as follows:

$$\mathbf{O}_P(X) = \{L \mid L \text{ is an objective literal such that } \tfrac{P}{X} \models_{od} L\}$$

A set of objective literals is an OD-answer set of P iff $\mathbf{O}_P(X) = X$.

Example 25. Consider the program obtained by adding the following rule to the program of Example 23:

$$interview(X) \leftarrow masters(X), not\, employed(X).$$
$$employed(sue).$$
$$masters(sue).$$

This program has exactly one OD-answer set, viz.

$$\{employed(sue), masters(sue),$$
$$interview(jim), exam(jim), credits(jim, 36), masters(jim)\}$$

The usual problems of answer-sets semantics carry over to OD-semantics, namely the inexistence of OD-answer sets for some programs. We conjecture that the restriction of Sakama and Inoue's logic **IV** to its upper three logic values (and mapping $not\ \top$ to **f**) provides an appropriate model theory for OD-answer sets semantics.

6 Detecting Support On Contradiction

We have reviewed several semantics for reasoning in the presence of contradiction with different perspectives and objectives. However, humans tend to be overly cautious when accepting conclusions in the presence of contradiction. So, we rationally require additional mechanisms supporting paraconsistent deductions, which give additional arguments for their validity. One obvious need is the ability to detect safe conclusions, i.e. those which do not depend on contradiction.

Our first effort is to clarify what we mean by support on contradiction, something which has often been neglected in the literature. Our definition is declarative and independent of the underlying semantics. We assume only the language of extended logic programs.

Definition 42. Let P be an extended logic program and $C = \{c, \neg c \mid c, \neg c \in SEM(P)\}$, the set of the contradictory facts in $SEM(P)$. We say that any literal L depends on contradiction with respect to semantics SEM iff there is $S \subseteq C$ such that:

$$SEM(P) \cap \{L\} \neq SEM(P - Rules(S)) \cap \{L\}$$

where $Rules(S)$ is the set of rules of P with head in S.

The rationale is clear-cut. We say that L depends on contradiction iff by removing some subset of the contradictory facts in the program's truth-value of L gets changed. Mark the implicit adoption of the N-scheme in the construction of set C as a means for defining the notion of "contradiction".

Example 26. Suppose a bus driver is approaching a railway crossing where the barriers are lifted but the warning lights are on. The dilemma the bus driver faces can be expressed by the following extended logic program:

$$cross \leftarrow \neg train. \qquad train. \qquad \neg train.$$

According to the semantics given in Section 3 the model of this program is

$$\{cross, not\ \neg cross, train, \neg train\}$$

Thus the semantics advises the driver to cross the rails. This is a suicidal semantics for basing behaviour, unless additional information is provided. By making use of Definition 42 we can detect the support of conclusion *cross* on the contradictory pair $train/\neg train$: by removing the set of facts $\{\neg train\}$ or $\{train, \neg train\}$ from the above program conclusion *cross* is no longer entailed by the model of the resulting programs.

Definition 42 is readily applicable to any semantics. However, the process described is exponential, and useless in practice. Few semantics in the literature solve or even address this problem. In this section, we present the extensions of some of the semantics previously surveyed which provide additional information regarding support on contradiction, according to Definition 42. In particular, we will discuss Sakama's suspicious well-founded and stable semantics [78, 80] and our own $WFSX_{\mathrm{p}}$ and its extension described in [21]. More careful semantics in dealing with contradiction will be studied in the next section.

6.1 Sakama's suspicious well-founded semantics

Sakama describes another variant of his *EWFS* semantics (see Section 4.1) which can return information about the support of a literal, i.e. whether or not the truth of a literal depends on inconsistent information. Those literals that are supported on contradictory literals are called suspicious, otherwise they are called safe. Technically this is achieved by keeping track of all the literals involved in the proofs of every literal in the bottom-up computation of the well-founded model. We only present the basic definitions in order to discuss their approach. For more details consult [78].

The author requires the new notion of suffixed literal L^{Σ}, where L is an objective literal in the language of an underlying program and Σ is a set of sets of literals. The idea is to use Σ to keep track of the literals involved in the proofs for L. As we see in the next definition the construction used is rather operational and asymmetric:

Definition 43. [78] Let P be an extended logic program and $I = I_T \cup not\ I_F$ be an interpretation where I_T is a set of suffixed literals and I_F a set of objective literals. Let T and F be a set of suffixed literals and a set of objectve literals, respectively. The mappings Φ_I^s and Ψ_I^s are defined as follows:

$$
\begin{aligned}
\Phi^s_I(T) = \{ & A^{\Sigma} \mid \text{there are } k \text{ rules } A \leftarrow B_{l1}, \ldots, B_{lm}, not\, C_{l1}, \ldots, not\, C_{ln} \\
& \text{in } P \text{ with } 1 \leq l \leq k \text{ such that } \forall B_{li}\ (1 \leq i \leq m) B_{li}^{\Sigma_{li}} \in I_T \cup T, \\
& \forall\ C_{lj} (1 \leq j \leq n)\ C_{li} \in I_F \text{ and where } \Sigma \text{ is} \\
& \bigcup_l \{ \{B_{l1}, \ldots, B_{lm}, not\, C_{l1}, \ldots, not\, C_{ln}\} \cup \sigma_{l1} \cup \ldots \cup \sigma_{lm} \mid \\
& \quad \sigma_{li} \in \Sigma_{li} \} \quad \} \\
\Psi^s_I(F) = \{ & A \mid \text{for every rule } A \leftarrow B_1, \ldots, B_m, not\, C_1, \ldots, not\, C_n \\
& \text{in } P, \exists B_i \in I_F \cup F\ (1 \leq i \leq m) \text{ or } \exists C_j\ (1 \leq j \leq n) \\
& \text{such that } C_j^{\Sigma_j} \in I_T \quad \}
\end{aligned}
$$

Operator Φ^s_I is used to determine the set of objective true literals given the current interpretation I, and keeps track of all literals involved in the proof of such objective literals. In fact, it is an enhanced version of the T_P operator. On the other hand, Ψ^s_I is used to determine the set of default falsities given I. The complete definition of the suspicious well-founded model M^s_P can be found in [78] and is similar to Przymuskinski's constructive definition of *WFS* in terms of the Θ operator [69]. A suffixed literal L^{Σ} in M^s_P is suspicious iff every set in Σ has a literal L_1 such that $L_1^{\Sigma_1}$ and $\neg L_1^{\Sigma_2}$ both belong to M^s_P. It can be clearly seen from the definition that the treatment of objective and default literals is different. We will see shortly what are the consequences of such a commitment.

Example 27 [78]. Let P be the program:

$$
\begin{aligned}
& innocent \leftarrow \neg guilty. \\
& \neg guilty \leftarrow charged \wedge not\, guilty. \\
& charged.
\end{aligned}
$$

The suspicious well-founded model is

$$
\begin{gathered}
\left\{ charged^{\{\{\}\}}, \neg guilty^{\{\{charged, not\, guilty\}\}} \right\} \cup \\
\left\{ innocent^{\{\{\neg guilty, charged, not\, guilty\}\}} \right\} \cup \\
not\, \{\neg charged, guilty, \neg innocent, man, \neg man\}
\end{gathered}
$$

None of the literals is contradictory or suspicious.

On the other hand, if $\neg charged$ and man are added instead to P, the truth value of *charged* becomes contradictory, *man* is true (safe) and the truth-values of the other literals remain the same as before. The new suspicious well-founded model is:

$$
\begin{gathered}
\left\{ charged^{\{\{\}\}}, \neg charged^{\{\{\}\}}, man^{\{\{\}\}}, \neg guilty^{\{\{charged, not\, guilty\}\}} \right\} \\
\cup \\
\left\{ innocent^{\{\{\neg guilty, charged, not\, guilty\}\}} \right\} \cup not\, \{guilty, \neg innocent, \neg man\}
\end{gathered}
$$

The truth of both *innocent* and $\neg guilty$ is now less credible since they are derived from the inconsistent literal *charged*, as can be checked from their suffixes. Thus, *innocent* is true suspiciously, and *guilty* false suspiciously.

The author argues that the extra information of support on contradiction is only required for objective literals: the truth of a default literal can never depend on the success of a pair of contradictory literals since all "proofs" for that literal fail. This is not very convincing since the failure may be due to a pair of contradictory literals too, as shown in the next example.

Example 28. Consider the following extended logic program:

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

The suspicious well-founded model of the above program entails a. 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 is incapable of detecting this situation, since the suffixes are not propagated over default negated literals.

Thus, according to our Definition 42, the suspicious well-founded model lacks some power for detecting support on contradiction over default negated literals. In fact, one can remove all occurences of default negated literals in the suffixes of literals, since they are not used at all, for the purpose of checking dependency on contradiction and simplifying the definition of Φ^s_I. The definition of suspicious well-founded model is overly complicated, since there is no need to keep all the dependencies. Furthermore, the size of the suffixes can be exponential in the size of the program which is a disadvantage for a practical implementation.

However we have the following partial result:

Theorem 44. *Let P be a definite extended logic program with suspicious well-founded model M^s_P. A literal L depends on contradiction iff L is suspiciously true in M^s_P.*

6.2 Sakama and Inoue's suspicious p-stable semantics

The work of Sakama on a suspicious well-founded model has been continued in [80] under an extension of stable model semantics. The semantics is defined for disjunctive extended logic programs with integrity constraints. As usual, and for the sake of comparisons, we restrict the analysis and definitions to the extended logic programming case.

The idea is to introduce adorned literals of the form L^s, where L is an objective literal. The new literals of the form L^s denote suspicious literals. Interpretations I^s are subsets of

$$\{A, \neg A, A^s, \neg A^s \mid \text{where } A \text{ is an atom in the language of a program } \}$$

For simplicity of presentation we assume that if L belongs to some interpretation then its adorned counterpart L^s also does. We shall say that a literal is (safely) true when $L \in I^s$. Literal L is suspiciously true iff $L^s \in I^s$ and $L \notin I^s$. The reason for this will be apparent below. A new definition of the immediate consequence operator is given in order to cope with these extended interpretations. As usual, the operator is defined only for definite extended logic programs.

Definition 45. Let P be a definite extended logic program and an interpretation I^s. Then

$$\begin{aligned} T_P^s(I^s) = \{\, L \mid & \text{ exists } L \leftarrow L_1, \ldots, L_n \text{ in } P \text{ such that} \\ & \forall_{1 \leq i \leq n}\ L_i \in I^s \text{ and } \neg L_i \notin I^s \text{ and } \neg L_i^s \notin I^s\} \\ \cup \{\, L^s \mid & \text{exists } L \leftarrow L_1, \ldots, L_n \text{ in } P \text{ such that} \\ & \forall_{1 \leq i \leq n}\ L_i \in I^s \text{ or } L_i^s \in I\} \end{aligned}$$

The intuition is straightforward. A literal is safely entailed iff there is at least one rule with the body safely entailed. In order to check that the body does not depend on contradiction we have to test that the explicit complements of the body literals and their corresponding adorned literals are not entailed. A literal is suspiciously entailed whenever there is a rule for it with the body at least suspiciously entailed. The above definition guarantees that whenever $L \in T_P^s(I^s)$ then it is also the case that $L^s \in T_P^s(I^s)$. The model of program P is obtained by finding the least fixpoint of T_P^s, i.e. $\mathcal{M}_P^s = T_P^s \uparrow^\omega$.

Example 29. Consider again the program of Example 26

$$\mathit{cross} \leftarrow \neg \mathit{train}. \qquad \mathit{train}. \qquad \neg \mathit{train}.$$

Then

$$\begin{aligned} & T_P^s \uparrow^0 = \{\} \\ & T_P^s \uparrow^1 = T_P^s(T_P^s \uparrow^0) = \{\mathit{train}, \mathit{train}^s, \neg \mathit{train}, \neg \mathit{train}^s\} \\ & T_P^s \uparrow^2 = T_P^s(T_P^s \uparrow^1) = \{\mathit{cross}^s, \mathit{train}, \mathit{train}^s, \neg \mathit{train}, \neg \mathit{train}^s\} \\ & T_P^s \uparrow^3 = T_P^s(T_P^s \uparrow^1) = \mathcal{M}_P^s \end{aligned}$$

Thus, *train* and $\neg \mathit{train}$ are contradictory and *cross* is suspiciously true.

The extension of the above semantics to arbitrary extended logic programs is immediate and is along the lines of the definition of stable models:

Definition 46. Let P be an extended logic program and I^s an interpretation. As in Definition 23, the reduct of P by I^s is the program $\frac{P}{I^s}$ obtained from P by removing all rules containing a default literal $\mathit{not}\, A$ such that $A^s \in I^s$, and then by removing all the remaining default literals from P.

The suspicious p-stable models of P are the interpretations I^s such that

$$\mathcal{M}^s_{\frac{P}{I^s}} = I^s$$

Example 30. Consider the program:

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

Its single suspicious p-stable model I^s is

$$\begin{aligned} \{ & \mathit{man}, \mathit{man}^s, \mathit{charged}, \mathit{charged}^s, \neg \mathit{charged}, \\ & \neg \mathit{charged}^s, \neg \mathit{guilty}^s, \mathit{innocent}^s\} \end{aligned}$$

This can be easily checked by noting that $\frac{P}{I^s}$ is

$$innocent \leftarrow \neg guilty.\ charged.man.$$
$$\neg guilty \leftarrow charged.\ \ \neg charged.$$

with the model identical to I^s. One can conclude that *man* is true, *charged* and $\neg charged$ are contradictory, and that $\neg guilty$ and *innocent* are suspiciously true. These are the expected results.

Compare this logical approach of suspicious stable models with the previous semantics based on well-founded semantics. It can be easily shown that for positive definite extended well-founded programs both semantics coincide. But, the definition of the semantics in this section is much more declarative and intuitive. A reformulation of the suspicious well-founded semantics in terms of a nested application of $\mathcal{M}^s_{\frac{P}{I^s}}$ is immediate. We leave the details to the interested reader.

As for suspicious well-founded semantics, the propagation of support on contradiction through default negated literals is not taken care. Example 28 also applies to the semantics presented above. Therefore, a partial completeness result along the lines of Theorem 44 holds for suspicious p-stable models, with the obvious changes.

The authors define the logic **VI** to provide a model theory for their semantics. Besides the usual four values $\bot$, **f**, **t** and $\top$, there are two additional logical values, **sf** and **st** which denote suspiciously false and suspiciously true, respectively. The logic is not a bilattice (see Figure 8). However we were able to define truth-tables for all connectives which respect the caveats of the authors. These can be found in Tables 10, 11 and 12.

Table 10. Truth table for negations in Logic **VI**

A	$\bot$	**sf**	**st**	**f**	**t**	$\top$
$\neg A$	$\bot$	**st**	**sf**	**t**	**f**	$\top$
$not\ A$	$\top$	**st**	**sf**	**t**	**f**	$\bot$

The consequence relation $\models_6$ is defined by means of the set of designated truth-values $\{\mathbf{st}, \mathbf{t}, \top\}$.

The implication table is defined as usual in order to satisfy

$$I \models_6 A \leftarrow B \text{ iff } I \models_6 A \text{ or } I \not\models_6 B$$

Under the equivalence relation $\models\models_6$ a lot of properties of classical logic are obeyed. First, default negation is classical negation. This is expected since logic **VI** is a refinement of logic **IV** of the same authors, discussed in Section 5.1. As usual, explicit negation fails to obey the excluded middle, the contradiction law,

[ht]

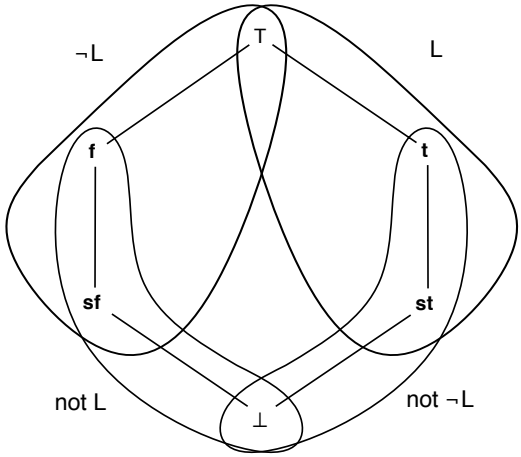

Fig. 8. Sakama and Inoue's Logic **VI**

Table 11. Truth table for conjunction in Logic **VI**

$\wedge$	$\bot$	**sf**	**st**	**f**	**t**	$\top$
$\bot$	$\bot$	**sf**	$\bot$	**f**	$\bot$	**f**
sf	**sf**	**sf**	**sf**	**f**	**sf**	**sf**
st	$\bot$	**sf**	**st**	**f**	**st**	$\top$
f	**f**	**f**	**f**	**f**	**f**	**f**
t	$\bot$	**sf**	**st**	**f**	**t**	$\top$
$\top$	**f**	**sf**	$\top$	**f**	$\top$	$\top$

Table 12. Truth table for disjunction in Logic **VI**

$\vee$	$\bot$	**sf**	**st**	**f**	**t**	$\top$
$\bot$	$\bot$	$\bot$	**st**	$\bot$	**t**	**t**
sf	$\bot$	**sf**	**st**	**sf**	**t**	$\top$
st	**st**	**st**	**st**	**st**	**t**	**st**
f	$\bot$	**sf**	**st**	**f**	**t**	$\top$
t	**t**	**t**	**t**	**t**	**t**	**t**
$\top$	**t**	$\top$	**st**	$\top$	**t**	$\top$

contraposition, coherence, Modus Tollens, and Disjunctive Syllogism. However the stronger equivalence $\equiv_6$ is much worse behaved. For instance, the associativity and distributivity laws are not satisfied.

Finally, we present the mapping between suspicious p-stable models and six-valued interpretations.

Theorem 47. [78] *Let P be an extended logic program and I^s a suspicious p-stable model. Then, construct the* **VI** *interpretation as follows:*

- *If A and $\neg A$ belong to I^s then $I(A) = \top$;*
- *else, if $A \in I^s$ (resp. $\neg A \in I^s$) then $I(A) = \mathbf{t}$ (resp. $I(A) = \mathbf{f}$);*
- *else, if $A^s \in I^s$ (resp. $\neg A^s \in I^s$) then $I(A) = \mathbf{st}$ (resp. $I(A) = \mathbf{sf}$);*
- *otherwise, $I(A) = \bot$.*

The interpretation I is a **VI** *model of program P.*

6.3 Suspicious $WFSX_{\mathrm{p}}$

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. We start by stating a main result regarding $WFSX_{\mathrm{p}}$:

Theorem 48. *Let P be an extended logic program with paraconsistent well-founded model M such that for every objective literal L we have $L \in M$ or not $L \in M$ or both. Some $L \in M$ depends on contradiction iff both L and not L are in M.*

Thus $WFSX_{\mathrm{p}}$ detects support on contradiction in total paraconsistent models simply by testing for the presence of both L and *not* L in the model. Dependence is also an operational concept, related to the underlying proof procedures for $WFSX_{\mathrm{p}}$ described in [3, 19].

Theorem 48 can be made more general: 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.

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

Example 31. Consider again the situation of Example 26. The $WFSX_{\mathrm{p}}$ semantics states that the driver can cross the railroad, but to be careful since this conclusion depends on contradictory information. This is checked by noticing that both *cross* and *not cross* simultaneously hold in the program's model. This is achieved as a result of the coherence principle: *train* holds therefore *not* $\neg$*train* also holds; since the body of the rule for *cross* has the body falsified by *not* $\neg$*train* we conclude *not cross*.

Example 32. The program of Example 30 has the following paraconsistent well-founded model:

$$\{charged, \neg charged, \neg guilty, innocent, man\} \cup$$
$$not\, \{charged, \neg charged, guilty, \neg guilty, innocent, \neg innocent, \neg man\}$$

For this program, objective literal conclusions in our semantics coincide with the usual ones obtained by suspicious well-founded and p-stable models. 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.

As we have seen, the previous semantics in Sections 6.1 and 6.2 are not able to propagate the knowledge about the support on contradiction through negation by default. Let us compare with the results obtained by $WFSX_{\mathrm{p}}$:

Example 33. Returning to the program of Example 28 we obtain as paraconsistent well-founded model the set of literals:

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

This means that c and $\neg c$ are contradictory and that a and b depend on contradictory information, which are the intended results.

Note that we can apply $WFSX_{\mathrm{p}}$ to detect support on contradiction on definite extended logic programs instead of using the suspicious semantics of the previous two sections. For objective literals the results of the three semantics coincide.

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 48. 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 via Theorem 48. This can occur only if there are undefined literals in the model. Theorem 51 below corrects this situation, so that $WFSX_{\mathrm{p}}$ does cover all the cases. Let us see what the problem is.

Example 34. 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 suspicious well-founded semantics handles this program properly.

In [21] we equipped $WFSX_{\mathrm{p}}$ semantics with additional mechanisms to handle the non-total cases. Failure to detect support on contradiction by Theorem 48 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 49. 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 35. Consider the program of Example 34. 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 that of the original program.

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

The idea is to apply the Δ operator iteratively till the least fixpoint is reached. At each step the undefined rules which can prevent the propagation of contradiction supportedness are removed. In the end one obtains the desired property of contradiction support detection, since all rules which could prevent the application of Theorem 48 have been removed.

Definition 50. 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 the suspicious paraconsistent well-founded model $WFM_p^s(P) = WFM_p(P_\lambda)$, and the main result is readily obtained:

Theorem 51. *Let P be an extended logic program and let $M = WFM_p^s(P)$. Both literal L and its default complement $not\, L$ are in M iff $L \in M$ and L depends on contradiction.*

Example 36. The sequence originated by program P of Example 34 is:

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

An important remark is that the construction of Definition 50 is not continuous. As a result, more than ω steps may be necessary to obtain the least fixpoint. This behaviour is illustrated next.

Example 37. Let P be the following extended logic program, whose variables range over the natural numbers:

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

The transfinite sequence of programs and associated models $M_i = WFM_p(P_i)$ is, where "_" is an anonymous variable:

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

The $WFM^s_p(P)$ 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 default false. According to Theorem 51 this means that all literals depend on some contradictory literal. Since c is the single contradictory literal, all other literals depend on c.

Finally, the $\mathcal{NINE}$ logic provides the appropriate model theory for the suspicious version of the $WFSX_{\mathrm{p}}$ semantics (see Section 4.2).

7 Blocking Contradiction Propagation

The semantics we have discussed allow one to conclude for the truth of literals even if on the basis of contradictory information. 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, when faced with the

contradiction $\neg surgery\_indication$ and $surgery\_indication$ a physician opts for the latter. Also, consider the situation of the bus driver who wants to cross a railroad and has contradictory information about a coming train. The best strategy is to not cross the railroad. However, a criminal trying to escape the police might risk it and cross the railroad. Thus the appropriate type of reasoning is situation dependent. In this section we discuss several approaches to implicitly or explicitly block the propagation of contradiction.

We start by presenting Wagner's conservative, credulous, and skeptical forms of reasoning [90]. We then overview the introspective framework presented in [93], where the language of extended logic programs is augmented with two new operators. These modalities provide the means of implementing reasoning forms of the kind "Does L depend on some contradiction ?" and "Is L contradictory ?". We finally discuss the extension of $WFSX_{\mathrm{p}}$ incorporating these two operators which appeared in [21].

Wagner's entailment relations and the introspective framework of [93] adopt the potential-contradictions view, while we stick to the actual contradictions view but providing new mechanisms for explicitly blocking contradiction propagation. In some sense these two semantics open new horizons for extended logic programming paraconsistent semantics. However, as we shall see, both suffer from problems. The reader is warned that this section presents work that is still exploratory and no definite solutions have been reached.

7.1 Wagner's credulous, conservative and skeptical reasoning

To handle the more careful reasoning discussed in this section's introduction, Wagner defines new reasoning forms based on defeasible inheritance systems [90]. The idea is that contradictory pieces of information neutralize each other. Essentially, a conclusion is accepted only if it is supported and not doubted. By selecting different notions of supportedness and doubt different forms of reasoning are obtained, i.e. what counts in favour of some potential conclusion and what counts as a neutralizing counterargument[9]. He proposes credulous, conservative and skeptical reasoning, besides the liberal kind. The blocking of contradiction propagation (neutralization) is inbuilt into the several entailment relations. There is no explicit control by the user of which form of blocking to use: if one wants a different behaviour one should select another semantics.

Credulous ($\vdash_{cr}$), conservative ($\vdash_{c}$) and skeptical ($\vdash_{s}$) reasoning are defined proof theoretically via entailment relations containing the basic rules (1), ($--$), ($\wedge$), and ($-\wedge$) (see Section 4.3 for the definitions and language used), plus the following ones for derivation of literals and their weak negation complements:

[9] For definition of paraconsistent logics based on argumentation concepts see Anthony Hunter's chapter in this volume.

$$
\begin{array}{lll}
(l)\ \ X \vdash_{cr} l & \text{iff} & X \vdash_{l} l, \text{ and } \forall(\tilde{l} \leftarrow E) \in X : X \vdash_{cr} -E \\
(-l)\ X \vdash_{cr} -l & \text{iff} & X \vdash_{l} -l, \text{ or } \exists(\tilde{l} \leftarrow E) \in X : X \vdash_{cr} E
\end{array}
$$

$$
\begin{array}{lll}
(l)\ \ X \vdash_{c} l & \text{iff} & \exists(l \leftarrow E) \in X : X \vdash_{c} E, \\
 & \text{iff} & \qquad \text{and } \forall(\tilde{l} \leftarrow F) \in X : X \vdash_{c} -F \\
(-l)\ X \vdash_{c} -l & \text{iff} & \forall(l \leftarrow E) \in X : X \vdash_{c} -E, \\
 & \text{iff} & \qquad \text{or } \exists(\tilde{l} \leftarrow F) \in X : X \vdash_{c} F
\end{array}
$$

$$
\begin{array}{lll}
(l)\ \ X \vdash_{s} l & \text{iff} & \exists(l \leftarrow E) \in X : X \vdash_{s} E, \text{ and } X \vdash_{l} -\tilde{l} \\
(-l)\ X \vdash_{s} -l & \text{iff} & \forall(l \leftarrow E) \in X : X \vdash_{s} -E, \text{ or } X \vdash_{l} \tilde{l}
\end{array}
$$

In credulous reasoning a conclusion l is accepted if it is liberally inferred but not credulously doubted, i.e. when $\tilde{l}$ is not (potentially) credulously entailed, where $\tilde{l}$ is the strong negation complement of l. In conservative reasoning, support and doubt have the same weight: a conclusion holds if it is conservatively supported but not conservatively doubted. Finally, Wagner proposes a skeptical inference relation. The intuition is that liberally entailed literals cast doubt on their strong negation complements, i.e. there must be no doubt at all in order to establish a conclusion. Both conservative and skeptical reasoning are inherently consistent.

Example 38. [90] Let X_1 be the following extended deductive database:

$$p \leftarrow 1. \quad \sim p \leftarrow 1. \quad \sim q \leftarrow 1. \quad q \leftarrow p. \quad r \leftarrow p. \quad \sim r \leftarrow q.$$

Literals p, $\sim p$, q, $\sim q$, r and $\sim r$ are liberally entailed by X_1. But only $\sim q$ and $\sim r$ are credulously entailed. It is obvious that p and $\sim p$ are not credulously entailed because, although they are liberally entailed, there are rules of the form $\sim p \leftarrow 1$ and $p \leftarrow 1$, respectively, which do not have their bodies "falsified". The same applies to q. Now, $\sim q$ is credulously entailed since $\sim q$ is liberally entailed but $-p$ is credulously entailed. The same sort of reasoning applies to r and $\sim r$, and therefore both are entailed. Conservatively we can only conclude $\sim q$. Nothing is skeptically derivable.

It is important to notice that conclusion r, under credulous reasoning, is in some sense not supported because the body of its single rule is not entailed. Furthermore, the example shows that the set of formulae inferred under credulous reasoning is not a model of the extended deductive base, according to Wagner's notion of liberal model, because we have a true body, 1, but a false head (for instance p). The same problem can occur with conservative and skeptical reasoning.

In the above systems of reasoning, the case when there is a dependency of reasoning of l on $\tilde{l}$, or vice versa, is not allowed: only the more restricted class of strongly well-founded *XDBs* is considered by Wagner, where no positive or negative loops (through either explicit or default negation) are permitted.

We can reduce entailment in strongly well-founded *XDBs*, under the above three reasoning forms, to entailment in normal logic programs under *WFS*. Each form of reasoning requires a different program transformation, to be found in the following definitions, where X is a *XDB*.

Definition 52. Program P^{cr} is constructed as follows. First, program P^l (see Definition 31) is included in P^{cr}. For each atom in the language of X the following two rules

$$a^+ \leftarrow a^p, not\ doubt\_a^+ \text{ and } a^- \leftarrow a^n, not\ doubt\_a^-$$

are added to P^{cr}, where a^+, $doubt\_a^+$, a^-, and $doubt\_a^-$ are new predicate symbols. Finally, for each rule of the form (where a_i, b_j, c_k, d_l are atoms)

$$l \leftarrow a_1, \ldots, a_m, \sim b_1, \ldots, \sim b_n, -c_1, \ldots, -c_o, - \sim d_1, \ldots, - \sim d_p$$

add to P^{cr} the rule with head $doubt\_a^-$ if $l = a$ (or head $doubt\_a^+$ if $l = \sim a$) and body

$$a_1^+, \ldots, a_m^+, b_1^-, \ldots, b_n^-, not\ c_1^+, \ldots, not\ c_o^+, not\ d_1^-, \ldots, not\ d_p^-$$

Definition 53. [85] Program P^{co} is constructed as follows. First, for each atom in the language of X the two rules $a^+ \leftarrow a^p, not\ a^n$ and $a^- \leftarrow a^n, not\ a^p$ are added to P^{co}, where a^+, a^-, a^p, and a^n are new predicate symbols. For each rule of the form (where a_i, b_j, c_k, d_l are atoms)

$$l \leftarrow a_1, \ldots, a_m, \sim b_1, \ldots, \sim b_n, -c_1, \ldots, -c_o, - \sim d_1, \ldots, - \sim d_p$$

add to P^{co} the rule with head a^p if $l = a$ (or head a^n if $l = \sim a$) and body

$$a_1^+, \ldots, a_m^+, b_1^-, \ldots, b_n^-, not\ c_1^+, \ldots, not\ c_o^+, not\ d_1^-, \ldots, not\ d_p^-$$

Definition 54. Program P^s is constructed as follows. First, program P^l is included in P^s. For each rule of the form (where a, a_i, b_j, c_k, d_l are atoms)

$$a \leftarrow a_1, \ldots, a_m, \sim b_1, \ldots, \sim b_n, -c_1, \ldots, -c_o, - \sim d_1, \ldots, - \sim d_p$$

add to P^s the rule

$$a^+ \leftarrow a_1^+, \ldots, a_m^+, b_1^-, \ldots, b_n^-, \\ not\ c_1^+, \ldots, not\ c_o^+, not\ d_1^-, \ldots, not\ d_p^-, not\ a^n$$

For each rule of the form (where a, a_i, b_j, c_k, d_l are atoms)

$$\sim a \leftarrow a_1, \ldots, a_m, \sim b_1, \ldots, \sim b_n, -c_1, \ldots, -c_o, - \sim d_1, \ldots, - \sim d_p$$

add to P^s the rule

$$a^- \leftarrow a_1^+, \ldots, a_m^+, b_1^-, \ldots, b_n^-, \\ not\ c_1^+, \ldots, not\ c_o^+, not\ d_1^-, \ldots, not\ d_p^-, not\ a^p$$

The rationale behind the first transformation is that inference rule (l) is captured by the logic program rules introduced in the second step of the transformation. We have a conjunction of two literals: one checks whether the literal is liberally entailed, and the other tests whether it is not doubted. The rules introduced in the last step test if a literal is doubted, i.e. implement the test $\exists(\tilde{l} \leftarrow E) \in X : X \vdash_{cr} E$. Weak negation derivability inference rules are captured by negation as failure. For conservative reasoning the transformation described is due to Teusink [85], which he uses to compute approximations to the answer-set semantics. The idea is again to code rule (l) with the first type of rules. Rule with head a^p (a^n) test the condition $\exists(a \leftarrow E) \in X : X \vdash_c E$ ($\exists(\sim a \leftarrow E) \in X : X \vdash_c E$). Now the translation is obvious, since we can again equate negation by default with weak negation derivability. For skeptical reasoning the intuition is quite simple. The extra literal added at the end of the rule performs the test whether the strong complement of the literal in the head is liberally entailed. Now the equivalence theorem is immediate:

Theorem 55. *Let X be a strongly well-founded* XDB*. Then the following equivalences hold, where a is an atom and $*$ is one of cr, co or s:*

$$\begin{array}{ll} X \vdash_* a & \textit{iff } a^+ \in WFM(P^*) \\ X \vdash_* -a & \textit{iff not } a^+ \in WFM(P^*) \\ X \vdash_* \sim a & \textit{iff } a^- \in WFM(P^*) \\ X \vdash_* -\sim a & \textit{iff not } a^- \in WFM(P^*) \end{array}$$

Example 39. Consider the *XDB* X_1 of Example 38. Its corresponding program P^{cr} is

$$\begin{array}{lll} p^p. & p^+ \leftarrow p^p, not\, doubt\_p^+. & doubt\_p^-. \\ p^n. & p^- \leftarrow p^n, not\, doubt\_p^-. & doubt\_p^+. \\ q^n. & q^+ \leftarrow q^p, not\, doubt\_q^+. & doubt\_q^+. \\ q^p \leftarrow p^p. & q^- \leftarrow q^n, not\, doubt\_q^-. & doubt\_q^- \leftarrow p^+. \\ r^p \leftarrow p^p. & r^+ \leftarrow r^p, not\, doubt\_r^+. & doubt\_r^- \leftarrow p^+. \\ r^n \leftarrow q^p. & r^- \leftarrow r^n, not\, doubt\_r^-. & doubt\_r^+ \leftarrow q^+. \end{array}$$

The well-founded model of P^{cr} consists of the following true facts[10]:

$$\{p^p, p^n, q^p, q^n, r^p, r^n, doubt\_p^+, doubt\_p^-, doubt\_q^+, q^-, r^+, r^-\}$$

which is in accordance with the credulous model of X_1.

Example 40. Consider the *XDB* X_1 of Example 38. Program P^{co} is

$$\begin{array}{ll} p^+ \leftarrow p^p, not\, p^n. & p^p. \\ p^- \leftarrow p^n, not\, p^p. & p^n. \\ q^+ \leftarrow q^p, not\, q^n. & q^p \leftarrow p^+. \\ q^- \leftarrow q^n, not\, q^p. & q^n. \\ r^+ \leftarrow r^p, not\, r^n. & r^p \leftarrow p^+. \\ r^- \leftarrow r^n, not\, r^p. & r^n \leftarrow q^+. \end{array}$$

[10] All other literals are false since the model is two-valued.

The well-founded model of P^{co} consists of the following true facts:

$$\{p^p, p^n, q^n, q^-\}$$

Therefore, $\sim q$ is entailed and all other literals are weakly false in X_1 under conservative reasoning.

Example 41. Consider the *XDB* X_1 of Example 38. Program P^s is

$$\begin{array}{ll} p^+ \leftarrow not\, p^n. & p^p. \\ p^- \leftarrow not\, p^p. & p^n. \\ q^+ \leftarrow p^+, not\, q^n. & q^p \leftarrow p^p. \\ q^- \leftarrow not\, q^p. & q^n. \\ r^+ \leftarrow p^+, not\, r^n. & r^p \leftarrow p^p. \\ r^- \leftarrow q^+, not\, r^p. & r^n \leftarrow q^p. \end{array}$$

The well-founded model of P^s consists of the following true facts:

$$\{p^p, p^n, q^p, q^n, r^p, r^n\}$$

and thus all literals are weakly false in X_1 under skeptical reasoning.

One advantage of using a normal logic program under *WFS* semantics is that now one can assign semantics to non-strongly well-founded *XDBs*. Moreover, one can combine the four different entailment relations in a single program. Furthermore we can detect the mutual recursive problematic cases via literals which are undefined in the well-founded model. The other non-undefined literals still derive sound conclusions according to Wagner's inference systems. Of course, the proof-theoretical inference system should be extended in order to cope with this larger class of *XDBs* as Wagner does for liberal reasoning with system $\vdash_l^{loop}$.

We should notice that Wagner named his systems skeptical, conservative, credulous, and liberal, on the motivation that for weak negation-free *XDBs* the following relation holds, where l is an atom or the strong negation of an atom:

$$X \vdash_s l \Rightarrow X \vdash_c l \Rightarrow X \vdash_{cr} l \Rightarrow X \vdash_l l$$

The introduction of weak negation destroys some of these inclusions. One important example is Teusink's *XDB* $\{\sim p \leftarrow -p; p \leftarrow 1\}$, where p is skeptically but neither conservatively nor credulously derivable. Finally, by replacing in an *XDB* every strongly negated literal $\sim a$ by a new atom, say a^n, then liberal reasoning can be indifferently "implemented" with resort to credulous, conservative, or skeptical reasoning, hence concluding that these four inference relations have the same expressive power.

7.2 Paraconsistent Introspective Programs

A three-valued framework for paraconsistent logic programs has been proposed in [93], including a form of 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 will start by sketching the language and semantics as given in [93] We will show that the semantics is ill-defined for the class of positive introspective logic programs, i.e. with the modal operators and without occurrences of default negation. We identify the problem and then proceed by discussing the semantics with default negation. We restrict the discussion to disjunction free programs.

Definition 56. [93] Introspective 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 not permitted.

Example 42. [93] 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:

$$penalty \leftarrow watering. \quad watering.$$
$$penalty \leftarrow \neg watering. \; \neg watering.$$

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. Note that neither *watering* nor $\neg watering$ nor *penalty* are entailed.

It is clear from the example that 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 makes the reading of logic programs difficult (one cannot immediately see if some literal is entailed or not). Furthermore, the authors discard the coherence principle.

The semantics for positive introspective programs is given by structures, as defined below.

Definition 57. An interpretation is a set of objective literals. A structure is a tuple $\langle I, \{I_1, \ldots, I_k\}\rangle$ where I is an interpretation and each I_j, $1 \leq j \leq k$, is a consistent subset of I.

According to the authors, I is the current world, possibly inconsistent, and the I_js are possible consistent subworlds of I. The structures induce an entailment relation described in Definition 58.

Definition 58. Let $S = \langle I, \{I_1, \ldots, I_k\}\rangle$ be a structure. The logical entailment relation between structures and formulas is defined as follows, where L is an objective literal and F and G are arbitrary formulas:

- $S \models_{para} L$ iff $L \in I_i$ for every $1 \leq i \leq k$;
- $S \models_{para} F \wedge G$ iff $S \models_{para} F$ and $S \models_{para} G$;
- $S \models_{para} not\, L$ iff $S \not\models_{para} L$;
- $S \models_{para} \mathbf{C}L$ iff $L \in I$ and $\neg L \in I$;
- $S \models_{para} \mathbf{C_d}L$ iff $L \in I$ and $S \models_{para} not\, L$;

The intuition should be obvious. An objective literal is entailed iff it is true in all consistent subworlds, i.e it must be safely inferred. The satisfaction of a conjunction and of the default negation operator are obvious. If a literal and its explicit complement both belong to the current world then it is contradictory. A literal depends on contradiction iff it belongs to the current world but cannot be safely entailed from the consistent worlds. Notice the similarities with Definition 42.

The candidate models for positive introspective programs are obtained from structures with additional properties:

Definition 59. Let P be a positive introspective program. A structure $S = \langle I, M\rangle$ is said to be a pre-model of P iff:

1. For each rule $L \leftarrow B_1, \ldots, B_m$ in P, if for all B_j $(1 \leq j \leq m)$ either $B_j \in I$ or $S \models_{para} B_j$, then $L \in I$;
2. Each I_i is S-supported, i.e. for each literal $L \in I_i$ there is a rule $L \leftarrow B_1, \ldots, B_m$ in P such that $S \models_{para} B_1 \wedge \ldots \wedge B_m$;
3. Each I_i is maximal in the sense that there is no consistent, S-supported I' with $I_i \subset I' \subseteq I$.

The authors then define two ordering relations among pre-models to choose the ones with the intended properties, the minimal para-models of P. For the sake of simplicity we do not present such relations here. It is said that every positive paraconsistent programs has a single minimal para-model.

Example 43. Consider the program of Example 42 with the rule for *appeal* added. Its unique minimal para-model is

$$\left\langle \begin{array}{c} \{appeal, penalty, watering, \neg watering\}, \\ \{\{appeal, watering, \}, \{appeal, \neg watering\}\} \end{array} \right\rangle$$

This minimal para-model implies *appeal*, $\mathbf{C}watering$, $\mathbf{C_d}penalty$ and *not penalty*.

Unfortunately, in [93] it is not realized that there are positive introspective programs without any pre-models, as the next example shows.

Example 44. Consider the following program:

$$a \leftarrow b. \qquad a \leftarrow \mathbf{C_d}a. \qquad b. \qquad \neg b.$$

By condition (1) of Definition 59 I should include a, b and $\neg b$. The candidate structures are:

$$S_1 = \langle \{a, b, \neg b\}, \{\{\}\} \rangle$$
$$S_2 = \langle \{a, b, \neg b\}, \{\{b\}, \{\neg b\}\} \rangle$$
$$S_3 = \langle \{a, b, \neg b\}, \{\{a, b\}, \{\neg b\}\} \rangle$$
$$S_4 = \langle \{a, b, \neg b\}, \{\{b\}, \{a, \neg b\}\} \rangle$$
$$S_5 = \langle \{a, b, \neg b\}, \{\{a, b\}, \{a, \neg b\}\} \rangle$$

However, none obeys the three conditions of being a pre-model. S_1 S_2, S_3 and S_4 do not satisfy the maximality condition (3). Structure S_5 is not S_5-supported since $\mathbf{C_d}a$ is not entailed in S_5.

The problem should be obvious: if $\mathbf{C_d}a$ is false then a depends on contradiction and therefore $\mathbf{C_d}a$ should hold. If $\mathbf{C_d}a$ is true then there is a proof for a which does not depend on contradiction and therefore $\mathbf{C_d}a$ should be false. This behaviour is very similar to the inexistence of stable models in programs with odd loops through negation by default. We conjecture that the problems may be addressed by introducing a notion of stratification with respect to the occurrences of the $\mathbf{C_d}$ operator. We resume the discussion of computing the minimal para-model in the next section.

The extension of the semantics for handling default negation is straightforward. First, a notion of reduct $\frac{P}{S}$ of a program P by a structure S is provided and is obtained by removing all rules which have a default literal $not\, L$ such that $S \models_{para} L$, and then deleting all default negated literals occuring in the remaining rules. This allows the definition of several semantics in terms of fixpoint equalities, along the lines of the semantics for normal logic programs.

Definition 60. Let P be an introspective program and $\mathcal{F}_P(S)$ be the operator returning the minimal para-model of $\frac{P}{S}$. Define the relation $\preceq$ over structures by letting $S_1 \preceq S_2$ iff $I_1 \subset I_2$ or ($I_1 = I_2$ and $M_1 \subseteq M_2$). Let S be a structure, then

- S is a stable structure iff S is a fixpoint of $\mathcal{F}_P$, i.e. $\mathcal{F}_P(S) = S$;
- S is an alternating fixpoint of $\mathcal{F}_P$ iff $\mathcal{F}_P(\mathcal{F}_P(S)) = S$. It is called normal if $S \preceq \mathcal{F}_P(S)$;
- S is the well-founded structure iff S is the least normal alternating fixpoint of $\mathcal{F}_P$;

Example 45. [93] Consider the following program:

$$bird \leftarrow tweety. \quad fly \leftarrow bird, notab. \quad ab \leftarrow Cfly.$$
$$tweety. \qquad\qquad \neg fly \leftarrow tweety.$$

The well-founded structure of this program is

$$S = \langle \{tweety, bird, \neg fly\}, \{\{tweety, bird, \neg fly\}\} \rangle$$

Therefore, it entails *tweety*, *bird* and fly, illustrating the blocking of the default assumption $not\, ab$ in the presence of a contradiction over fly. Note that default negated literals are evaluated in $S' = \mathcal{F}_P(P^S)$, i.e.

$$S' = \left\langle \begin{array}{l} \{tweety, bird, fly, \neg fly, ab\}, \\ \{\{tweety, bird, fly, ab\}, \{tweety, bird, \neg fly, ab\}\} \end{array} \right\rangle$$

Therefore we have $S' \models_{para} not\, fly$ and $S' \models_{para} not \neg fly$. This means that ab is undefined and that we have fly, $not\, fly$ and $not \neg fly$ simultaneously true! This behaviour does not seem to be realized (and intended) by the proposers of the semantics. There is no stable structure for this program because of the odd-loop through negation introduced by the rule for *ab*.

Besides the above mentioned problems, both the stable and well-founded structures do not cater for propagation of contradiction through negation by default, as in the suspicious well-founded and p-stable models semantics of [80].

Example 46. 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 a single stable structure which coincides with the well-founded one, namely $< \{b, c, \neg c\}, \{\{b, c\}, \{b, \neg c\}\} >$. Accordingly we have $not\, a$, b, $not\, c$, $not \neg c$ and $\mathbf{C}c$ true in that structure. One of the interesting points of this semantics is that neither c nor $\neg c$ hold in the above program, since they are contradictory. Now, suppose we remove one of c or $\neg c$. In the resulting programs, and the associated well-founded structures and unique stable structure 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 42. However, the approach followed in [93] is not capable of detecting this situation.

7.3 Introspective well-founded model

In Section 6.3 we have identified for $WFSX_{\mathrm{p}}$ 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.

We've adopted the ideas of [93] in our semantics, including the terminology, and we will show how their two operators can be implemented in $WFSX_{\mathrm{p}}$ by means of a program transformation into *WFS*, without recourse to modalities.

We slighty extend the language of introspective programs by allowing in our language the application of default negation to the modalities.

Our approach is to make use of the P^{T-TU} program transformation presented in Section 4.2 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 61. 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} for every atom A in the language of the program:

- If $\mathbf{C}A$ then include $\mathbf{C}A \leftarrow A^p_T, A^n_T$;
- If $\mathbf{C}\neg A$ then include $\mathbf{C}\neg A \leftarrow A^p_T, A^n_T$;
- If $\mathbf{C_d}A$ then include $\mathbf{C_d}A \leftarrow A^p_T, not\, A^p_{TU}$;
- If $\mathbf{C_d}\neg A$ then include $\mathbf{C_d}\neg A \leftarrow A^n_T, not\, A^n_{TU}$.

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

First, notice that the translation of the introspective operators in T and TU rules is the same. Therefore, an introspective operator has the same truth-value in T rules and in TU rules in the P^{int} program. So, the situation never occurs that the introspective operator and its default negation simultaneously hold. This accomplishes the desired blocking of propagation of inconsistencies. For instance, if a literal L is contradictory then $\mathbf{C}L$ holds but not $not\, \mathbf{C}L$. Second, the bodies of the rules which implement the new operators capture the two conditions previously discussed. The semantics of the introspective programs is now obtained straightforwardly.

Definition 62. Let P be an introspective program generating 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 $a^p_T \in WFM(P^{int})$;
- $\neg a \in WFM_i(P)$ iff $a^n_T \in WFM(P^{int})$;
- $not\, a \in WFM_i(P)$ iff $not\, a^p_{TU} \in WFM(P^{int})$;
- $not\, \neg a \in WFM_i(P)$ iff $not\, a^n_{TU} \in WFM(P^{int})$.

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

Example 47. Consider the program of Example 42. 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 from the resulting model of [93], presented in Example 42. The above literals are derived irrespectively of whether they 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, by reprogramming, to detect the contradiction and block its propagation whenever desired. For example, if we 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: only a is retained.

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 [93]. This feature is illustrated in the next example.

Example 48. 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.

Example 49. Consider the program of Example 44:

$$a \leftarrow b. \qquad a \leftarrow \mathbf{C_d}a. \qquad b. \qquad \neg b.$$

In the introspective well-founded model of the above program we have $\mathbf{C_d}a$ undefined. We still have a, b and $\neg b$ true. Even though $not\, b$ and $not\, \neg b$ are true it is not the case that $not\, a$ is true. Recall that the minimal para-model is not defined for this program. Our semantics solves the problem by saying that a does not depend on contradiction and by letting $\mathbf{C_d}a$ undefined. The intuitiveness of this result might be questionable.

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 the construction of Definition 50 is needed to determine the dependencies on contradiction, the results might be unexpected: the integration of the introspective operators into our language destroys the monotonicity of the construction of the suspicious $WFSX_{\mathrm{p}}$. The transfinite sequence of programs as per Definition 50 monotonically decreases when we substitute the WFM_p by WFM_i in the Δ operator. However, we have no more the corresponding inclusion

$$WFM_i(P) \subseteq WFM_i(\Delta P).$$

For the problems involved see the next example.

Example 50. 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 sequence of programs and associated models $M_i = WFM_i(P_i)$ is:

$$\begin{array}{l} P_0 = P \\ M_0 = \{a, c, d, \neg d\} \cup not\,\{\neg a, b, \neg b, \neg c, d, \neg d\} \\ P_1 = P_0 - \{a \leftarrow u, c \leftarrow u\} \\ M_1 = \{b, c, d, \neg d\} \cup not\,\{a, \neg a, b, \neg b, c, \neg c, d, \neg d\} \end{array}$$

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 and no loops over the introspective operators, as secured by Theorem 48.

For programs in the above conditions, and when the minimal para-model exists, we conjecture a relationship with the construction of [93]. Basically, the literals which do not depend on contradiction coincide in both approaches, as do the introspective operators. For the remaining literals, we entail them in our semantics while in the minimal para-model of [93] they are not entailed. On this conjecture, it is immediate to obtain the minimal para-model of a paraconsistent program from the corresponding introspective well-founded model.

The integration of the techniques presented here with the suspicious $WFSX_{\mathrm{p}}$ are the subject of our current research. This is an open problem in need of attention. One may proceed, as we have pointed out before, by defining an appropriate notion of stratification with respect to the $\mathbf{C}$ and $\mathbf{C_d}$ operators. At least for this class of so stratified programs we expect the correct results.

8 Other Paraconsistent Semantics

In this section we briefly overview other paraconsistent semantics for generalizations of logic programming, but in a direction different from the one of extended logic programs.

The use and study of multi-valued logics in an integrated fashion, in particular bilattices, and for applications in AI is mainly due to Ginsberg [37]. The logics discussed in this work require the existence of a $\top$ truth-value. However, as pointed out in [81], the adoption of his fundamental principle by the proposed logics, that of apparently-closed truth-assignments, gives rise to the collapse of his framework in the presence of contradiction. Thus, from a paraconsistency point of view the work of [37] is mostly irrelevant for our purposes.

Paraconsistent logic programming originated with the works of Blair and Subrahmanian [13]. Their work has been extended in order to consider disjunctive rules [84], and arbitrary upper semilattices of truth-values [42, 43]. The work in [42], on generalized annotated logic programs (GAP), allows variable annotations in the head or body of a rule as well as total computable function annotations in the head. The language does not permit default negation. Two semantics are provided for these programs, one ideal-theoretic and another along the lines of Definition 11. It is shown that the former is continuous and that the latter is monotonic but not continuous. Sound and complete proof-procedures are discussed for the ideal-theoretic semantics whenever the set of truth values is a lattice. The framework of GAP's theory is applied to obtain reformulations of several multivalued and temporal formalisms. A more recent application of these semantics to the problem of amalgamation of knowledge bases is discussed in [1].

The work on annotated predicate calculus [43] only allows constant annotations of atomic first-order formulae with truth values of an upper semilattice (the belief semilattice BSL). This work considers two forms of negation and implication: ontological "$\neg$" and epistemic "$\sim$". The former has a behaviour similar to that of classical negation and the latter is a generalization of explicit negation to BSLs. Ontological implication is defined as usual "$\psi \leftarrow \phi \equiv \psi \vee \neg\phi$" and epistemic implication by the equivalence "$\psi \lessapprox \phi \equiv \psi \leftarrow (\phi \wedge \neg \sim \phi)$". Using an entailment relation which minimizes inconsistency, the former implication is able to draw conclusions from inconsistent beliefs while the latter blocks them. A proof theory is presented for the usual entailment relation. It is also argued that inconsistent beliefs in APC represent the causes of inconsistency in predicate calculus. An ideal-theoretic semantics is also presented.

In [12] an approach based on default logic is used to provide a semantics for arbitrary propositional theories. The idea is to view A and $\neg A$ as separate entities and incrementally restore the relation between those pairs till an inconsistency is generated. Several semantics are discussed based on this simple bright idea, with differente degrees of skepticism or credulity. Such semantics can be readily applicable to definite extended logic programs.

A different line of work has been followed by Fitting. In [28] he provides a Kripke/Kleene semantics for logic programs under a richer range of logics than the classical by resorting to bilattices. An illuminating discussion of the problems and solutions is presented. The main result is the extension of Fitting's operator Φ_P [27] and associated results for a broader class of logic programs: compound logic programs. Compound logic programs allow the use of operators "$\otimes$" (consensus) and "$\oplus$" (accept-anything) for the representation of distributed logic programs. Deep results concerning the relationship of the least and greatest fixpoints of Φ_P with respect to the knowledge and truth orderings are presented. This work was continued in [29] for a similar generalization of well-founded semantics, but with the syntax limited to normal logic programs. The main criticism leveled at these works is that the syntax of the programs is not as general as the underlying logic, and thus some knowledge representation limitation exists. In particular, and under very general conditions, the models returned are exact and therefore consistent. The underlying logic may be paraconsistent but the models provided are restricted to their consistent subset. In some sense this is due to the absence of an explicit form of negation. This criticism is yet more pertinent for the extensions of *WFS*.

Pradhan has described [67] a truth-functional five-valued paraconsistent logic, dubbed C_5, which subsumes the answer-sets based semantics for extended logic programs. This work is based on the idea of well-supported models of [26]. The underlying semantics is paraconsistent but the resulting entailment relation is always consistent.

In [30] the authors define two logics, the **L**ogic of **A**rgumentation (LA) and the **M**eta-**L**ogic of **A**rgumentation (MLA). With logic LA, Toulmin's notions of support and confirmation of arguments are formalized [86].

Supported arguments are uncertain by nature and confirmed arguments the certain ones. To each formula in their logic is assigned either the sign '+' or '++', meaning respectively that the formula is supported or confirmed. A natural deduction scheme based on propositional minimal logic is provided for logic LA. One remarkable rule is "Weakening": from $\varphi : ++$ infer $\varphi : +$. There is a natural correspondence from extended logic programming to LA logic: L, $\neg L$,*not* $\neg L$ and *not* L correspond, respectively, to $L : ++$, $\neg L : ++$, $L : +$ and $\neg L : +$. With this interpretation in mind it is easy to see that their *Weakening* rule is nothing more than the coherence principle.

The logic MLA is used to perform reasoning with contradictory knowledge. The authors assume that contradictory knowledge is permissible only if supported by distinct viewpoints/assumptions. They then define a meta-logic of argumentation, where each argument for a proposition can be labeled with addi-

tional signs, representing different degrees of inconsistency. Given "two distinct theories" the possible pairs of contradictions are described in Table 8.

Table 13. Distinct contradictory pairs of logic LA

Contradictory pairs	Sign	Meaning
$(\varphi : ++) \,\&\, (\neg\varphi : ++)$	**I**	Strong claim/argument rebuttal
$(\varphi : ++) \,\&\, (\neg\varphi : +)$	**II**	Strong claim/argument discount
$(\varphi : +) \,\&\, (\neg\varphi : ++)$	**III**	Weak claim/argument rebuttal
$(\varphi : +) \,\&\, (\neg\varphi : +)$	**IV**	Weak claim/argument discount

These four signs correspond to truth values in logic $\mathcal{NINE}$. The similarities of $WFSX_{\mathrm{p}}$ with logics LA and MLA end here because other mechanisms are available in our $WFSX_{\mathrm{p}}$ semantics to infer weak information, namely the default reasoning capabilities, and we are also more liberal, allowing any form of contradiction to appear in extended logic programs. One disadvantage of logic LA is that from a contradiction the negation of every proposition is entailed, because of its minimal logic basis. For a more recent (proof-theoretical) discussion of logics LA and MLA consult [45].

Another semantics based on minimal logic appeared recently in [17]. Three-valued interpretations like the ones of OD-semantics [40] are used for assigning meaning to normal logic programs, subsuming the stable models semantics. The whole purpose of this work is to give a compositional semantics for stable models which is defined for all normal logic programs. To handle the rule $a \leftarrow not\, a$ the author assigns the interpretations $\{a, \neg a, \bot\}$ to the program. The conclusion is that another logical value is required to treat such programs. While in well-founded semantics the value undefined is used, in Bry's intended models $\top$ is used instead.

The work reported in [77] follows the potential-contradictions view and deals with the problem of maintaining consistency in a dynamic updatable deductive database. Basically, they start from a consistent database and update it with self-consistent formulae. In order to keep consistency, the newer information has precedence over the older. An algorithm is proposed to perform queries over such databases.

Finally, a modal logic has been proposed in [48], 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 concerned with the axiomatization of these operators and applies them to some knowledge representation problems. The relationships to logic programming are not explored. But the ideas behind the two operators are very similar to the ones in [93].

9 Discussion and Conclusions

Due to the extension of the paper (nearly twenty semantics have been treated) we organize this conclusions section along several general features that we classify as the most important. We hope that we've convinced the reader of the need of paraconsistent semantics and that logic programming can provide the adequate tools for knowledge representation applications which require such reasoning forms. Depending on the situation the reader surely will find interesting applications of the results reported here.

Semantical Concepts. The study of extended logic programming can be traced back to the works of Blair and Subrhamanian [13] on paraconsistent logic programs. We have shown that their Generalized Horn Programs are equivalent to Wagner's Logic Programs with Strong Negation [91] and to extended logic programs without default negated literals. This is the common basis on which, with the exception of the semantics discussed in Section 7, all extended logic programming semantics agree on, and has its own roots in the works on paraconsistent constructive logics [55, 6]. Belnap's logic [11] provides the underlying model theory for this semantics. The over-determined semantics of [40] extends GHPs by introducing case analysis and the validity of the law of the excluded middle.

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 [61], relating the two forms of negation, as basic and provides a localized "explosion" of consequences when faced with contradiction; our $WFSX_{\mathrm{p}}$ and its extensions are the only representative of this type of semantics; 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 [30], which also identifies two strengths of belief and four forms of inconsistency.

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 [71, 78] 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 [56, 57, 80] which are variants of the Answer Sets semantics [35]. Weak Answer Sets and Paraconsistent Stable Models are related to Almukdad and Nelson's logic system N^- [6] as shown in [57]. To overcome the problem of non-existing models for some extended logic programs Sakama defined the Semi-Stable Models semantics, but it suffers from problems in the treatment

of "*undefined*" literals. An extension of the over-determined semantics with a stable-models substratum is also given in [40].

An interesting remark is that most of the paraconsistent semantics proposals for extended logic programs use fairly standard techniques of normal logic programming semantics. First, the semantics is defined for default negation free case. Then, if the behaviour of default negation is to be similar to the one in WFS, an alternating fixpoint definition is pursued. If the *not* should be stable-models like then a corresponding fixpoint equation is used to define the semantics for the general case. The notable exception is Semi-Stable Models which rely on a semantics for disjunctive logic programs.

None of the referred semantics cater for the problem of detecting support on contradiction. This new feature was first touched upon by Sakama and Inoue, with the suspicious well-founded model [78] and suspicious p-stable models [80], extending *EWFS* [78] and Paraconsistent Stable Models [80]. As we have shown, their notions of support do not cope well with default negation. For the same purpose and for well-founded based semantics, the suspicious $WFSX_{\mathrm{p}}$ [21] is much more adequate, since detection of support on contradiction is guaranteed for arbitrary extended logic programs.

Besides detecting unsafe conclusions, it is sometimes necessary to block the propagation of contradiction. An argumentation approach to this problem is adopted by Wagner in [90] with his credulous, conservative, and skeptical inference relations. The last two are inherently consistent. However, none of the three semantics are reflexive and nor obey Modus Ponens. The introduction of the introspective operators of [93] brings in new fresh issues. There, a full-fledged paraconsistent framework for introspective programs allowing disjunction is provided. The programs can test in their rules whether a literal is contradictory or depends on contradiction. However, their semantics is ill-defined for some classes of programs and does not handle well the detection of support on contradiction in paraconsistent programs with default negation.

By means of a program transformation the $WFSX_{\mathrm{p}}$ semantics has been extended to treat these new modalities [21]. The advantages of this semantics is the adoption of the coherence principle, and a liberal way of propagating consequences of contradictory literals relying on it. But the introspective well-founded model, in the presence of non-stratified occurrences of introspective operators, might not produce the intended results whenever the model contains undefined literals. This occurs whenever there are supports on contradictions which are not properly propagated.

The family tree of the semantics surveyed on this chapter can be found in Figure 9. The non-obvious abbreviations on that figure are as follows. PSM, WAS, and SE stand for, respectively, Paraconsistent Stable Models, Weak Answer Sets and Stable Environments. Suspicious well-founded and p-stable models are shortened to WFS^s and PSM^s, respectively. The suspicious $WFSX_{\mathrm{p}}$ semantics is represented by $WFSX_p^s$, while introspective $WFSX_{\mathrm{p}}$ by $WFSX_p^i$. The well-founded and stable structures are abridged to WS and SS. The same designations will be used in the remainder of this section.

[ht]

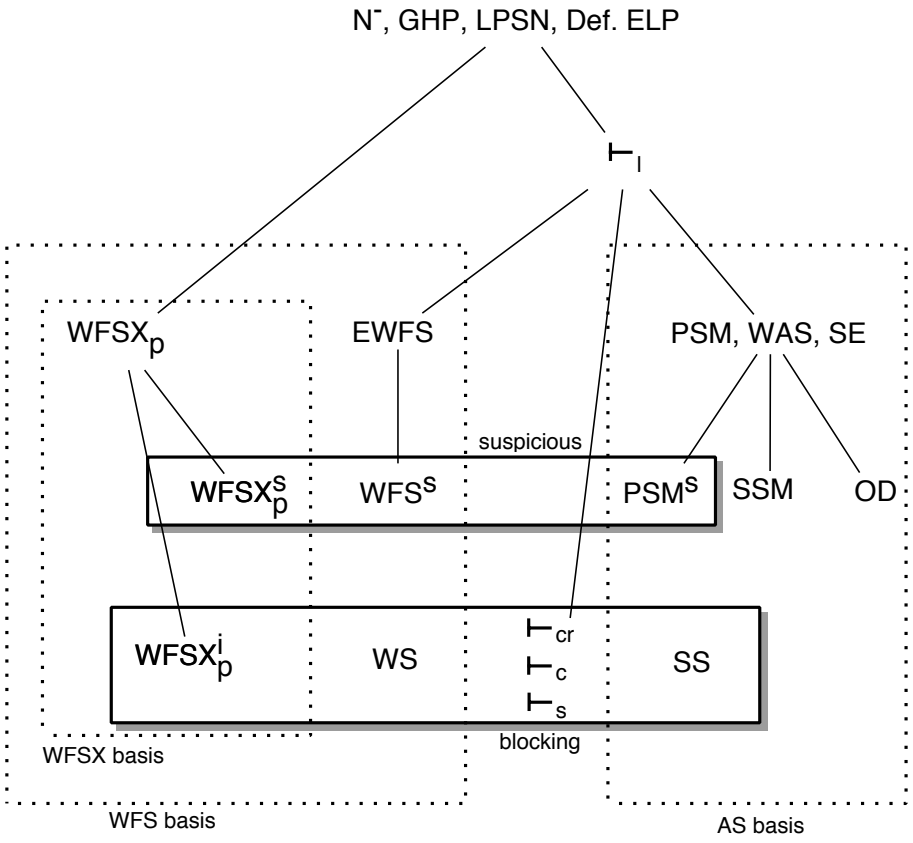

Fig. 9. Paraconsistent logic programming semantics' genealogy

Existence of Semantics. The introspective $WFSX_{\mathrm{p}}$ is defined for the whole class of introspective programs, subsuming the one for extended logic programs. As we have seen, well-founded structures are defined for every extended logic programming. Its existence is not guaranteed for arbitrary introspective programs. We suspect that programs without positive loops through the $\mathbf{C_d}$ operator are also assigned a well-founded structure. Stable-structures are not defined for the whole class of extended logic program because of the problems due to infinite recursion through negation by default. Regarding introspective programs, the problems advanced for well-founded, plus the additional ones originated by the stable-models basis, carry over for stable structures.

The semantics which are defined for the whole class of extended logic pro-

grams are Semi-Stable Models, $EWFS$, WFS^s, $WFSX_{\mathrm{p}}$, and $WFSX_p^s$. Then comes Wagner's extended deductive bases under liberal and skeptical reasoning, defined for weakly well-founded XDBs. Finally, we find the class of strongly well-founded XDBs under credulous and conservative reasoning, the class of Generalized Horn Programs and Logic Programs with Strong negation, which are strictly smaller than the previous one but incomparable with each other.

The class of programs defined for Answer Sets and the one for Paraconsistent Stable Models appear in between the classes of liberal reasoning and of Semi-stable semantics. The suspicious p-stable models are defined whenever the PSMs are. The OD-semantics is defined for every definite extended logic program, and for the class of extended logic programs where loops through negation do not occur. It is difficult to identify the class of programs for which stable structures are defined. They are guaranteed to exist for (locally) stratified extended logic programs without introspective operators.

Embeddings and complexity results. One of the main results of this work is the equivalence-preserving polynomial program transformations, from most of the extended logic programming approaches into normal logic programming. These transformations allow us to compare the expressiveness of the various approaches, as well as their complexity by resorting to the results for normal logic programming. Furthermore, one can use the current implementations of normal logic programming systems for implementing these new paraconsistent reasoning forms.

Regarding the expressive power of the several semantics, we have identified the following equivalence classes, in increasing order of expressiveness:

- Generalized Horn Programs, and Logic Programs with Strong Negation. Mapped to positive definite logic programs by means of the $P^{\neg}$ transformation;
- Liberal, Credulous, Conservative, and Skeptical reasoning. Mapped to locally stratified programs by the P^{l}, P^{cr}, P^{co} and P^{s} transformations;
- $WFSX_{\mathrm{p}}$, $WFSX_p^i$, and Sakama's extended and suspicious well-founded semantics. Our semantics are mapped to WFS by making use of the P^{T-TU} transformation and its extension for introspective operators P^{int}, respectively. $EWFS$ is mapped to WFS again by $P^{\neg}$;
- Weak Answer Sets, Stable Environments, Paraconsistent Stable Models, and Suspicious p-stable Models. The first three are transformable into normal logic programs by means of the $P^{\neg}$ transformation;

The first three equivalence classes were for the first time identified in the present work. Semi-stable models and OD-semantics are at least as expressive as the Weak Answer Sets class. It is possible that these two semantics coalesce into the latter class. Similarly, the suspicious $WFSX_{\mathrm{p}}$ is at least as expressive as the class of $WFSX_{\mathrm{p}}$. We do know that is less expressive than the class of WASs.

Accordingly, we adapted the known complexity results for normal logic programs of [82] to extended logic programs. These results can be found in Table 14.

There $|$ P $|$ is length of the program, $|\ \mathcal{A}\ |$ the number of atoms in the langauge of P, and E is the extensional database. It can be easily shown that for the propositional case the suspicious $WFSX_{\mathrm{p}}$ has worst case complexity $O(|\ P\ |^2\ .\ |\ \mathcal{A}\ |)$. The complexity results for stable and well-founded structures are not known.

Table 14. Complexity Results

Semantics	Propositional	DATALOG
GHP, $LPSN$	O($|$ P $|$)	inductive in E
$\vdash_l, \vdash_{cr}, \vdash_{co}, \vdash_s$	O($|$ P $|$)	inductive in E
$EWFS$,WFS^s,$WFSX_{\mathrm{p}}$,$WFXS^i_p$	O($|$ P $|$. $|\ \mathcal{A}\ |$)	comp. ind. in E
PSM, PSM^s, WAS, SE	Co-NP-comp.	Co-NP-comp.

Note that the translation of extended logic programs into normal ones is achieved in most cases by the $P^{\neg}$ program transformation. Using a variant of the same transformation, liberal reasoning in extended deductive bases is also captured by logic programs. The $P^{\neg}$ approach was also followed in the definition of disjunctive extended logic programming semantics in [52].

The coherent semantics, $WFSX_{\mathrm{p}}$ and $WFSX^i_p$, rely on the P^{T-TU} transformation for translation into normal logic programs. This completely clarifies the different results and properties obtained in our semantics. There are several forms of explicit negation!

Logical Properties. In our study we have described six different many-valued logics. Surprisingly (or maybe not...), this is where most of the results converge and are more noticeable. First, the logics are defined in terms of designated truth-values. A remarkable property is that this set is obtained from the truth-values lying in the line connecting $\mathbf{t}$ to $\top$. Furthermore, the implication sign is two valued and defined in such a way that the following properties plus Modus Ponens hold in every logic:

$$\begin{array}{l} I \models A \wedge B \ \text{ iff } I \models A \text{ and } I \models B \\ I \models A \vee B \ \text{ iff } I \models A \ \text{ or } \ I \models B \\ I \models A \leftarrow B \text{ iff } I \models A \ \text{ or } \ I \not\models B \end{array}$$

A set of classical truths obeyed by all logics is given in Table 15. Recall that we are interested mostly in the weaker equivalence relation $\models\models$. The explicit negation behaviour is fairly standard among the several logics, both in the theorems which are satisfied and the falsified ones, as can be seen from Table 15 and Table 16. Mark that the OD-semantics is the single one which enforces the excluded middle property for explicit negation. Furthermore, Modus Tollens and Disjunctive Syllogism are unsound rules for explicit negation in all logics addressed. Coherence is only satisfied in $\mathcal{NINE}$.

Table 15. Propositional theorems verified by the logics

Commutativity	$(A \vee B)$	$\models\!\models$	$B \vee A$
	$(A \wedge B)$	$\models\!\models$	$B \wedge A$
Associativity	$(A \vee B) \vee C$	$\models\!\models$	$A \vee (B \vee C)$
	$(A \wedge B) \wedge C$	$\models\!\models$	$A \wedge (B \wedge C)$
Distributivity	$A \vee (B \wedge C)$	$\models\!\models$	$(A \vee B) \wedge (A \vee C)$
	$A \wedge (B \vee C)$	$\models\!\models$	$(A \wedge B) \vee (A \wedge C)$
Idempotent Laws	$A \vee A$	$\models\!\models$	A
	$A \wedge A$	$\models\!\models$	A
Absorption Laws	$A \vee (A \wedge B)$	$\models\!\models$	A
	$A \wedge (A \vee B)$	$\models\!\models$	A
Zero Laws	$A \vee \mathbf{f}$	$\models\!\models$	A
	$A \wedge \mathbf{f}$	$\models\!\models$	$\mathbf{f}$
One Laws	$A \vee \mathbf{t}$	$\models\!\models$	$\mathbf{t}$
	$A \wedge \mathbf{t}$	$\models\!\models$	A
Implication Laws	$(A \leftarrow B) \wedge (A \leftarrow C)$	$\models\!\models$	$A \leftarrow (B \vee C)$
	$(A \leftarrow B) \vee (A \leftarrow C)$	$\models\!\models$	$A \leftarrow (B \wedge C)$
	$(B \leftarrow A) \wedge (C \leftarrow A)$	$\models\!\models$	$(B \wedge C) \leftarrow A$
	$(B \leftarrow A) \vee (C \leftarrow A)$	$\models\!\models$	$(B \vee C) \leftarrow A$
	$(A \leftarrow B) \leftarrow C$	$\models\!\models$	$A \leftarrow (B \wedge C)$
De Morgan Laws	$\neg(A \vee B)$	$\models\!\models$	$(\neg A) \wedge (\neg B)$
	$\neg(A \wedge B)$	$\models\!\models$	$(\neg A) \vee (\neg B)$
Double Neg. Law	$\neg\neg A$	$\models\!\models$	A
Negations Swap	$\neg \mathit{not}\, A$	$\models\!\models$	$\mathit{not}\, \neg A$

The distinct features appear in the definition and properties of the *not* operator. Logics **IV** and **VI** treat it as classical negation. The De Morgan laws for default negation are obeyed by all logics, with the exception of **VII**. Logics **VII**, **IX** and $\mathcal{NINE}$ agree that *not* should not obey the excluded middle, contraposition, and definition laws, as well as Modus Tollens. This is not strange since all these logics allow an undefined value for default negation, for handling infinite recursion through negation by default. The distinguishing features of default negation in $\mathcal{NINE}$ are evident from the fact that neither the contradiction nor double negation laws nor Disjunctive Syllogism are valid. However, the implication $A \rightarrow \mathit{not}\, \mathit{not}\, A$ is always satisfied in $\mathcal{NINE}$.

Under the stronger equivelence relation $\equiv$ the several logics are rather distinct. However, these results are not particularly relevant for our characterization of the models of logic programs.

We hope to have contributed to fostering the use of paraconsistent logic programming by showing it has become of age.

Table 16. Explicit negation theorems falsified by all logics

Excluded Middle*	$A \vee \neg A$	$\not\models$	$\mathbf{t}$
Contradiction Law	$A \wedge \neg A$	$\not\models$	$\mathbf{f}$
Contraposition Laws	$(A \leftarrow B)$	$\not\models$	$\neg B \leftarrow \neg A$
Definition Law	$(A \leftarrow B)$	$\not\models$	$A \vee \neg B$

Coherence** If $I \models \neg L$ then $I \models not\, L^*$

$*$ With the single exception of OD-semantics.
$**$ With the single exception of logic $\mathcal{NINE}$, which obeys it.

Acknowledgments

We acknowledge the support of project MENTAL (PRAXIS XXI contract number 2/2.1/TIT/1593/95) and JNICT, Portugal. We also thank Gerd Wagner for helpful discussions.

Carlos Viegas Damasio
Universidade Aberta, and Universidade Nova de Lisboa, Portugal.

Luis Moniz Pereria
Universidade Nova de Lisboa, Portugal.

References

1. S. Adali and V. S. Subrahmanian. Amalgamating knowledge bases, III: Algorithms, data structures, and query processing. *Journal of Logic Programming*, 28(1):45–88, 1996.
2. J. J. Alferes. *Semantics of Logic Programs with Explicit Negation.* PhD thesis, Universidade Nova de Lisboa, October 1993.
3. J. J. Alferes, C. V. Damásio, and L. M. Pereira. A logic programming system for non-monotonic reasoning. *Special Issue of the Journal of Automated Reasoning*, 14(1):93–147, 1995.
4. J. J. Alferes and L. M. Pereira. *Reasoning with Logic Programming*, volume LNAI 1111. Springer–Verlag, 1996.
5. J. J. Alferes, L. M. Pereira, and T. Przymusinski. Strong and explicit negation in non-monotonic reasoning and logic programs. In *JELIA'96, European Workshop on Logic in Artificial Intelligence*. Springer-Verlag, 1996. An extended version will appear in JAR.
6. A. Almukdad and D. Nelson. Constructible falsity and inexact predicates. *Journal of Symbolic Logic*, 49:231–233, 1984.
7. J. N. Aparício. *Logic Programming: a tool for reasoning.* PhD thesis, Universidade Nova de Lisboa, January 1994.
8. K. Apt and M. Bezem. Acyclic programs. *New Generation Computing*, 9(3):335–363, 1991. Also appeared in ICLP'90.

9. K. Apt and R. Bol. Logic programming and negation: A survey. *Journal of Logic Programming*, 19,20:9–71, 1994.
10. C. Baral and M. Gelfond. Logic programming and knowledge representation. *Journal of Logic Programming*, 19/20:73–148, 1994.
11. N. D. Belnap. A useful four-valued logic. In J. M. Dunn and G. Epstein, editors, *Modern Uses of Many-valued Logic*, pages 8–37. D. Reidel, 1977.
12. P. Besnard and T. Schaub. Signed systems for paraconsistent reasoning. In J. J. Alferes, L. M. Pereira, and E. Orlowska, editors, *Logics in Artificial Intelligence. Proceedings of the European Ws. JELIA'96*, volume LNAI 1126, pages 404–416. Springer–Verlag, 1996.
13. H. A. Blair and V. S. Subrahmanian. Paraconsistent logic programming. *Theoretical Computer Science*, 68:135–154, 1989.
14. P. Bonatti. Autoepistemic logics as a unifying framework for the semantics of logic programs. In K. Apt, editor, *International Joint Conference and Symposium on Logic Programming*, pages 417–430. MIT Press, 1992.
15. F. Bry. Logic programming as constructivism: a formalization and its application to databases. In *Proc. of the Eighth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'89)*, pages 34–50, 1989.
16. F. Bry. Negation in logic programming: a formalization in constructive logic. In *Information Systems and Artificial Intelligence: Integration Aspects*, pages 30–46. Springer–Verlag, 1990.
17. F. Bry. A compositional semantics for logic programs and deductive databases. In *Proc. Joint International Conference and Symposium on Logic Programming'96 (JICSLP'96)*, pages 453–467. The MIT Press, 1996.
18. N. Costa. On the theory of inconsistency formal system. *Notre Dame Journal of Formal Logic*, 15:497–510, 1974.
19. C. V. Damásio. *Paraconsistent Extended Logic Programming with Constraints*. PhD thesis, Universidade Nova de Lisboa, October 1996.
20. C. V. Damásio and L. M. Pereira. A model theory for paraconsistent logic programming. In C. Pinto-Ferreira and N. J. Mamede, editors, *Progress in Artificial Intelligence - 7th Portuguese Conference on Artificial Intelligence*, LNAI 990, pages 377–386. Springer-Verlag, 1995.
21. C. V. Damásio and L. M. Pereira. A paraconsistent semantics with contradiction support detection. In J. Dix and A. Nerode, editors, *Proceedings of LPNMR'97*, Lecture Notes in Artificial Intelligence. Springer–Verlag, 1997.
22. J. Dix. A Classification-Theory of Semantics of Normal Logic Programs: I. Strong Properties. *Fundamenta Informaticae*, XXII(3):227–255, 1995.
23. J. Dix. A Classification-Theory of Semantics of Normal Logic Programs: II. Weak Properties. *Fundamenta Informaticae*, XXII(3):257–288, 1995.
24. W. Drabent and M. Martelli. Strict completion of logic programs. *New Generation Computing*, 9(1):69–79, 1991.
25. M. V. Emden and R. Kowalski. The semantics of predicate logic as a programming language. *Journal of ACM*, 4(23):733–742, 1976.
26. F. Fages. A new fixpoint semantics for general logic programs compared with the well-founded and stable semantics. *New Generation Computing*, 9:425–443, 1991.
27. M. Fitting. A Kripke-Kleene semantics for logic programs. *Journal of Logic Programming*, 2(4):295–312, 1985.
28. M. Fitting. Bilattices and the semantics of logic programming. *Journal of Logic Programming*, 11:91–116, 1991.

29. M. Fitting. Well-founded semantics, generalized. In *Proceedings of ILPS'91*, pages 71–84. MIT Press, 1991.
30. J. Fox, P. Krause, and S. Ambler. Arguments, contradictions and practical reasoning. In B. Neumann, editor, *Proceedings ECAI'92*, pages 623–627. John Wiley & Sons, 1992.
31. A. V. Gelder. The alternating fixpoints of logic programs with negation. In *8th Symposium on Principles of Database Systems*. ACM SIGACT-SIGMOD, 1989.
32. A. V. Gelder, K. A. Ross, and J. S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, 1991.
33. M. Gelfond. On stratified autoepistemic theories. In *AAAI'87*, pages 207–211. Morgan Kaufmann, 1987.
34. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R. Kowalski and K. A. Bowen, editors, *5th International Conference on Logic Programming*, pages 1070–1080. MIT Press, 1988.
35. M. Gelfond and V. Lifschitz. Logic programs with classical negation. In Warren and Szeredi, editors, *7th International Conference on Logic Programming*, pages 579–597. MIT Press, 1990.
36. M. L. Ginsberg. Multivalued logics. In *Proceedings of AAAI'86*, pages 243–247, 1986.
37. M. L. Ginsberg. Multivalued logics: a uniform approach to reasoning in artificial intelligence. *Computational Intelligence*, 4:265–316, 1988.
38. J. Grant. Inconsistent and incomplete logics. *Mathematics Magazine*, 48(3):154–159, 1975.
39. J. Grant. Classifications for inconsistent theories. *Notre Dame Journal of Formal Logic*, XIX:435–444, 1978.
40. J. Grant and V. S. Subrahmanian. Reasoning in inconsistent knowledge bases. *IEEE Transactions on Knowledge and Data Engineering*, 7(1):177–189, 1995.
41. C. M. Jonker and C. Witteveen. Revision by expansion. In G. Lakemeyer and B. Nebel, editors, *Proceedings ECAI'92 Workshop on Theoretical Foundations of Knowledge Representation*, pages 40–44. ECAI'92 Press, 1992.
42. M. Kifer and E. Lozinskii. A logic for reasoning with inconsistency. *Journal of Automated Reasoning*, 8:179–215, 1992.
43. M. Kifer and V. Subrahmanian. Theory of generalized annotated logic programming and its applications. *Journal of Logic Programming*, 12:335–367, 1992.
44. R. Kowalski and F. Sadri. Logic programs with exceptions. In Warren and Szeredi, editors, *ICLP90*. MIT Press, 1990.
45. 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.
46. H. Levesque. Making believers out of computers. *Artificial Intelligence*, 30:81–107, 1986.
47. H. Levesque. Logic and the complexity of reasoning. *Journal of Philosophical Logic*, 17:355–389, 1988.
48. J. Lin. A semantics for reasoning consistently in the presence of inconsistency. *Artificial Intelligence*, 86:75–95, 1996.
49. J. Lukasiewicz. Philosophische bemerkungen zu mehrwertigen systemen des aussagenkalkiils. *Comptes rendus de la Societé des Sciences et Lettres de Varsovie*, 23:51–77, 1930.
50. W. Marek and M. Truszczyński. *Nonmonotonic Logic - Context-Dependent Reasoning*. Springer-Verlag, 1993.

51. B. Meltzer. Theorem-proving for computers: some results on resolution and renaming. *Automatation of Reasoning*, 1:493–495, 1983.
52. J. Minker and C. Ruiz. Semantics for disjunctive logic programs with explicit and default negation. *Fundamenta Informaticae*, 20(3/4):145–192, 1994.
53. R. C. Moore. Possible-world semantics for autoepistemic logic. In *Proc. AAAI Workshop on Non-monotonic Reasoning*, pages 396–401, New Paltz, 1984.
54. R. C. Moore. Semantical considerations on nonmonotonic logic. *Artificial Intelligence*, 25:75–94, 1985.
55. D. Nelson. Constructible falsity. *Journal of Symbolic Logic*, 14:16–26, 1949.
56. 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*, number 619 in LNAI, pages 63–79. Springer–Verlag, 1992.
57. D. Pearce. Answer sets and constructive logic, II: Extended logic programs and related non-monotonic formalisms. In L. Pereira and A. Nerode, editors, *Logic Programming and Nonmonotonic Reasoning - proceedings of the second international workshop*, pages 457–475. MIT Press, 1993.
58. D. Pearce. Safety, stability and deductive bases. Technical report, DFKI, 1995.
59. 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.
60. D. Pearce and G. Wagner. Logic programming with strong negation. In P. Schroeder-Heister, editor, *Extensions of Logic Programming*, pages 311–326. LNAI 475, Springer–Verlag, 1991.
61. 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 1992*, pages 102–106. John Wiley & Sons, 1992.
62. 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 Non-monotonic Reasoning*, pages 105–119. MIT Press, 1991.
63. L. M. Pereira, C. V. Damásio, and J. J. Alferes. Debugging by diagnosing assumptions. In P. A. Fritzson, editor, *Automatic Algorithmic Debugging, AADEBUG'93*, LNCS 749, pages 58–74. Springer–Verlag, 1993.
64. L. M. Pereira, C. V. Damásio, and J. J. Alferes. Diagnosis and debugging as contradiction removal. In L. M. Pereira and A. Nerode, editors, *2nd Int. Workshop on Logic Programming and Non-Monotonic Reasoning*, pages 334–348, Lisboa, Portugal, 1993. MIT Press.
65. L. M. Pereira, C. V. Damásio, and J. J. Alferes. Diagnosis and debugging as contradiction removal in logic programs. In L. Damas and M. Filgueiras, editors, *Progress in Artificial Intelligence. Proceedings of the 6th Portuguese AI Conf.*, LNAI 727, pages 183–197, Porto, Portugal, 1993. Springer–Verlag.
66. 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, *Logic Programming and Non-monotonic Reasoning*, pages 228–242. MIT Press, 1991.
67. S. Pradhan. A family of paraconsistent semantics for extended logic programs. Technical report, CS, University of Maryland, 1996.

68. G. Priest, R. Routley, and J. Norman. *Paraconsistent logics*. Philosophia Verlag, 1988.
69. H. Przymusinska and T. C. Przymusinski. Semantic issues in deductive databases and logic programs. In R. Banerji, editor, *Formal Techniques in Artificial Intelligence, a Sourcebook*, pages 321–367. North Holland, 1990.
70. T. C. Przymusinski. On the declarative semantics of stratified deductive databases. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 193–216. Morgan Kaufmann, 1988.
71. T. C. Przymusinski. Extended stable semantics for normal and disjunctive programs. In Warren and Szeredi, editors, *7th International Conference on Logic Programming*, pages 459–477. MIT Press, 1990.
72. T. C. Przymusinski. Stationary semantics for disjunctive logic programs and deductive databases. In Debray and Hermenegildo, editors, *North American Conference on Logic Programming*, pages 40–57. MIT Press, 1990.
73. T. C. Przymusinski. Autoepistemic logic of closed beliefs and logic programming. In A. Nerode, W. Marek, and V. S. Subrahmanian, editors, *Logic Programming and Non-monotonic Reasoning*, pages 3–20. MIT Press, 1991.
74. T. C. Przymusinski. Static semantics for normal and disjunctive logic programs. *Annals of Mathematics and Artificial Intelligence*, 14:323–357, 1995.
75. R. Reiter. A logic for default reasoning. *Artificial Intelligence*, 13:81–132, 1980.
76. N. Rescher and R. Brandom. *The logic of inconsistency*. Basil Blackwell, 1980.
77. N. S and F. Rossi. Reasoning in inconsistent databases. In *Proc. of the 1990 Norht American Conf. on Logic Programming*, pages 255–272. The MIT Press, 1990.
78. C. Sakama. Extended well–founded semantics for paraconsistent logic programs. In *Fifth Generation Computer Systems*, pages 592–599. ICOT, 1992.
79. C. Sakama. *Studies on Disjunctive Logic Programming*. PhD thesis, Faculty of Engineering of Kyoto University, July 1994.
80. C. Sakama and K. Inoue. Paraconsistent Stable Semantics for Extended Disjunctive Programs. *Journal of Logic and Computation*, 5(3):265–285, 1995.
81. M. Schaerf. Notes on ginsberg's multivalued logics. *Computational Intelligence*, 7:154–159, 1991.
82. J. S. Schlipf. Complexity and undecidability results for logic programming. *Annals of Mathematics and Artificial Intelligence*, 15(3,4):257–288, 1995.
83. J. Shepherdson. Negation in logic programming for general logic programs. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 19–88. Morgan Kaufmann, 1988.
84. V. S. Subrahmanian. Paraconsistent disjunctive deductive databases. *Theoretical Computer Science*, 93:115–141, 1992.
85. F. Teusink. A proof procedure for extended logic programs. In *Proc. ILPS'93*. MIT Press, 1993.
86. S. Toulmin. *The uses of arguments*. Cambridge University Press, 1958.
87. A. Urquhart. Many-valued logic. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic, Vol. III*,, pages 71–116. D. Reidel Publishing Company, 1986.
88. 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.
89. G. Wagner. Logic programming with strong negation and inexact predicates. *Journal of Logic and Computation*, 1(6):835–859, 1991.

90. G. Wagner. Reasoning with inconsistency in extended deductive databases. In L. M. Pereira and A. Nerode, editors, *2nd International Workshop on Logic Programming and Non-monotonic Reasoning*, pages 300–315. MIT Press, 1993.
91. G. Wagner. Vivid logic: Knowledge-based reasoning with two kinds of negation. LNAI 764, 1994.
92. M. Wallace. Tight, consistent, and computable completions for unrestricted logic programs. *Journal of Logic Programming*, 15:243–273, 1993.
93. J.-H. You, S. Ghosh, L.-Y. Yuan, and R. Goebel. An introspective framework for paraconsistent logic programs. In J. W. Lloyd, editor, *ILPS95*. The MIT Press, 1995.

Index

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