Abduction over 3-valued extended logic programs

Carlos Viegas Damásio, Luís Moniz Pereira*

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

1 Introduction

Default negation was introduced so that negative conclusions could be drawn from logic programs. A first attempt to characterize this specific form of Closed World Assumption (CWA) [25] was the completion semantics [9].

This form of negation is capable of dealing with incomplete information via its non-monotonicity. However, completion semantics is inconsistent for a large class of normal logic programs and introduces undesirable tautologies. A spate of semantics proposals were set forth to solve these problems, from the late eighties onwards, including the well-founded semantics (WFS) [13], which addresses semantically non-terminating top-down computations by assigning such computations the truth value of "false" or "undefined", thereby giving semantics to every program. The well-founded semantics deals only with normal programs and thus provides no mechanism for explicitly declaring the falsity of literals. This can be a serious limitation.

Logic programming semantics has now been extended with a new explicit form of negation, beside the other implicit (or default) negation typical of Logic Programming.

Different semantics for such extended programs with a ¬-negation have appeared (e.g. [4, 15, 20, 24, 29, 30]). The generalization for extended programs of the WFS defined in [20], *WFSX*, adopted as the base semantics in this paper, provides a representational expressivity that captures a wide variety of logical reasoning forms [6, 16, 22].

The two forms of negation, default and explicit, are not unrelated: the "coherence principle" stipulates that the latter entails the former. Of course, introducing explicit negation requires, in addition, dealing with contradiction. Default negation and the revising of such default assumptions in the face of contradiction (or integrity constraint violation) are both non-monotonic reasoning mechanisms available in logic programming.

One major contribution of this paper is that of tackling contradiction, with some generality, and in particular within our choice semantics of extended logic programs, *WFSX*. Although such issues can be appproached from a belief revision point of view, they can likewise, and equivalently, be thought of within

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

an abductive framework. Indeed, a significant result of this paper is to define such three-valued abductive framework, *directly* over three-valued extended logic programming semantics.

Like [28], we too allow for contradiction to appear and show moreover how it can be removed by freely changing the truth-values (among "true", "false", and "undefined") of a subset of pre-designated *revisable* literals: the abducibles. To do so, we start by introducing a paraconsistent version of *WFSX*, $WFSX_p$, that permits reasoning with contradictory programs. Consistency is regained by revising the truth value of the revisables that lend "support" to integrity constraint violating literals, which are detected in $WFSX_p$.

A notion of global minimal revision (improving on previous work in the literature) is defined as well, which establishes a closeness relation between the several abductive solutions, thereby defining minimality of abduction too. Forthwith, the changes required to achieve program revision can be enforced by introducing or deleting program rules, for the abducible literals only, without loss in generality.

Subsequently, we show how other abductive frameworks relate to our own (namely Generalised Stable Models (GSM) [18]). We also demonstrate our framework can be used to characterize various logic programming semantics, namely Stable Models [14], Stationary Models [24], Well Founded Semantics [13], Preferred extensions [11], Extended Partial Stable Models [1] and Answer Sets [15].

A final result of the paper is that of proving that GSM and the abductive framework defined here have the same expressive power.

2 Paraconsistent *WFSX*

In this section we recall the language of logic programs extended with explicit negation (or extended logic programs for short) and define the paraconsistent version of well founded semantics with explicit negation [20]. An extended program is a set of rules of the form $L_0 \leftarrow L_1, \ldots, L_m, not\ L_{m+1}, \ldots, not\ L_n$ $(0 \leq m \leq n)$ where each L_i is an objective literal. An objective literal is either an atom A or its explicit negation $\neg A$. In the sequel, we also use $\neg$ to denote complementary literal wrt. the explicit negation, so that $\neg\neg A = A$. The set of all objective literals of a program P is called the extended Herbrand base of P and denoted by $\mathcal{H}(P)$. The symbol *not* stands for negation by default. $not\ L$ is called a default literal. Literals are either objective or default literals. By $not\ \{a_1, \ldots, a_n, \ldots\}$ we mean $\{not\ a_1, \ldots, not\ a_n, \ldots\}$. An interpretation of an extended program P is denoted by $T \cup not\ F$, where T and F are subsets of $\mathcal{H}(P)$. Objective literals in T are said to be *true* in I, objective literals in F *false by default* in I, and in $\mathcal{H}(P) - I$ *undefined* in I.

WFSX follows from WFS for normal programs plus the coherence requirement relating the two forms of negation: "For any objective literal L, if $\neg L$ is entailed by the semantics then $not\ L$ is also entailed.

As mentioned in the introduction, we're also interested in detecting contradiction occurrences, in order to remove it. To remove a contradiction, as we shall see in detail in section 4, we need first to detect it and, more importantly, we need to know what are the "reasons" or "causes" that lead to contradiction, i.e. which assumptions made are responsible for contradiction by "supporting" it, meaning that if they were withdrawn then contradiction would be dispelled.

To determine such "causes" a paraconsistent *WFSX* (or $WFSX_p$ for short) is required. The main idea is to calculate, always in keeping with coherence, all consequences of the program, even those leading to contradiction, as well as those arising from contradiction. For examples see [3].

Here we do not argue that $WFSX_p$ is in fact a *"semantics"*. Instead we view $WFSX_p$ as a tool needed for detecting contradiction and its causes, when *WFSX* is not able to assign meaning to a program. This tool is then used as the basis for removing contradiction where possible. The semantics assigned to the original program is defined as the *WFSX* of the program obtained after removing contradiction.

Since the purpose of $WFSX_p$ is to detect contradiction on the basis of *WFSX*, every principle that enforces the truth-value of literals in *WFSX* must also be enforced in $WFSX_p$. Thus, coherence must be kept in $WFSX_p$ too.

We will define $WFSX_p$ in terms of alternating fixpoints of Gelfond–Lifschitz Γ–like operators [14, 15]. For more details and relationships see [1, 3]. In definition 1 every literal $\neg A$ is treated as a new literal, say $\neg\_A$, unrelated to A.

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

To impose the coherence requirement we introduce:

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

Below we use $\Gamma(S)$ to denote $\Gamma_P(S)$, and $\Gamma_s(S)$ to denote $\Gamma_{P_s}(S)$.
It can be easily proved [3] that:

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

The above suggests introducing a definition of $WFSX_p$ where the paraconsistent WFM (WFM_p) is given by the least fixpoint of $\Gamma\Gamma_s$:

Definition 4 Paraconsistent *WFSX*. Let P be an extended program whose least fixpoint of $\Gamma\Gamma_s$ is T. Then, the paraconsistent well-founded model of P is $WFM_p(P) = T \cup not\ (\mathcal{H} - \Gamma_s T)$.

In this definition note how the use of Γ_s imposes coherence: if $\neg L$ is true, i.e. it belongs to T, then in $\Gamma_s T$, via semi-normality, all rules for L are removed and, consequently, $L \notin \Gamma_s T$, i.e. L is false by default. If condition $T \subseteq \Gamma_s T$ is satisfied, thereby guaranteeing that L and $\neg L$ do not belong both to the WFM_P, then $WFSX_p$ and $WFSX$ coincide.

Other paraconsistent semantics exist in the literature, probably the most representative ones being [7] and [27]. The former is an extension of definite logic programs with literals annotated with truth-values from Belnap's four valued logic. We are able to transform any annotated program to an extended logic program with isomorphic semantics. The latter is defined for extended logic programs but the coherence principle is not used. In fact, this semantics can easily be captured by viewing every explicit negated literal as a new literal, and it corresponds to a paraconsistent version of the semantics proposed by Przymuskinski in [24]. For lack of space we cannot explore these relations in more detail.

3 Three-valued Abductive Framework

After having presented a paraconsistent version of $WFSX$ (able therefore to detect contradiction in programs), we are now ready to foster the outlook, adopted in the introduction, that if a program is contradictory its revision is in order. By revision of a contradictory program P we mean a non-contradictory program P' obtained from P without introducing new predicates. One question immediately arises: what changes are allowed to P in order to obtain P'?

As argued in our previous work [21, 23], for the sake of simplicity and without loss of generality, the modification should be based solely on those literals that depend on no other literals, and by introducing or removing rules for such basic literals only. This is the stance we take here. The reader acquainted with abduction [10, 11, 12, 18] in logic programs will immediately notice the similarities between this condition on basic literals and conditions obeyed by abducibles.

In fact, for the purpose of defining revision, we will define a three-valued abductive framework using $WFSX_p$ as the underlying semantics. As usual, we will divide the logic program into a pair, which we call scenario: one element contains the subprogram that cannot be changed; the other contains the subprogram that can be changed in a predefined way, namely by adding rules of a simple fixed form for the abducible literals only. The latter part is such that it contains rules only for literals that depend on no other.

Until now we've considered contradiction as detected by $WFSX_p$. In general, however, there can be other reasons to consider a scenario unacceptable, due to violation of integrity knowledge about the intended program semantics. To encompass these situations, we now extend our language to deal with general integrity constraints with constraint checking defined according to theoremhood view [26, 19].

Definition 5 Integrity Constraints. An integrity constraint of an extended logic program P has the normal form $L_1 \vee L_2 \vee \ldots \vee L_n \Leftarrow L_{n+1} \wedge L_{n+2} \wedge \ldots \wedge L_{n+m} (n + m \geq 0)$, where each L_i is a literal belonging to the language of P. To an empty consequent (head) we associate the symbol $\mathbf{f}$ and to an empty antecedent (body) the symbol $\mathbf{t}$. An integrity theory is a set of integrity constraints, standing for their conjunction.

An integrity constraint is satisfied by an interpretation I if for every literal $L_{n+1}, \ldots, L_{n+m} \in I$ then at least one of the literals $L_1, \ldots, L_n$ is in I. Otherwise, the constraint is violated by I.

Notice that the literals appearing in the constraints can be objective or default ones. Violation of constraints of the form $\mathbf{f} \Leftarrow L, \neg L$ for each objective literal L detect paraconsistency.

It is only necessary to consider, as our basic building blocks, constraints of the form $L \Leftarrow \mathbf{t}$ and $\mathbf{f} \Leftarrow L$, as the table below shows.

Condition	Integrity Theory	Condition	Integrity Theory
$L = \mathbf{f}$	$not\ L \Leftarrow \mathbf{t}$	$L \neq \mathbf{f}$	$\mathbf{f} \Leftarrow not\ L$
$L = \mathbf{u}$	$(\mathbf{f} \Leftarrow L) \wedge (\mathbf{f} \Leftarrow not\ L)$	$L \neq \mathbf{u}$	$L \vee not\ L \Leftarrow \mathbf{t}$
$L = \mathbf{t}$	$L \Leftarrow \mathbf{t}$	$L \neq \mathbf{t}$	$\mathbf{f} \Leftarrow L$

Now, a program is contradictory iff its *WFSX* is contradictory, or violates some integrity constraint. In our previous work, we've always considered separately the two cases of removing contradiction by changing the truth value of revisable (or abducible) literals either into "undefined" [5, 21], or into true [23]. Note that, in the absence of integrity constraints, contradiction can always be removed by changing the truth value of abducibles into undefined only. However, in order to apply our contradiction removal approach to model based diagnosis [23], for instance, we also felt the need for stronger revisions, where the truth value of literals can be changed into true.

In the present work, as mentioned above, we will allow revision by introducing or removing rules for abducibles, such that we can change their truth values from any one value to any other value, in a mixed way. By relying on the above more general integrity constraints, a mixture of literal revisions to "undefined" and to the default complement value can be obtained.

The truth-varying basic beliefs are represented by a set of objective literals (the abducibles) whose truth-values are not determined beforehand. The fixed part of an abductive framework is an arbitrary extended logic program, which derives additional knowledge from the abducibles, once their value is known.

Definition 6 Three-valued Abductive Framework. An abductive framework is a triple $\langle P, A, I \rangle$. The first component, P, is an extended logic program, the second one a set of objective literals $A \subseteq \mathcal{H}(P)$ such that $L \in A$ iff $\neg L \in A$, and the third a set of integrity constraints. Set A contains all abducible literals. Additionally, there are no rules in P for any of the abducibles.

Given a three-valued abductive framework, a scenario for it is established by introducing, for each abducible, a rule defining its truth value.

Definition 7 Scenario. A scenario of an abductive framework $\langle P, A, I\rangle$ is a tuple $\langle P, P_{Var}, A, I\rangle$. For each literal $L \in A$ the variable part, P_{Var}, contains just one of the rules $L \leftarrow \mathbf{t}$ or $L \leftarrow \mathbf{u}$ or $L \leftarrow \mathbf{f}$. Moreover, if $L \leftarrow \mathbf{t} \in P_{Var}$ then $\neg L \leftarrow \mathbf{f} \in P_{Var}$ (and if $\neg L \leftarrow \mathbf{t} \in P_{Var}$ then $L \leftarrow \mathbf{f} \in P_{Var}$). No other rules belong to P_{Var}.

By definiton, the symbols $\mathbf{t}$ and *not* $\mathbf{f}$ belong to all models. Neither $\mathbf{u}$ nor *not* $\mathbf{u}$ belong to any model, where $\neg\mathbf{u} \equiv \mathbf{u}$.

An abductive explanation of a goal G can be expressed as satisfaction of an integrity constraint of the form $G \Leftarrow$. The variable component of a scenario determines an interpretation, and vice-versa. The coherence requirement on abducibles is imposed by the additional proviso in definition 7 and also guarantees non-contradiction in P_{Var}. By convention, when writing the variable part we omit all rules of the form $L \leftarrow \mathbf{f}$, since the effect is the same. To each scenario we associate two interpretations.

Definition 8. Let $\Omega = \langle P, A, I\rangle$ denote an abductive framework and σ a scenario $\langle P, P_{Var}, A, I\rangle$ of Ω. We define $\mathcal{M}(\sigma)$ by $WFSX_p(P \cup P_{Var})$, and $\mathcal{B}(\sigma)$ by $\mathcal{M}(\sigma) \cap \mathcal{H}(P_{Var})$.

Definition 9 Abductive solution. An abductive solution is a scenario σ of an abductive framework such that all integrity constraints in σ are satisfied by $\mathcal{M}(\sigma)$. Otherwise, the scenario is said to be contradictory.

The set of literals that we are willing to change in order to restore consistency is given a priori. These literals are called the revisables. These must have not rules for them in the program part. We enforce consistency with $\mathbf{f} \Leftarrow L, \neg L$ for all L:

Definition 10 Semantics of contradictory programs. Let logic program P be contradictory and R be a set of revisables. The semantics of P is given by the abductive solutions of the abductive framework:

$$\langle P, R, \{\mathbf{f} \Leftarrow L, \neg L \;:\; L \in \mathcal{H}(P)\}\rangle$$

A simple program transformation can be performed on the original program making every literal ersatz "revisable", even those with rules:

Definition 11. Let P be an extended logic program and P_{REV} the program obtained by applying the following transformation:

- For each literal $L \in \mathcal{H}(P)$ the rule $L \leftarrow true(L)$ is in P_{REV};
- Replace by $H \leftarrow B_1, \ldots, B_n, not\ false(id)$ any rule $H \leftarrow B_1, \ldots, B_n$ in P, where id is a new constant which uniquely identifies the new rule.

To define the semantics of a contradictory program, when the set of revisables is not given, we simply use definition 10 with P_{REV} and the revisables being all $true(\_)$ and $false(\_)$ literals. The same technique, restricted to the relevant rules, can be used to make revisable a literal which has rules.

4 Revision

First there exists an initial scenario. When it is contradictory abducibles must change so as to "reach" a new non-contradictory scenario, if possible. However this change is not arbitrary: the abducibles should change in a minimal way wrt the initial scenario. This raises the issue how to define a notion of "closeness" between two scenarios. Given a closeness relation we can then define (minimal) revisions of a scenario. We shall only consider revisions of a single scenario, thereof possibly obtaining several new scenarios[2].

Our notion of closeness is an extension, to the three-valued case, of Winslett's difference between pairs of models of the Possible Model Approach (PMA) [31]. Difference between models is defined as the facts upon which a pair of models disagree. In the PMA the models which have minimal difference, i.e. minimal change in the sense of set-inclusion, are preferred. Because we are in a three-valued setting, a refined notion of difference is necessary. What is distinctive in our approach is that if a literal is not "undefined" then we prefer to revise it to "undefined", rather than revising it to the opposite truth-value; if it is "undefined" then we revise it, with equal preferance, either to true or false. This is mainly motivated by the truth and knowledge orders among literals[3]: moving from *true* to *false* or vice-versa, *in either ordering*, is always via *undefined*.

Definition 12 Difference between interpretations. Let $I_1 = T_1 \cup not\ F_1$ and $I_2 = T_2 \cup not\ F_2$ be two interpretations. Then,

$$Diff(I_1, I_2) = (F_2 - F_1) \times \{\mathbf{f}\} \cup (F_1 - F_2) \times \{\mathbf{u}\} \cup (T_1 - T_2) \times \{\mathbf{u}\} \cup (T_2 - T_1) \times \{\mathbf{t}\}$$

Literals labeled with $\mathbf{t}$ (resp. $\mathbf{f}$) are those that change their truth-value to $\mathbf{t}$ (resp. $\mathbf{f}$), when going from I_1 to I_2. Literals labeled with $\mathbf{u}$ are those that change their original truth-value either to $\mathbf{u}$ or to their complementary truth-value. Some of these sets might not be disjoint. For more details see [3].

Definition 13 Closeness relation. Let M, A and B be interpretations wrt. a finite language. We say that A is closer to M than B iff $Diff(M, A) \subset Diff(M, B)$.

The closeness relation is defined between interpretations, not between scenarios. When scenarios are considered, one of the following two interpretations could be used: either the model determined by the scenario, $\mathcal{M}(\sigma)$, or else its associated beliefs, $\mathcal{B}(\sigma)$. We assume that what changes are the agent's basic beliefs, and that other changes of belief are just consequences of basic belief changes. So, when facing contradiction, the agent should minimally change its basic beliefs, not necessarily what it extracts from those beliefs (i.e. their consequences).

[2] In order to iterate revisions we should also consider revisions of sets of scenarios. That is left to future work.

[3] In the truth ordering $\mathbf{f} < \mathbf{u} < \mathbf{t}$, and in the knowledge one $\mathbf{u} < \mathbf{f}$ and $\mathbf{u} < \mathbf{t}$.

Definition 14 Revision of a scenario. Let σ be the initial scenario of an abductive framework Ω_P. A revision of σ is $\tau \in \Sigma_P$ (the set of all abductive solutions) such that: $\forall_{\delta \in \Sigma_P}\ Diff(\mathcal{B}(\sigma), \mathcal{B}(\delta)) \subseteq Diff(\mathcal{B}(\sigma), \mathcal{B}(\tau)) \ \Rightarrow\ \delta = \tau$

Notice, again, that revision is defined here wrt a single initial scenario.

5 Comparisons

In the first subsection below, we make the bridge with our contradiction removal methods, which are captured by simple abductive frameworks.

The rest of the section provides a reconstruction of some of the most important logic programming semantics, by strongly restricting our abduction and revision frameworks: namely we will forego the use of explicit and default negation in the programs, and abductive solutions are *always* two-valued. In fact, we use default negation in our transformed programs just in order to improve readability. These default literals can simply be thought of as new objective literals. In any case, default literals will have no rules for them; furthermore, they are always abducible.

Our approach has close relationships with the recent work on argumentation theoretic semantics [11, 8]. Thus, all that we do here can equivalently be done with the Generalized Stable Models approach. At the end of our comparisons, and as a side result, we prove the equivalence between GSM and our three-valued framework.

Notice that we started off with $WFSX_p$, a less expressive semantics (if $P \neq NP$), and that by adding abduction we obtain the same expressive power as an abductive framework built on top of a more expressive semantics (Stable Models). This is an advantage, because more efficient top-down proof procedures are defined for $WFSX$ [2].

Not using explicit negation in the abductive framework transformations needs some care. When we define our abductive frameworks, we will omit the condition "if L is an abducible then ¬L is also abducible." For the sake of simplicity, we assume we make all these explicitly negated literals abducibles. Because we are deliberately excluding the use of explicit negation, we enforce that in all abductive solutions $not\ \neg L$ must be true. This can be simply engendered by adding the integrity constraints $not\ \neg L \Leftarrow$ to the integrity theory, for all positive literals L in the language of the program.

5.1 Contradiction Removal Methods

Here we show how our three-valued (CRS) [21, 1] and two-valued contradiction [23] revision methods can be straightforwardly obtained from our abductive and revision frameworks [4].

[4] In [21, 1, 23] what we dubbed revisables had the conventional form "*not* r"; here we use their default complement "r" instead.

In *CRS* revisable literals are all assumed false by default at the start. Integrity constraints are restricted to denials of the form $\mathbf{f} \Leftarrow L_1, \ldots, L_n$. In order to satisfy the integrity constraints, the revisable literals are allowed to change to the undefined value. Changes of revisables' truth-value to *true* are interdicted. The intended transformation is immediate, and the only restriction to enforce is that of discarding those abductive solutions which assign true to a revisable.

Theorem 15. *Let P be a logic program with revisables R and integrity constraints I. The abductive framework $\Omega_{CRS} = \langle P, R, I_{CRS} \rangle$, where $I_{CRS} = I \cup \{false \Leftarrow r : r \in R\}$. Then, σ_{CRS} is an abductive solution of Ω_{CRS} iff the pair $\langle \mathcal{M}(\sigma_{CRS}), \mathcal{B}(\sigma_{CRS}) \rangle$ is a non-contradictory submodel of P with I and revisables R.*

In our two-valued contradiction removal methods, the above restrictions to the integrity theory also apply, but now revisables are not allowed to change to undefined. If we need to revise an initially false assumption, then we can change its value only to true:

Theorem 16. *Let P be a logic program with revisables R and integrity constraints I in the sense of [23]. We define the abductive framework $\Omega_{2REV} = \langle P, R, I_{2REV} \rangle$, where $I_{2REV} = I \cup \{r \vee not\ r \Leftarrow \mathbf{t} : r \in R\}$. Then, σ_{2REV} is a solution of Ω_{2REV} iff $\{r \mid r \in \mathcal{B}(\sigma_{2REV})\}$ is a revision of P wrt I and R. The abductive solutions obtained by revision of the initial scenario with variable part $\{\}$ are the minimal revisions of P.*

5.2 Two-valued semantics

The manner of capturing Stable Model semantics through abduction is well known from the literature [12, 17] and here we follow their approach. Also, the authors of Generalised Stable Models [18] present a transformation from an abductive framework with negation as failure in rules of programs into a pure abduction problem. In this subsection we simply recapture those transformations in our framework, showing that our setting is as powerful as GSM. The equivalence between our abductive framework and Kakas and Mancarella's one is discussed in a later next section.

Theorem 17. *Let P be a normal logic program. The abductive framework Ω_{SM} is $\langle P_{SM}, A_{SM}, I_{SM} \rangle$, where*

$$P_{SM} = \{H \leftarrow B_1, \ldots, B_n, C_1^*, \ldots, C_m^* \mid H \leftarrow B_1, \ldots, B_n, not\ C_1, \ldots, not\ C_m \in P\}$$
$$A_{SM} = \{L^* \mid L \in \mathcal{H}(P)\} \text{ and } I_{SM} = \{L \vee L^* \Leftarrow \mathbf{t}, \quad not\ L \vee not\ L^* \Leftarrow \mathbf{t} \mid L \in \mathcal{H}(P)\}$$

Then, σ_{SM} is a solution of Ω_{SM} iff $\{L \mid L^* \notin \mathcal{B}(\sigma_{SM})\}$ is a stable model of P.

The way of capturing *GSM* is by an extension of the above transformation as defined in [18]:

Theorem 18. *Let $\langle P, A, I\rangle$ be an abductive framework in the sense of [18]. The abductive framework $\Omega_{GSM} = \langle P_{GSM}, A_{GSM}, I_{GSM}\rangle$ is:*

$$P_{GSM}{=}P_{SM} \qquad A_{GSM} = A_{SM} \cup A \cup \{a^* \mid a \in A\}$$
$$I_{GSM} = \{L \vee L^* \Leftarrow \mathbf{t}, not\ L \vee not\ L^* \Leftarrow \mathbf{t} \mid L^* \in A_{GSM}\} \cup I^*$$

where I^* is obtained from I by replacing every default literal not L by L^*. Then, σ_{GSM} is a solution of Ω_{SM} iff $M(\Delta) = \{L \mid L \notin A \wedge L^* \notin \mathcal{B}(\sigma_{SM})\}$, where $\Delta = \{a \in A \mid a \in \mathcal{B}(\sigma_{GSM})\}$, is a GSM of $\langle P, A, I\rangle$.

5.3 Three-valued semantics

Even with the strong restrictions to the abductive framework mentioned in the introduction of this section, we can still characterize how stationary (or, equivalently, partial stable) models semantics (*PSM*) [24] are obtained. Likewise, we also show how to obtain the WFM and Dung's preferred extensions [11], via revision.

What we intend to capture is the fixpoints of $T = \Gamma\Gamma(T)$ which obey the condition $T \subseteq \Gamma(T)$. Thus, we need a way of computing $\Gamma\Gamma(T)$ and $\Gamma(T)$. We achieve this by a program transformation and additional integrity constraints. Part of these integrity constraints enforce the inclusion condition above, and the others the fixpoint condition. We start with the program transformation:

Definition 19. Let P be a normal logic program. Program P_{T-TU} is obtained as follows. Consider a rule $H \leftarrow B_1, \ldots, B_n, not\ C_1, \ldots, not\ C_m$ of P. Then the following two rules belong to P_{T-TU}:

- $H^T \leftarrow B_1^T, \ldots, B_n^T, false(C_1), \ldots, false(C_m)$
- $H^{TU} \leftarrow B_1^{TU}, \ldots, B_n^{TU}, not\ true(C_1), \ldots, not\ true(C_m)$

The first kind of rule, labeled with superscript T, will be used to derive what is true in the program, i.e. it computes the outer Γ of $\Gamma\Gamma(T)$. The rules having TU superscripted literals determine what is true or undefined, i.e. it corresponds to the inner Γ of the alternating fixpoint definition of WFS.

As noticed before, the use of default literal "$not\ true(L)$" in the above rules can be substituted by, for instance, "$false\_or\_und(L)$." By removing all occurrences of default literals from the original program, cycles through negation by default disappear. The burden of default negation reasoning is taken on by abduction via integrity constraints. Now we introduce the integrity constraints used to guarantee that obtaining the *PSMs* of WFS:

Definition 20. To each literal L in the language of P we associate the following integrity theory I_L^{T-TU}:

$not\ L^{TU} \vee not\ false(L) \Leftarrow$ (1)	$not\ L^T \vee L^{TU} \Leftarrow$ (3)	$not\ L^T \vee true(L) \Leftarrow$ (4)
$L^{TU} \vee false(L) \Leftarrow$ (2)		$L^T \vee not\ true(L) \Leftarrow$ (5)

Integrity constraints (1) and (2) above link the computation of the inner Γ operator to the outer one. The abductive solutions satisfying (1) and (2) should have either one of two models: $\{L^{TU}, not\ false(L)\}$ or $\{not\ L^{TU}, false(L)\}$. Thus, if L is true or undefined iff it is not false.

Constraints (4) and (5) ensure the fixpoint condition. Any abductive solution conforming to these two ICs has either one of the two models: $\{L^T, true(L)\}$ or $\{not\ L^T, not\ true(L)\}$. Therefore L can be assumed true iff it is derived by the program. Finally, integrity constraint (3) enforces the condition $T \subseteq \Gamma(T)$. As a whole, the integrity constraints are satisfied by an abductive solution obeying one of the three following models:

$$\begin{array}{ll}\{not\ false(L), not\ true(L), not\ L^T, L^{TU}\} & (M1)\\ \{false(L), not\ true(L), not\ L^T, not\ L^{TU}\} & (M2)\\ \{not\ false(L), true(L), L^T, L^{TU}\} & (M3)\end{array}$$

Models M1, M2, and M3, correspond to the situation where literal L is, respectively, undefined, false, or true.

Now we can immediately define the abductive framework which captures all *PSMs* of an arbitrary normal logic program.

Theorem 21. *Let P be a normal logic program and Ω_{PSM} be the framework $\left\langle P_{T-TU}, \{false(L), true(L) \mid L \in \mathcal{H}(P)\}, \bigcup_{L\in\mathcal{H}(P)} I_L^{T-TU}\right\rangle$. Then σ_{PSM} is a solution of Ω_{PSM} iff $\{L \mid true(L) \in \mathcal{B}(\sigma_{PSM})\} \cup \{not\ L \mid false(L) \in \mathcal{B}(\sigma_{PSM})\}$ is a Partial Stable Model of P.*

Enforcing an additional constraint $false(L) \vee true(L)$ on each literal, we obtain again the stable models. This was to be expected, because stable models are two-valued partial stable models. The previous constraint simply discards model $(M1)$. The WFM is the F-least PSM of the program. This minimality condition can be captured by revision:

Theorem 22. *Let P be a normal logic program. Then, there is a unique revision of Ω_{PSM} wrt. the initial scenario σ with $\mathcal{B}(\sigma) = \{\}$. Furthermore, this revision determines the WFM of P as in theorem 21.*

In the other way around, Dung's preferred extensions are F-maximal *PSMs* of the program. These extensions can also be captured through revision:

Theorem 23. *Let P be a normal logic program. Then the revisions of Ω_{PSM} wrt. the initial scenario σ with $\mathcal{B}(\sigma) = \{false(L), true(L) \mid L \in \mathcal{H}(P)\}$ are in 1:1 correspondence with preferred extensions. Furthermore, they can be determined as in theorem 21.*

5.4 Semantics with Explicit Negation

Finally, we discuss how the $PSMs$ of *WFSX* can be obtained. As a side effect we show how answer set semantics is also embedded. Afterwards, we go on to determine the paraconsistent WFM, and end this subsection by presenting the restricted abductive formulation of our three-valued framework. As a corollary, we prove that GSM has the same expressive power as our three-valued abductive framework.

Definition 24. Let P be an extended logic program. Program $P^{\neg}_{T-TU}$ is obtained as follows. Every explicitly negated literal $\neg L$ is replaced by a new atom $'\neg\_L'$ in the transformed program. Every rule of program P of the form $H \leftarrow B_1, \ldots, B_n, not\ C_1, \ldots, not\ C_m$ is replaced by the two rules:

- $H^T \leftarrow B_1^T, \ldots, B_n^T, false(C_1), \ldots, false(C_m)$
- $H^{TU} \leftarrow B_1^{TU}, \ldots, B_n^{TU}, not\ true(C_1), \ldots, not\ true(C_m), not\ true(N_H)$

Where N_H is L^T if $H = \neg\_L$, and $\neg\_H^T$ otherwise.

So we've replaced the inner Γ operator by the Γ_s one. For ease of presentation, we assume that the transformation between $\neg L$ in the language of the program and $\neg\_L$ in the language of $P^{\neg}_{T-TU}$ is implicitly applied in the remainder of the paper. For instance, in the theorem below, when we say "$\{L \in \mathcal{H}(P)\}$" we mean "$\{\neg\_L \mid \neg L \in \mathcal{H}(P)\} \cup \{L \mid \text{L is a positive literal of } \mathcal{H}(P)\}$."

Theorem 25. *Let P be a normal logic program and Ω_{xpsm} be the framework $\left\langle P^{\neg}_{T-TU}, \{false(L), true(L) \mid L \in \mathcal{H}(P)\}, \bigcup_{L\in\mathcal{H}(P)} I_L^{T-TU} \right\rangle$. Then, σ_{xpsm} is a solution of Ω_{xpsm} iff $\{L \mid true(L) \in \mathcal{B}(\sigma_{xpsm})\} \cup \{not\ L \mid false(L) \in \mathcal{B}(\sigma_{xpsm})\}$ is a Partial Stable Model of P under* WFSX.

Answer Set Semantics [15] is obtained by adding the additional constraint $false(L) \vee true(L) \Leftarrow$ for every positive objective literal in $\mathcal{H}(P^{\neg}_{T-TU})$. As for *WFS*, the WFM of *WFSX* can be obtained through revision too:

Theorem 26. *Let P be an extended logic program. Then there is a unique revision of Ω_{XPSM} wrt. the initial scenario σ with $\mathcal{B}(\sigma) = \{\}$. Furthermore, this revision determines the WFM under* WFSX *of P, as per theorem 25.*

The paraconsistent WFM can also be obtained by revision of an abductive framework Ω_{WFMp}. This framework is gotten from Ω_{XPSM} by removing all third type integrity constraints from I_{XPSM}: these are the ones which ensure $T \subseteq \Gamma(T)$.

Theorem 27. *Let P be an extended logic program. Then there is a unique revision of Ω_{WFMp} wrt. the initial scenario σ with $\mathcal{B}(\sigma) = \{\}$. Furthermore, this revision determines the $WFMp$, as per theorem 25.*

Finally we introduce our last program transformation, describing our general framework abductive solutions in terms of the more restricted one.

Definition 28. Let $\langle P, A, I \rangle$ be a three-valued abductive framework as in definition 6. Let Ω_{ABD} be defined as follows:

$$\begin{aligned}
P_{ABD} &= P^{\neg}_{T-TU} \cup \left\{L^T \leftarrow true(L), L^{TU} \leftarrow true(L), L^{TU} \leftarrow und(L) \mid L \in A\right\} \\
A_{ABD} &= \{false(L), true(L) \mid L \in \mathcal{H}(P)\} \cup \{und(L) \mid L \in A\} \\
I_{ABD} &= \{not\ true(L) \vee not\ und(L) \Leftarrow,\ false(L) \Leftarrow true(\neg_L) \mid L \in A\} \cup \\
&\quad\ I_{WFMp} \cup I_I
\end{aligned}$$

where I_I is obtained from I by replacing every objective literal L in an integrity constraint by $true(L)$ and every default literal $not\ L$ by $false(L)$.

Our main theorem follows:

Theorem 29. *Let* $\Omega = \langle P, A, I \rangle$ *be a three-valued abductive framework. Then* α *is a revision of* Ω_{ABD} *wrt to the initial empty scenario iff* σ*, with* $\mathcal{M}(\sigma) = \{L \mid true(L) \in \mathcal{B}(\alpha)\} \cup \{not\ L \mid false(L) \in \mathcal{B}(\alpha)\}$*, is an abductive solution of* Ω*.*

The introduction of abducibles $und(\_)$ is only necessary to guarantee that all abductive solutions are incomparable. The minimality condition on the revisions has the purpose of finding the least fixpoints of $\Gamma\Gamma_s$, for each possible combination of values of abducibles, as per theorem 27.

Corollary 30. *The three-valued abductive framework has the same expressive power as GSM.*

Thus we find our abductive solutions without relying on a three-valued semantics. All is defined in terms of a two-valued abductive framework. Thus, if we translate our transformation into the *GSM* setting, the generalised stable models of the transformed abductive theory are in a 1:1 correspondence with our abductive solutions, when a minimality condition is enforced. This, in conjunction with the results of subsection 5.2, prove the equivalence of both abductive frameworks.

References

1. J. J. Alferes. *Semantics of Logic Programs with Explicit Negation.* PhD thesis, Universidade Nova de Lisboa, October 1993.
2. J. J. Alferes, C. V. Damásio, and L. M. Pereira. A top-down derivation procedure for programs with explicit negation. In M. Bruynooghe, editor, *ILPS'94*, pages 424–438. MIT Press, 1994.
3. J. J. Alferes, C. V. Damásio, and L. M. Pereira. A logic programming system for non-monotonic reasoning. *Special Issue of the JAR*, 1995. To appear.
4. J. J. Alferes and L. M. Pereira. On logic program semantics with two kinds of negation. In K. Apt, editor, *IJCSLP'92*, pages 574–588. MIT Press, 1992.
5. J. J. Alferes and L. M. Pereira. Contradiction: when avoidance equal removal. In R. Dyckhoff, editor, *4th Int. Ws. on Extensions of LP*, volume 798 of *LNAI*. Springer-Verlag, 1994.
6. C. Baral and M. Gelfond. Logic programming and knowledge representation. *J. Logic Programming*, 19/20:73–148, 1994.
7. H. A. Blair and V. S. Subrahmanian. Paraconsistent logic programming. *Theoretical Computer Science*, 68:135–154, 1989.
8. A. Bondarenko, F. Toni, and R. Kowalski. An assumption–based framework for nonmonotonic reasoning. In L. M. Pereira and A. Nerode, editors, *2nd Int. Ws. on LP & NMR*, pages 171–189. MIT Press, 1993.
9. K. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum Press, 1978.

10. M. Denecker and D. D. Schreye. SLDNFA: an abductive procedure for normal abductive programs. In K. Apt, editor, *IJCSLP'92*, pages 686–700. MIT Press.
11. P. M. Dung. Negation as hypotheses: An abductive framework for logic programming. In K. Furukawa, editor, *8th Int. Conf. on LP*, pages 3–17. MIT Press, 1991.
12. K. Eshghi and R. Kowalski. Abduction compared with negation by failure. In *6th Int. Conf. on LP*. MIT Press, 1989.
13. 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.
14. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R. Kowalski and K. A. Bowen, editors, *ICLP'88*, pages 1070–1080. MIT Press.
15. M. Gelfond and V. Lifschitz. Logic programs with classical negation. In Warren and Szeredi, editors, *7th Int. Conf. on LP*, pages 579–597. MIT Press, 1990.
16. M. Gelfond and V. Lifschitz. Representing actions in extended logic programs. In K. Apt, editor, *IJCSLP'92*, pages 559–573. MIT Press, 1992.
17. A. Kakas, R. Kowalski, and F. Toni. Abductive logic programming. *Journal of Logic and Computation*, 2:719–770, 1993.
18. A. C. Kakas and P. Mancarella. Generalized stable models: A semantics for abduction. In *Proc. ECAI'90*, pages 401–405, 1990.
19. J. Lloyd and R. Topor. A basis for deductive database systems. *Journal of Logic Programming*, 2:93–109, 1985.
20. L. M. Pereira and J. J. Alferes. Well founded semantics for logic programs with explicit negation. In B. Neumann, editor, *ECAI'92*, pages 102–106. Wiley, 1992.
21. 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, *LP & NMR*, pages 105–119. MIT Press, 1991.
22. L. M. Pereira, J. N. Aparício, and J. J. Alferes. Non–monotonic reasoning with logic programming. *Journal of Logic Programming. Special issue on Nonmonotonic reasoning*, 17(2, 3 & 4):227–263, 1993.
23. L. M. Pereira, C. Damásio, and J. J. Alferes. Diagnosis and debugging as contradiction removal. In L. M. Pereira and A. Nerode, editors, *2nd Int. Ws. on LP & NMR*, pages 316–330. MIT Press, 1993.
24. T. Przymusinski. Extended stable semantics for normal and disjunctive programs. In Warren and Szeredi, editors, *ICLP'90*, pages 459–477. MIT Press, 1990.
25. R. Reiter. On closed–world data bases. In H. Gallaire and J. Minker, editors, *Logic and DataBases*, pages 55–76. Plenum Press, 1978.
26. R. Reiter. Towards a logical reconstruction of relational database theory. In M. Brodie and J. Mylopoulos, editors, *On Conceptual Modelling*, pages 191–233. Springer–Verlag, 1984.
27. C. Sakama. Extended well–founded semantics for paraconsistent logic programs. In *Fifth Generation Computer Systems*, pages 592–599. ICOT, 1992.
28. F. Teusink. A proof procedure for extended logic programs. In *Proc. ILPS'93*. MIT Press, 1993.
29. G. Wagner. Logic programming with strong negation and innexact predicates. *J. of Logic and Computation*, 1(6):835–861, 1991.
30. G. Wagner. Neutralization and preeemption in extended logic programs. Technical report, Freien Universitat Berlin, 1993.
31. M. Winslett. Reasoning about action using a possible model approach. In *7th AAAI*, pages 89–93, 1988.