A rewriting method for Well-Founded Semantics with Explicit Negation.

Pedro Cabalar

Dept. of Computer Science,
University of Corunna, Spain
cabalar@dc.fi.udc.es

Abstract. We present a modification of Brass et al's transformation-based method for the bottom-up computation of well-founded semantics (WFS), in order to cope with explicit negation, in the sense of Alferes and Pereira's WFSX semantics. This variation consists in the simple addition of two intuitive transformations that guarantee the satisfaction of the so-called *coherence principle*: whenever an objective literal is founded, its explicit negation must be unfounded. The main contribution is the proof of soundness and completeness of the resulting method with respect to WFSX. Additionally, by a direct inspection on the method, we immediately obtain results that help to clarify the comparison between WFSX and regular WFS when dealing with explicit negation.

1 Introduction

Logic Programming (LP) has become one of the most popular tools for practical nonmonotonic reasoning, thanks to the use of declarative semantics, particularly, *stable models* [7] and *well-founded semantics* (WFS) [13]. This success is probably due to two important facts: (1) the availability of efficient algorithms and implementations for computing these semantics; and (2), the evolution of the basic LP paradigm to allow a more flexible knowledge representation. Of course, the progress in these two directions has not been simultaneous: we typically face new LP extensions for which the current implementations are not applicable.

For efficiency purposes, the improvement of inference methods for WFS has become interesting, not only when using WFS as a basic semantics, but also for stable models checkers, since they usually involve intermediate computations of the well-founded model. This interest motivated the research line followed by Brass et al. [3], who developed a method that improves the efficiency of the original *alternated fixpoint procedure* introduced by van Gelder [12]. Their method relies on the successive application of simple program transformations, until an equivalent non-reducible program (the so-called *program remainder*) is obtained.

Brass et al's method, however, was not thought for Extended Logic Programming (ELP), i.e., logic programs with explicit negation. The introduction of ELP comes from the need for distinguishing between *default negation*, '$not\, p$' (that is, we fail to prove p) and *explicit negation* ' $\overline{p}$ ' (that is, we assert p

to be explicitly false). In the case of stable models, the required modification is straightforward: we just rule out stable models containing any pair $\{p, \overline{p}\}$ (where $\overline{p}$ is simply treated as a regular atom). These stable models receive the name of *answer sets* [8]. Unfortunately, in the case of WFS, this contradiction rejection is not enough, as observed by Alferes and Pereira [9, 1]. More concretely, one would expect that whenever an atom p is explicitly false, $\overline{p}$, its default negation, *not* p, should hold (*coherence principle*), something not guaranteed in the resulting well-founded model, which may leave *not* p undefined. To incorporate the coherence principle into WFS, Alferes and Pereira proposed a variation called WFSX (*well-founded semantics with explicit negation*).

In this paper we study how to update the program remainder method to compute WFSX. We will show that the update is, in fact, quite simple and natural, consisting in the addition of two intuitive transformations to the already existing ones for computing WFS. As a result, we prove that WFS obtains less or equal information (i.e., defined atoms) than WFSX. The modified method becomes especially interesting for the use of WFSX as an underlying semantics, but it could also be applied to improve the computation of answer sets, since it obtains more information in each intermediate computation of the well-founded model.

The paper is organized as follows. The next section contains a brief review of basic LP definitions and WFS. Sections 3 and 4 respectively describe Brass et al's transformation method and Alferes and Pereira's WFSX. Section 5 presents the proposed variation of Brass et al's method, explaining the main results. Finally, Section 6 concludes the paper. Proofs for theorems in Section 5 have been collected in Appendix A.

2 Basic definitions

The syntax of logic programs is defined starting from a (possibly infinite) set of ground atoms $\mathcal{H}$ called the *Herbrand base*. We assume that all the variables have been previously replaced by their possible ground instances. We will use lower-case letters $a, b, c, \dots, p, q, \dots$ to denote atoms from $\mathcal{H}$. A *program literal* is either an atom $a \in \mathcal{H}$ or its default negation *not* a (which is called a *default literal*). A *normal logic program* is a (possibly infinite) set of *rules* of shape:

$$H \leftarrow B_1, \dots, B_n \tag{1}$$

where $n \geq 0$, H is an atom and the $B_i's$ are program literals. Given a rule r like (1), we respectively define its *head* and *body* as $head(r) = H$ and $body(r) = \{B_1, \dots, B_n\}$. When $n = 0$, we usually write 'H' to stand for '$H \leftarrow$', and say that H is a program *fact*. For any normal logic program P, the sets $facts(P)$ and $heads(P)$ respectively contain the program facts and the heads of all the rules in P. Consequently, $facts(P) \subseteq heads(P)$. A normal logic program is said to be *positive* (or *definite*) when it does not contain any default literal. Another interesting type of logic programs are those not containing any kind of cyclic dependence. Program P is said to be *hierarchical* (or *acyclic*) when all its atoms can be arranged in levels, i.e., there exists an integer mapping $\lambda : \mathcal{H} \rightarrow \mathcal{Z}$ satisfying $\lambda(head(r)) < \lambda(b_i)$, for any rule r and each atom b_i occurring (possibly negated) in $body(r)$.

A (2-valued) *interpretation* M is defined as any subset of $\mathcal{H}$, $M \subseteq \mathcal{H}$. It can also be seen as a function $M : \mathcal{H} \rightarrow \{\mathtt{t}, \mathtt{f}\}$ mapping a truth value for each atom in $\mathcal{H}$, so that $M(a) = \mathtt{t}$ iff $a \in M$. The interpretation can be extended to provide a valuation for any formula ϕ, so that $M(\phi)$ follows the standard propositional definitions, where '*not*', ',' and '$\leftarrow$' represent classical negation, conjunction and material implication, respectively. When $M(\phi) = \mathtt{t}$ we say that M *satisfies* ϕ. An interpretation is said to be a *model* of a program P iff it satisfies all its rules.

As shown in [11], any positive logic program P has a *least model* (with respect to set inclusion), which corresponds to the least fixpoint of the monotonic operator T_P:

$$T_P(M) = \{c \mid (c \leftarrow a_1, \ldots, a_n) \in P, \text{ and } a_i \in M \text{ for all } i \in [1, n]\}$$

As T_P is monotonic, the Knaster and Tarski's theorem [10] applies, and the least fixpoint is computable by iteration of T_P, starting from the smallest interpretation $\emptyset$.

We define the *reduct* of a normal logic program P with respect to an interpretation M, written P^M, as:

$$P^M = \{(c \leftarrow a_1, \ldots, a_n) \mid (c \leftarrow a_1, \ldots, a_n, not\, b_1, \ldots, not\, b_m) \in P, \\ \text{and } b_i \notin M \text{ for all } i \in [1, m]\}$$

Since P^M is a positive program, it has a least model we will denote as $\Gamma_P(M)$, or simply $\Gamma(M)$ when there is no ambiguity. The fixpoints of Γ receive the name of *stable models*.

The well-founded model will have the shape of a 3-valued or *partial interpretation* M, formally defined as a pair (M^+, M^-) of disjoint sets of atoms. The sets M^+, M^- and $\mathcal{H} - (M^+ \cup M^-)$ represent the *true*, *false* and *undefined* atoms, respectively. Given M^-, we will usually refer to its complementary set $\mathcal{H} - M^-$, that is, the nonnegative atoms. It is clear that, since M^+ and M^- are disjoint, $M^+ \subseteq \mathcal{H} - M^-$. We say that M is *complete* iff $M^+ = \mathcal{H} - M^-$, i.e., $\mathcal{H} = M^+ \cup M^-$. Besides, two partial interpretations can be compared with respect to their amount of information (i.e., defined atoms):

Definition 1. (Information or Fitting's ordering, $\leq_F$)
We say that interpretation $M_1 = (M_1^+, M_1^-)$ contains less information *than interpretation $M_2 = (M_2^+, M_2^-)$, denoted as $M_1 \leq_F M_2$, iff $M_1^+ \subseteq M_2^+$ and $M_1^- \subseteq M_2^-$.*

The characterization of WFS relies on the fact that the Γ operator is antimonotonic, and so, Γ^2, that is, Γ applied twice, results to be monotonic. In this way, we may apply again Knaster-Tarski's theorem for Γ^2: there exists a least (resp. greatest) fixpoint, $lfp(\Gamma^2)$ (resp. $gfp(\Gamma^2)$), which is computable by iteration on the least set of atoms $\emptyset$ (resp. the greatest set of atoms $\mathcal{H}$).

Moreover, each fixpoint of this pair can be computed in terms of the other: $gfp(\Gamma^2) = \Gamma(lfp(\Gamma^2))$ and $lfp(\Gamma^2) = \Gamma(gfp(\Gamma^2))$.

Definition 2. (Well-founded model (WFM))
The well-founded model *(WFM) of a normal logic program P corresponds to the 3-valued interpretation: $\big(lfp(\Gamma^2), \mathcal{H} - gfp(\Gamma^2)\big)$*

The relation to stable models is clarified by the following well-known properties:

Proposition 1. *Let W be the WFM of a normal logic program P. For any stable model M of P, we have that $W \leq_F (M, \mathcal{H} - M)$.*

Proposition 2. *Let $W = (W^+, W^-)$ be the WFM of a normal logic program P. If W is complete, i.e. $W^+ = \mathcal{H} - W^-$, then W^+ is the unique stable model of P.*

3 Brass et al's method

Brass et al's method can be introduced by initially identifying what we will call the *trivial interpretation* of a normal logic program.

Definition 3. (Trivial interpretation)
The trivial interpretation *of a normal logic program P is the 3-valued interpretation: $\big(facts(P), \mathcal{H} - heads(P)\big)$.*

That is, the trivial interpretation makes true all the atoms that occur as facts in P and makes false all the atoms that are not head of any rule in P. The method is mainly based on successively simplifying rules that deal with atoms with a defined truth value in the trivial interpretation. If P is the current program, then we define the transformations:

1. *Positive reduction* $\overset{\mathtt{P}}{\longmapsto}$: for any rule r, delete any literal $(not\ p) \in body(r)$ such that $p \notin heads(P)$.
2. *Negative reduction* $\overset{\mathtt{N}}{\longmapsto}$: delete any rule r containing some $(not\ p) \in body(r)$ with $p \in facts(P)$.
3. *Success* $\overset{\mathtt{S}}{\longmapsto}$: for any rule r, delete any literal $p \in body(r)$ such that $p \in facts(P)$.
4. *Failure* $\overset{\mathtt{F}}{\longmapsto}$: delete any rule r containing some $p \in body(r)$ with $p \notin heads(P)$.
5. *Positive loop detection* $\overset{\mathtt{L}}{\longmapsto}$: delete any rule r containing some $p \in body(r)$ such that $p \notin \Gamma(\emptyset)$.

Proposition 3. (see theorem 4.17 in [3])
The transformations $\{\mathtt{P},\mathtt{N},\mathtt{S},\mathtt{F},\mathtt{L}\}$ are sound w.r.t. WFS and provide a confluent calculus which is strongly terminating. Furthermore, let P be the final program where no new transformation is applicable. This program is called the program remainder*. Then, the WFM of P corresponds to its trivial interpretation $(facts(P), \mathcal{H} - heads(P))$.*

It is easy to see that the first four transformations just simplify rules dealing with atoms that are known in the trivial interpretation (their truth value is not undefined). As an interesting additional result, (theorem 4.9 in [3]) the exhaustive application of these four rules allows obtaining the *Fitting's model* [6] of a normal logic program. In this way, the fifth transformation, `L`, can be seen as the real "contribution" of WFS with respect to Fitting's semantics. As it is well known, Fitting's model may yield undefined atoms because of self-supported positive cycles. For instance, the Fitting's model for the simple program $\{p \leftarrow p\}$ would leave p undefined instead of false. In order to avoid this behavior, transformation `L` adopts an optimistic point of view: we compute the consequences of the program assuming all the default literals to be true (this is the real meaning of $\Gamma(\emptyset)$). Atoms that are not obtained by this procedure will always be false (since we had assumed the most optimistic case for default negation).

Let us see a simple example. Consider the program P_1:

$$\begin{array}{lll} a \leftarrow \mathit{not}\, b, c & d \leftarrow \mathit{not}\, g, e & g \leftarrow \mathit{not}\, c \\ b \leftarrow \mathit{not}\, a & e \leftarrow \mathit{not}\, g, d & h \leftarrow g \\ c & f \leftarrow \mathit{not}\, d & \end{array}$$

Since $c \in \mathit{facts}(P_1)$ we apply success to the rule for a and negative reduction to the rule for g:

$$\begin{array}{lll} a \leftarrow \mathit{not}\, b & d \leftarrow \mathit{not}\, g, e & \\ b \leftarrow \mathit{not}\, a & e \leftarrow \mathit{not}\, g, d & h \leftarrow g \\ c & f \leftarrow \mathit{not}\, d & \end{array}$$

Now, as g is not head of any rule, we apply failure to the rule for h, and positive reduction to the rules for d and e:

$$\begin{array}{ll} a \leftarrow \mathit{not}\, b & d \leftarrow e \\ b \leftarrow \mathit{not}\, a & e \leftarrow d \\ c & f \leftarrow \mathit{not}\, d \end{array}$$

At this point, none of the four transformations $\{$`P`,`N`,`S`,`F`$\}$ is applicable. Fitting's model simply corresponds to the trivial interpretation of the program above: $(\{c\}, \mathcal{H} - \{a, b, c, d, e, f\})$, that is $(\{c\}, \{g, h\})$. To obtain the WFM, however, we must also consider rule `L`. Thus, we compute $\Gamma(\emptyset)$, i.e., the least model of the program:

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

which clearly corresponds to $\{a, b, c, f\}$. Then, as neither d nor e belong to this model, we delete the rules containing those atoms in their bodies:

$$\begin{array}{ll} a \leftarrow \mathit{not}\, b & c \\ b \leftarrow \mathit{not}\, a & f \leftarrow \mathit{not}\, d \end{array}$$

Finally, as d is not head of any rule, we can apply positive reduction to the rule for f:

$$\begin{array}{ll} a \leftarrow not\, b & c \\ b \leftarrow not\, a & f \end{array}$$

Now, no new transformation is applicable, and so, the WFM of P_1 is $(\{c, f\}, \mathcal{H} - \{a, b, c, f\}) = (\{c, f\}, \{d, e, g, h\})$.

4 WFSX and the coherence problem

As explained before, when using LP as a nonmonotonic reasoning tool, we usually need to incorporate a second type of negation to allow representing explicitly false facts. The addition of a second negation (named in different works as *classical*, *explicit* or *strong* negation) leads to the so called *extended logic programming* (ELP) [8, 9, 1, 2].

We will handle now two types of atoms: 'p', to represent that p has value *true*; and '$\overline{p}$', to represent[1] that p has value *false*. Normal logic programs dealing with this extended signature receive the name of *extended logic programs*. It is usual to call *objective literal*, denoted as L, to either p or $\overline{p}$ (that is, in our terminology, to any atom) and *default literal* to any *not* L. Besides, we will use the notation $\overline{L}$ to stand for the complementary objective literal of L, assuming $\overline{\overline{p}} \stackrel{\text{def}}{=} p$. Furthermore, given a set of objective literals M, we denote $\overline{M}$ to stand for their complementary literals: $\overline{M} \stackrel{\text{def}}{=} \{\overline{p} \mid p \in M\}$.

As explained in the introduction, the extension of the stable models semantics for ELP is extremely simple. We rule out the stable models containing any pair $\{p, \overline{p}\}$, that is, we take charge of explicit contradictions. We usually talk about *answer sets* [8] when referring to these (non-contradictory) stable models of extended logic programs.

At a first glimpse, it seems that something similar can be done for the WFS, considering the program to be *contradictory* when the WFM makes true both p and $\overline{p}$. However, as pointed out in [9, 1], a direct application of this method may lead to counterintuitive results. To understand the problem, it must be first noted that we handle now more possible epistemic states for a given atom. Rather than saying that L is *true* (when $L \in M^+$) or that it is *false* (when $L \in M^-$), we will say instead that it is *founded* or *unfounded*, respectively. In this way, we may distinguish between being *unknown* (that is, both $p \in M^-$ and $\overline{p} \in M^-$ are unfounded) and being *undefined*, which means that for some truth value of p we cannot establish whether it is founded or not ($p \notin M^+ \cup M^-$ or $\overline{p} \notin M^+ \cup M^-$).

In principle, we may have that p is undefined and $\overline{p}$ defined, or vice versa. However, it seems that there should exist a connection between complementary objective literals: when $L \in M^+$ is founded, we should have $\overline{L} \in M^-$ unfounded. Unfortunately, this property, called in [9] the *coherence principle*, is not satisfied

[1] The usual notation of ELP for $\overline{p}$ is $\neg p$. However, we use here the former to emphasize the view of $\overline{p}$ as an atom, so that we can compare to regular WFS.

by the usual definition of WFS. The typical counterexample is the program P_2:

$$
\begin{aligned}
& p \leftarrow \mathit{not}\, q \\
& q \leftarrow \mathit{not}\, p \\
& \overline{p}
\end{aligned}
$$

Intuitively, as we know that $\overline{p}$ is founded, its default negation *not* p should be immediately true, making q founded. That is, we should obtain the complete model $(\{\overline{p}, q\}, \{p, \overline{q}\})$, with $\overline{p}$ and q founded and their complements unfounded. However, it is easy to see that the WFM of P_2 is $(\{\overline{p}\}, \{\overline{q}\})$, which leaves both p and q undefined. The reason for this is that WFS does not provide any connection between p and $\overline{p}$ and so, we are not able to establish that the default literal $\mathit{not}\, p$ should be true when p is foundedly false.

To overcome this difficulty, Alferes and Pereira introduced a variation of WFS called WFSX (Well-Founded Semantics with Explicit Negation). For simplicity sake, we will just provide the iterative method to compute the WFM under WFSX semantics, which relies on a variation of the application of Γ^2. Most of the properties presented here have been directly extracted from [1].

Let r be a rule $H \leftarrow B$ of an extended normal logic program. By r_s we denote the seminormal version of r:

$$r_s \stackrel{\text{def}}{=} \ H \leftarrow B, \mathit{not}\, \overline{H}$$

Given a extended normal program P, we write P_s to stand for the seminormal version of P:

$$P_s \stackrel{\text{def}}{=} \ \{r_s \mid r \in P\}$$

For any set of atoms M and any fixed program P, we write $\Gamma_s(M)$ to denote the least model of P_s^M. By convention, we consider the function Γ_s *not defined* for a contradictory M, i.e., M containing both p and $\overline{p}$. A program is *contradictory* in WFSX iff it has no fixpoints for $\Gamma\Gamma_s$.

Proposition 4. *For non-contradictory programs, there exists a least fixpoint of $\Gamma\Gamma_s$, denoted as $lfp(\Gamma\Gamma_s)$.*

The combined function $\Gamma\Gamma_s$ is monotonic (on inclusion of sets of atoms), and so, its least fixpoint (when defined) can be computed by iteration on the least possible set of atoms, $\emptyset$. This is usually denoted as $\Gamma\Gamma_s \uparrow (\emptyset)$. The well-founded model is then defined in terms of $lfp(\Gamma\Gamma_s)$ as follows:

Definition 4. (WFSX's well-founded model)
The well-founded model *(WFM) of a (non-contradictory) extended logic program under the WFSX semantics corresponds to the 3-valued interpretation:*

$$\big(lfp(\Gamma\Gamma_s),\ \mathcal{H} - \Gamma_s(lfp(\Gamma\Gamma_s))\big)$$

It is interesting to note that the iteration of $\Gamma\Gamma_s$ may also be used to detect contradictory programs, as stated by the following property:

Proposition 5. *If, for some program P, the iteration of $\Gamma\Gamma_s \uparrow (\emptyset)$ reaches an interpretation that contains both p and $\overline{p}$, then the program P is contradictory in WFSX.*

Finally, another important property (proved in theorem 4.3.6 in [1]) is that WFSX actually generalizes WFS:

Proposition 6. *For programs without explicit negation, WFSX coincides with WFS.*

Unfortunately, this property does not help to establish the differences between WFS and WFSX when the program actually contains explicit negation. We will see how the WFSX extension of Brass et al's program remainder method will also be helpful to this purpose, showing that, in fact, WFSX obtains more or equal information than WFS.

5 A rewriting method for computing WFSX

The update of Brass et al's method for WFSX is very simple and natural. We will just incorporate the following two new transformations:

6. *Coherence failure* $\overset{\mathtt{C}}{\longmapsto}$: delete any rule r containing $\overline{L} \in body(r)$ such that $L \in facts(P)$.
7. *Coherence reduction* $\overset{\mathtt{R}}{\longmapsto}$: for any rule r, delete any default literal $(not\,\overline{L}) \in body(r)$ such that $L \in facts(P)$.

Notice how, in both cases, we simplify rules dealing with some $\overline{L}$ provided that L is trivially founded (it is a fact). In such a case, the coherence reduction, R, transforms $not\,\overline{p}$ into true. In fact, this transformation is the direct implementation of the coherence principle: default negation follows from explicit negation. As for transformation C, it allows replacing $\overline{p}$ by false, momentarily assuming that the program will be non-contradictory. As we will show later, even when this assumption is not eventually satisfied, the rewriting method (including these two new rules) is still capable of detecting contradiction.

We will also need to modify the definition of trivial interpretation:

Definition 5. (Trivial interpretation of an extended logic program) *Let P be an extended logic program not containing contradictory facts. The* trivial interpretation *of P is the 3-valued interpretation:*
$\big(facts(P), (\mathcal{H} - heads(P)) \cup \overline{facts(P)}\big)$

In other words, we consider now as false not only objective literals L that are not heads, but also those ones for which $\overline{L}$ is a fact. As an example, consider the program P_3:

$$\begin{array}{ll} a \leftarrow not\,b & \overline{p} \leftarrow b \\ b \leftarrow not\,a & p \end{array}$$

This program cannot be further transformed using $\{\mathtt{P},\mathtt{N},\mathtt{S},\mathtt{F},\mathtt{L}\}$. Therefore, its WFM would be $\big(\{p\},\{\overline{a},\overline{b}\}\big)$, which leaves a, b and $\overline{p}$ undefined. The trivial interpretation would contain, in this case, more information: $\big(\{p\},\{\overline{a},\overline{b},\overline{p}\}\big)$. It further considers $\overline{p}$ unfounded because p is a fact. The interest of the trivial interpretation is clarified by the following theorem:

Theorem 1. *Let P be a non-contradictory extended logic program, W its WFM (under WFSX) and U its trivial interpretation. Then $U \leq_F W$.*

That is, the trivial interpretation contain less or equal information than the WFM (under WFSX).

We prove now that the whole set of transformations, $\{\mathtt{P},\mathtt{N},\mathtt{S},\mathtt{F},\mathtt{L},\mathtt{C},\mathtt{R}\}$ are sound with respect to WFSX, that is, all the programs in a transformations chain either have the same WFM or are contradictory. To this aim, we provide first a lemma that establishes that the fixpoints for $\Gamma\Gamma_s$ remain unchanged after each transformation. We introduce here a remark on notation. When $P \xmapsto{\mathtt{x}} P'$ with some transformation rule $\mathtt{x}$, we write Γ' and Γ'_s to express that these functions implicitly correspond to the transformed program P', instead of P.

Lemma 1. *For any transformation $\xmapsto{\mathtt{x}}$ with $\mathtt{x} \in \{\mathtt{P},\mathtt{N},\mathtt{S},\mathtt{F},\mathtt{L},\mathtt{C},\mathtt{R}\}$, if $P \xmapsto{\mathtt{x}} P'$ then:*

(a) if $M = \Gamma\Gamma_s(M)$ then $\Gamma_s(M) = \Gamma'_s(M)$ and $M = \Gamma'\Gamma'_s(M)$.
(b) and vice versa, if $M = \Gamma'\Gamma'_s(M)$ then $\Gamma'_s(M) = \Gamma_s(M)$ and $M = \Gamma\Gamma_s(M)$.

Using this lemma, the result of soundness for all the transformations is almost straightforward.

Theorem 2. *The transformations $\xmapsto{\mathtt{x}}$, with $\mathtt{x} \in \{\mathtt{P},\mathtt{N},\mathtt{S},\mathtt{F},\mathtt{L},\mathtt{C},\mathtt{R}\}$ are sound w.r.t. WFSX. In other words, if a program P has a WFM then any resulting P', $P \xmapsto{\mathtt{x}} P'$ has the same WFM. Otherwise, if P has no WFM then P' has no WFM.*

Note that this theorem just points out that the transformations do not alter the final WFM (if there exists so), but it does not specify how to obtain this WFM. Before going further, we can already use this result to compare WFSX to WFS. As the transformations $\{\mathtt{P},\mathtt{N},\mathtt{S},\mathtt{F},\mathtt{L}\}$ used for WFS are a subset of the ones we have just proved to be sound for WFSX, we immediately get that:

Theorem 3. *Let P be any extended normal logic program and let W be its WFM under WFS. Then:*

(i) If W is contradictory (it makes true both some p and $\overline{p}$) then P is contradictory in WFSX.
(ii) If the program P has a WFM, X, under WFSX then $W \leq_F X$.

Note that the opposite for (i) does not hold, that is, we may have a program which has a non-contradictory WFM in WFS but has no solution in WFSX. As a simple counterexample, consider the program P_4:

$$a \leftarrow not\, a$$
$$\overline{a}$$

It is easy to see that in WFS, the WFM is not contradictory: we get $\overline{a}$ founded while a becomes *undefined* (remember that there is no connection between both atoms). When we move to WFSX, however, the above program would still be transformable (by coherence reduction) into the pair of facts a and $\overline{a}$, which are trivially contradictory. So, the program has no WFM under WFSX.

Theorem 3 also leads to the less general, but also useful result:

Corollary 1. *Let W be the WFM under WFS for some program P, and let W be non-contradictory and complete. Then, W is also the WFM of P under WFSX.*

To end up with the comparison, as for any hierarchical program, WFS leads to a complete WFM, we also obtain:

Corollary 2. *WFS and WFSX coincide for any hierarchical program.*

As we did for WFS, we say that a program P is the *program remainder* when no further transformation in $\{\mathtt{P},\mathtt{N},\mathtt{S},\mathtt{F},\mathtt{L},\mathtt{C},\mathtt{R}\}$ is applicable.

Theorem 4. (Main result)
Let P be the program remainder (of a possibly empty chain of transformations) and let P be free of contradictory facts. Then, the trivial interpretation of P is its WFM under WFSX.

The proof of the previous theorem (see appendix) makes use of the premise of non-applicability of all the transformations *excepting* the failure `F`. This means that this transformation is actually redundant. In fact, it is easy to see that failure is a particular case of positive loop detection `L`. Both transformations delete rules containing positive literals that satisfy a given condition which in `F` is stronger than in `L`: $L \notin heads(P) \Rightarrow L \notin \Gamma(\emptyset)$. Nevertheless, maintaining transformation `F` is interesting because of a pair of reasons. On the one hand, its computation means a considerably lower cost than loop detection and so, it may imply an efficiency improvement in many cases. On the other hand, it is interesting from the theoretical point of view, since, as we had seen, the subset of transformations $\{\mathtt{P},\mathtt{N},\mathtt{S},\mathtt{F}\}$ completely establish the Fitting's model of the program. Furthermore, we could even think about the extension of Fitting's semantics to cope with the coherence principle. This can be simply done by considering the set of transformations $\{\mathtt{P},\mathtt{N},\mathtt{S},\mathtt{F},\mathtt{C},\mathtt{R}\}$, that is, the ones for WFSX, excepting positive loop detection. A topic for future work could be to characterize the resulting semantics with a fixpoint definition or a models selection criterion.

6 Conclusion

We have shown how to extend Brass et al's transformation-based bottom-up computation of well-founded semantics (WFS) to cope with explicit negation, in the sense of Alferes and Pereira's WFSX semantics. The extension consists in adding two simple transformations to the ones already defined by Brass et al. for regular WFS. As expected, the additional transformations are directly related to the so-called *coherence principle* so that, whenever an objective literal L is founded, we must consider that its complementary literal $\overline{L}$ (i.e., its explicit negation) is unfounded.

The final method can be used for an efficient bottom-up computation of WFSX and it could be even applied to improve the efficiency of answer sets provers, due to their intermediate use of WFS. Future work could include a practical assessment in this sense. An specialized version of this method has been applied and implemented for the use of WFSX as an underlying semantics for causal representation of action domains [5, 4].

References

1. J. J. Alferes. *Semantics of Logic Programs with Explicit Negation.* PhD thesis, Facultade de Ciências e Tecnologia, Universidade Nova de Lisboa, 1993.
2. J. J. Alferes, L. M. Pereira, and T. C. Przymusinski. 'classical' negation in non-monotonic reasoning and logic programming. *Journal of Automated Reasoning*, 20(1):107–142, 1998.
3. S. Brass, J. Dix, B. Freitag, and U. Zukowski. Transformation-based bottom-up computation of the well-founded model. *Theory and Practice of Logic Programming*, to appear, 2001. (Draft version available at `http://www.cs.man.ac.uk/~jdix/Papers/01_TPLP.ps.gz`).
4. P. Cabalar. *Pertinence for causal representation of action domains.* PhD thesis, Facultade de Informática, Universidade da Coruña, 2001.
5. P. Cabalar, M. Cabarcos, and R. P. Otero. PAL: Pertinence action language. In *Proceedings of the 8th Intl. Workshop on Non-Monotonic Reasoning NMR'2000 (Collocated with KR'2000)*, Breckenridge, Colorado, USA, april 2000. (`http://xxx.lanl.gov/abs/cs.AI/0003048`).
6. M. Fitting. A kripke-kleene semantics for logic programs. *Journal of Logic Programming*, 2(4):295–312, 1985.
7. M. Gelfond and V. Lifschitz. The stable models semantics for logic programming. In *Proc. of the 5th Intl. Conf. on Logic Programming*, pages 1070–1080, 1988.
8. M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385, 1991.
9. L. M. Pereira and J. J. Alferes. Well founded semantics for logic programs with explicit negation. In *Proceedings of the European Conference on Artificial Intelligence (ECAI'92)*, pages 102–106, Montreal, Canada, 1992. John Wiley & Sons.
10. Alfred Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.
11. M. H. van Emden and R. A. Kowalski. The semantics of predicate logic as a programming language. *Journal of the ACM*, 23:733–742, 1976.
12. A. van Gelder. The alternating fixpoint of logic programs with negation. *Journal of Computing and System Sciences*, 47(1):185–221, 1993.
13. A. van Gelder, K. A. Ross, and J. S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, 1991.

Appendix A. Proofs of theorems.

Proof. **(theorem 1)** Let us call $W = (W^+, W^-)$ and $U = (U^+, U^-)$. By definition of $\leq_F$, we must prove $U^+ \subseteq W^+$ and $U^- \subseteq W^-$. Consider first any $L \in U^+ = facts(P)$. The result of applying Γ will always contain the fact L. By definition, $W^+ = \Gamma\Gamma_s(W^+)$, and so $L \in W^+$. Consider now $L \in U^-$. By definition, either $L \notin heads(P)$ or $L \in \overline{facts(P)}$, i.e., $\overline{L} \in facts(P)$. On the one hand, if $L \notin heads(P)$, the result of applying both Γ and Γ_s cannot contain L. Then, $L \notin \Gamma_s(W^+)$, i.e. $L \in \mathcal{H} - \Gamma_s(W^+)$ which, by definition, is W^-. On the other hand, if $\overline{L} \in facts(P)$, as $facts(P) \subseteq W^+$, then $\overline{L} \in W^+$. But then, the reduct $P_s^{W^+}$ will not contain any rule with L as head (in the seminormal program P_s, these rules contains *not* $\overline{L}$ in their bodies). As a result, $L \notin \Gamma_s(W^+)$ which means that $L \in \mathcal{H} - \Gamma_s(W^+)$ i.e. $L \in W^-$.

Proof. **(lemma 1)**

$\xmapsto{\mathtt{F}}$

As p is not head of any rule in P, the same applies for P_s, P' and P'_s. Thus, for any of these programs Q and for any interpretation M, when iterating $T_{Q^M} \uparrow (\emptyset)$, p is never obtained and so, any rule with p in the body is never used. Then, it can be deleted without varying the result. This directly implies that $\Gamma(M) = \Gamma'(M)$ and $\Gamma_s(M) = \Gamma'_s(M)$, for any M and so, the proofs for (a) and (b) become trivial.

$\xmapsto{\mathtt{L}}$

We will similarly show that, for any M and any $Q \in \{P, P_s, P', P'_s\}$: $p \notin T_{Q^M} \uparrow (\emptyset)$. As $p \notin \Gamma(\emptyset)$ we immediately get that p cannot belong to any application of $\Gamma(M)$, because the resulting reduct is a subset: $P^M \subseteq P^\emptyset$. Besides, as $P^\emptyset = P_s^\emptyset$, we have $p \notin \Gamma_s(\emptyset)$, and so p cannot belong to any $\Gamma_s(M)$ since again $P_s^M \subseteq P_s^\emptyset$. Finally, for Γ' and Γ'_s it suffices to see that $P' \subseteq P$ and $P'_s \subseteq P_s$. Then, the rules with p in the body are never used during the iteration $T_{Q^M} \uparrow (\emptyset)$ and so $\Gamma(M) = \Gamma'(M)$ and $\Gamma_s(M) = \Gamma'_s(M)$ for any M, being the proofs for (a) and (b) directly trivial.

$\xmapsto{\mathtt{P}}$

As proved in $\xmapsto{\mathtt{F}}$, since p is not head, it cannot belong to any application of $\Gamma, \Gamma_s, \Gamma'$ or Γ'_s. Besides, for any interpretation M such that $p \notin M$, it is easy to see that $P^M = P'^M$ and $P_s^M = P_s'^M$. Let us prove first (a). For any fixpoint $M = \Gamma\Gamma_s(M)$ we have that, as M is the result of applying Γ, $p \notin M$. But then, $\Gamma_s(M) = \Gamma'_s(M)$ (which is the first consequent of (a)), and in its turn, $p \notin \Gamma_s(M)$. Finally, this means that $M = \Gamma\Gamma_s(M) = \Gamma'\Gamma_s(M) = \Gamma'\Gamma'_s(M)$, that is M is fixpoint of $\Gamma'\Gamma'_s$. The proof for (b) is completely analogous.

$\xmapsto{\mathtt{S}}$

Notice first that, for any M, $\Gamma(M) = \Gamma'(M)$ and $p \in \Gamma(M)$, because p is a fact in P, and so, it will always valuated as true when applying T_P (resp. $T_{P'}$). Second, we show now that for any M such that $p \in M$ and $\overline{p} \notin M$, $\Gamma_s(M) = \Gamma'_s(M)$. The fact p occurs as the seminormal rule $p \leftarrow not\, \overline{p}$ both in P_s and in P'_s, but in P_s^M and $P_s'^M$, this rule will become again the original fact p. As a result, deleting

p from the rules will not vary the final outcome, i.e., $\Gamma_s(M) = \Gamma'_s(M)$. Besides, in P_s^M and $P_s'^M$ all the rules for $\overline{p}$ will be deleted (they contain in their bodies *not* p). This means that, additionally, $\overline{p} \notin \Gamma_s(M)$.

Now, we prove (a): let $M = \Gamma\Gamma_s(M)$. Then, as M is the result of applying Γ then $p \in M$. But, at the same time, as Γ_s is defined for M, $\overline{p} \notin M$. Therefore, we can apply the previous results, $\Gamma_s(M) = \Gamma'_s(M)$ (which is the first consequent of (a)). Let us call J to $\Gamma_s(M)$. Then, as we had seen, $\Gamma(J) = \Gamma'(J)$, i.e., $\Gamma\Gamma_s(M) = \Gamma'\Gamma_s(M) = \Gamma'\Gamma'_s(M)$. The proof for (b) is again analogous.

$\xmapsto{\mathtt{N}}$

First, note that for any M with $p \in M$, $P^M = P'^M$ and $P_s^M = P_s'^M$. Then, for proving (a), let $M = \Gamma\Gamma_s(M)$. As before, since p is a fact in P and M is the result of applying Γ, we get $p \in M$, and $\overline{p} \notin M$ (otherwise $\Gamma_s(M)$ would not be defined). Therefore $P_s^M = P_s'^M$ and $\Gamma_s(M) = \Gamma'_s(M)$ (the first part of (a)). Now note that the fact p in P becomes the seminormal rule $p \leftarrow \mathit{not}\, \overline{p}$ in P_s. However, in the reduct P_s^M (which is equal to $P_s'^M$) this rule becomes again the fact p, because $\overline{p} \notin M$. This means that $p \in \Gamma_s(M)$, and so, $P^{\Gamma_s(M)} = P'^{\Gamma_s(M)}$. It follows that $\Gamma\Gamma_s(M) = \Gamma'\Gamma_s(M) = \Gamma'\Gamma'_s(M)$. As always, the proof for (b) is analogous.

$\xmapsto{\mathtt{C}}$

Again, for any M with $p \in M$, in the reducts P_s^M and $P_s'^M$ all the rules with $\overline{p}$ as head are deleted (as they are seminormal, they contain *not* p in the body). As a result, $\overline{p}$ is never added when iterating the direct consequences operator, and so $\Gamma_s(M) = \Gamma'_s(M)$. Now, consider the proof for (a). If $M = \Gamma\Gamma_s(M)$, we have $p \in M$ (because of p being a fact and M the result of Γ) and so, the previous result is applicable: $\Gamma_s(M) = \Gamma'_s(M)$, which is the first part of (a). Now, as $\Gamma_s(M)$ is defined, we get that $\overline{p} \notin M$. But this means that when iterating direct consequences on the program $P^{\Gamma_s(M)}$, the fact $\overline{p}$ is never reached. Therefore, the rules with $\overline{p}$ in the body are never used, and so, the program $P'^{\Gamma_s(M)}$ has the same least model: $\Gamma\Gamma_s(M) = \Gamma'\Gamma_s(M) = \Gamma'\Gamma'_s(M)$. The proof for (b) is completely analogous.

$\xmapsto{\mathtt{R}}$

First observe that, for any M with $\overline{p} \notin M$, $P^M = P'^M$ and $P_s^M = P_s'^M$, and so, $\Gamma(M) = \Gamma'(M)$ and $\Gamma_s(M) = \Gamma'_s(M)$. Now, if $M = \Gamma\Gamma_s(M)$ we have (as in the two previous proofs): $p \in M$ and $\overline{p} \in M$. Therefore, we immediately have $\Gamma_s(M) = \Gamma'_s(M)$ (the first part of (a)). Now, note that $\overline{p} \notin \Gamma_s(M)$, because all the rules with $\overline{p}$ as head contain *not* p in their bodies, and we had that $p \in M$. By our first observation, this means that the reduct for P and P' w.r.t. $\Gamma_s(M)$ is the same one: $\Gamma\Gamma_s(M) = \Gamma'\Gamma_s(M) = \Gamma'\Gamma'_s(M)$. The proof for (b) is completely analogous.

Proof. **(theorem 2)** Simply note that the WFM in WFSX is defined as the three-valued interpretation $W = (W^+, W^-)$ with $W^+ = lfp(\Gamma\Gamma_s)$ and $W^- = \Gamma_s(M^+)$. As we have proved in lemma 1, any fixpoint of $\Gamma\Gamma_s$ is fixpoint of $\Gamma'\Gamma'_s$ and vice versa. So $W^+ = lfp(\Gamma'\Gamma'_s)$. Besides, as also proved in lemma 1, for any fixpoint M, $\Gamma_s(M) = \Gamma'_s(M)$. So $W^- = \Gamma'_s(W^+)$. Therefore, if W is WFM of P, it is WFM of P'. Finally, if P has no WFM, as the fixpoints for $\Gamma\Gamma_s$ and $\Gamma'\Gamma'_s$ coincide, then P' has no WFM.

Proof. **(theorem 3)** It follows from the previous results. Let $W = (W^+, W^-)$ and $X = (X^+, X^-)$. We consider (i) first. It is easy to see that any program P' containing the facts p and $\overline{p}$ is contradictory (has no fixpoints) in WFSX. As the transformations for WFS are also sound in WFSX, whenever we get the facts p and $\overline{p}$ in some of the transformed programs P', its WFM in WFSX is not defined, and so, the WFM for the original program is not defined as well. To prove (ii), it suffices with additionally applying lemma 1 for the resulting program P' after exhaustively applying all the WFS transformations. The facts of P' (i.e. W^+) are included in X^+ whereas the "non-head" atoms (i.e. W^-) are included in X^-. So $W \leq_F X$ for that program, and also for the original one.

Proof. **(theorem 4)** Let (U^+, U^-) be the trivial interpretation of P. We have to prove that:

i) $U^+ \stackrel{\text{def}}{=} facts(P) = lfp(\Gamma\Gamma_s)$
ii) $U^- \stackrel{\text{def}}{=} (\mathcal{H} - heads(P)) \cup \overline{facts(P)} = \mathcal{H} - \Gamma_s(U^+)$

We begin proving (ii). Consider $\Gamma_s(U^+)$, and more concretely, the reduct $P_s^{U^+}$. By non-applicability of $\stackrel{\mathtt{N}}{\longmapsto}$, program P cannot contain a rule with *not* p in the body, being p a fact of P. However, in P_s, any rule with $\overline{p}$ in the head contains *not* p in its body. So, $P_s^{U^+}$ is the result of deleting in P_s any rule whose head is in $\overline{facts(P)}$ plus the remaining default literals. As a first consequence $\Gamma_s(U^+) \cap \overline{facts(P)} = \emptyset$. But also, it is easy to see that $P_s^{U^+} \subseteq P^\emptyset$. By non-applicability of $\stackrel{\mathtt{L}}{\longmapsto}$, all the positive literals of P are included in $\Gamma(\emptyset)$ whereas by non-applicability of $\stackrel{\mathtt{C}}{\longmapsto}$, there is no positive literal of P in $\overline{facts(P)}$. As a result, all the rule bodies in $P^\emptyset$ are true w.r.t. $\Gamma(\emptyset)$ and so $\Gamma(\emptyset) = heads(P^\emptyset) = heads(P)$. Finally, since the rules $P^\emptyset - P_s^{U^+}$ are those with heads in $\overline{facts(P)}$ and these in their turn never occur in the bodies of P, we get that $\Gamma_s(U^+) = \Gamma(\emptyset) - \overline{facts(P)} = heads(P) - \overline{facts(P)}$. Then, it directly follows that $\mathcal{H} - \Gamma_s(U^+) = \mathcal{H} - (heads(P) - \overline{facts(P)}) = U^-$. Now, we proceed to prove (i). By lemma 1, the trivial interpretation (U^+, U^-) has less information than the WFM. So, $U^+ \subseteq lfp(\Gamma\Gamma_s)$ and it will suffice with showing that U^+ is simply a fixpoint: $\Gamma\Gamma_s(U^+) = U^+$. By (ii), $\Gamma\Gamma_s(U^+) = \Gamma(heads(P) - \overline{facts(P)})$. If we call $J = heads(P) - \overline{facts(P)}$, we want to establish the least model of P^J. By non-applicability of $\stackrel{\mathtt{P}}{\longmapsto}$ and $\stackrel{\mathtt{R}}{\longmapsto}$, given any *not* p in P, $p \in heads(P) - \overline{facts(P)}$. So, all the rules with default literals are deleted in P^J. Now, by non-applicability of $\stackrel{\mathtt{S}}{\longmapsto}$ in P, any body atom P^J cannot belong to $facts(P) = facts(P^J)$. This means that, when computing $T_{P^J} \uparrow (\emptyset)$, rules with nonempty body are never used. In other words, the least model of P^J, is $facts(P') = facts(P)$. That is, $\Gamma(J) = \Gamma\Gamma_s(U^+) = facts(P) = U^+$.