Derivation Procedures for Extended Stable Models

Luís Moniz Pereira and **Joaquim N. Aparício** and **José J. Alferes**
AI Centre, Uninova and DCS, U. Nova de Lisboa
2825 Monte da Caparica
Portugal

Abstract

We present derivation proof procedures for extended stable model semantics. Given program Π and goal G, G belongs to the well founded model of Π iff there is a WFM-derivation for G in Π. Likewise, given program Π and goal G, G belongs to some extended stable model of Π iff there is a XSM-derivation for G in Π. Correctness (completeness and soundness) of these procedures is discussed. Example derivations are exhibited, as well as a simple Prolog implementation that directly mirrors the procedures.

1 Introduction

Well Founded Semantics (WFS) [Van Gelder *et al.*, 1990] adequately captures various forms of hypothetical reasoning [Pereira *et al.*, 1991c, Pereira *et al.*, 1991d, Pereira *et al.*, 1991b, Pereira *et al.*, 1991a] if we interpret the well-founded model (WFM) of a program Π as a (possibly incomplete) core view of the world, the extended stable models (XSMs) specifying alternative complementary consistent views of the world, all of each containing the core WFM.

The paper is organized as follows: in section 2 we review well founded semantics. In section 3 we define WFM-derivations, discuss their correctness, and give examples. Next, in section 4, we define XSM-derivations and discuss their correctness. Finally, in section 5, a Prolog implementation is produced, directly reflecting the derivation procedures mentioned. More details can be found in an extended version of this paper [Pereira *et al.*, 1990].

By a logic program Π we mean a finite set of universally closed rules of the form: $H \leftarrow L_1, \ldots, L_n$ where $n \geq 0$, H is an atom and the L_i's are literals. When $n = 0$, we also write $H \leftarrow \mathbf{t}$, where $\mathbf{t}$ stands for an atom satisfied in all models. Negative literals are expressed as $\sim A$ where A is an atom. We denote by Lit(Π) the set of literals in ground(Π), the Herbrand instantiation of program Π. We denote the well founded model of a program Π by $WFM(\Pi)$, and an extended stable model (which may be the well founded one) by $XSM(\Pi)$.

2 The Extended Stable Model Semantics

In this section we characterize the Well Founded and Extended Stable Models of a program, based on [Przymusinska and Przymusinski, 1990]. Alternative definitions of the Well Founded Semantics can be found in [Van Gelder *et al.*, 1990] or in [Przymusinski, 1989]. Because the semantics is 3-valued, we begin by defining 3-valued interpretations.

Definition 2.1 A 3-*valued Herbrand interpretation* I of a first-order language L is any pair $\langle T; F\rangle$, where T and F are disjoint subsets of the Herbrand base H. T contains all ground atoms true in I, F contains all ground atoms false in I, and the truth value of the remaining atoms, those in $U = H - (T \cup F)$, is undefined (or unknown).

An alternative way to represent an interpretation $I = \langle T; F\rangle$ is $I = T \cup \{\sim L | L \in F\}$.

Proposition 2.1 Any interpretation $I = \langle T; F\rangle$ can be equivalently viewed as a function $I : H \to V$ where $V = \{0, 1/2, 1\}$, defined by:

- $I(A) = 0$ if $A \in F$
- $I(A) = 1/2$ if $A \in U$
- $I(A) = 1$ if $A \in T$

Definition 2.2 The function $I : H \to V$ can be recursively extended to the *truth valuation function* $\hat{\imath}$: $Lit(\Pi) \to V$ defined on the set Lit(Π) of all literals of the language as follows, where A is a ground atom:

- $\hat{\imath}(A) = I(A)$
- $\hat{\imath}(\sim A) = 1 - I(A)$

Definition 2.3 A *non-negative program* is a program whose premises are either positive atoms or the special proposition $\mathbf{u}$. Every interpretation I satisfies $I(\mathbf{u}) = 1/2$, and so $\hat{\imath}(\sim\mathbf{u}) = 1/2$. $\mathbf{u}$ denotes the undefined (or unknown) value.

Theorem 2.1 *(Generalization of [Van Emden and Kowalski, 1976])* Every non negative logic-program has a unique least[1] 3-valued model.

[1] If I and J are two intgerpretations then we say that $I \leq J$ if $I(A) \leq J(A)$ for any ground atom A. If $\mathcal{I}$ is a

Next we define the program transformation Π/M (Π modulo M), which is a 3-valued extension to the 2-valued transformation in [Gelfond and Lifschitz, 1988].

Definition 2.4 Let Π be a logic program and let I be a 3-valued interpretation. By the *extended GL-transformation of* Π *modulo* I we mean a new (non-negative) program Π/I obtained from Π by performing the following three operations:

- Removing from Π all rules which contain a negative premise $L = \sim A$ such that $\hat{\imath}(L) = 0$.
- Replacing in all remaining rules those negative premises $L = \sim A$ which satisfy $\hat{\imath}(L) = 1/2$ by $\mathbf{u}$.
- Removing from all the remaining rules those negative $L = \sim A$ which satisfy $\hat{\imath}(L) = 1$.

Since the resulting program Π/I is non-negative, by theorem 2.5, it has a unique least 3 valued model. We define $\Gamma^*(I)$ (a generalization of the Γ operator [Gelfond and Lifschitz, 1988]) to be the 3-valued least model of Π/I.

Definition 2.5 A 3-valued interpretation I of a logic program Π is called an *Extended Stable Model XSM* of Π iff $\Gamma^*(I) = I$.

In order to check if I is an XSM of a program we give a constructive definition of Γ^* operator. For this purpose we define Ψ^*, a generalization of the Van Emden-Kowalski operator Ψ.

Definition 2.6 Let Π be a non-negative program, I an interpretation of Π and A is a ground atom. Then $\Psi^*(I)$ is an interpretation defined as follows:

- $\Psi^*(I)(A) = 1$ iff there is a rule $A \leftarrow A_1, \ldots, A_n$ in Π such that $I(A_i) = 1$ for all $i \leq n$.
- $\Psi^*(I)(A) = 0$ iff for every rule $A \leftarrow A_1, \ldots, A_n$ in Π there is an $i \leq n$ such that $I(A_i) = 0$.
- $\Psi * (I)(A) = 1/2$, otherwise

We define $\Psi^{*\uparrow n} = \Psi^*(\Psi^{*\uparrow n-1})$ and $\Psi^{*\uparrow 0} =< \{\}, H >$

Definition 2.7 Let I be an interpretation and Π a logic program. $\Gamma^*(I)$ for Π can be defined as $\Gamma^*(I) = \Psi^{*\uparrow w}(\Pi/I)$.

This alternative definition of the WF semantics was proved equivalent to the original one [Van Gelder *et al.*, 1990] in [Przymusinska and Przymusinski, 1990] by the following theorem:

Theorem 2.2 The Well Founded Model of a program Π is the F-least[2] Extended Stable Model of Π. Consequently the WFS coincides with the $XSMS$.

collection of interpretations, then an interpretation $I \in \mathcal{I}$ is called minimal in $\mathcal{I}$ if there is no interpretation $J \in \mathcal{I}$ such that $J \leq I$ and $I \neq J$. An interpretation I is called least in $\mathcal{I}$ if $I \leq J$, for any other interpretation $J \in \mathcal{I}$. A model of a theory R is called minimal (resp. least) if it is minimal (resp. least) among all models of R.

[2]If $I = \langle T, F\rangle$ and $I' = \langle T', F'\rangle$ are two interpretations, then we say that $I \leq_F I'$ iff $T \subseteq T'$ and $F \subseteq F'$. An interpretation I is called F-least in a collection of interpretations $\mathcal{I}$ if $I \leq_F J$ for any interpretation $J \in \mathcal{I}$ [Fitting, 1985]

To obtain a constructive definition of the WFM of a program Π we use the following sequence of $\{I_\alpha\}$ of interpretations of Π:

- $I_0 = \langle\{\}, \{\}\rangle$
- $I_{\alpha+1} = \Gamma^*(I_\alpha)$

The WF model of Π is the least fixed point of this sequence, i.e. I_λ [Przymusinska and Przymusinski, 1990].

3 Derivation Procedure for the Well Founded Model

Now we present a derivation procedure such that given a program Π and a goal G the derivation succeeds iff G is in $WFM(\Pi)$. The procedure is defined over the ground instance of Π, the set of all ground instances of the rules in Π with respect to its Herbrand Universe. Without loss of generality we can assume that Π has been already *instantiated* and thus consists of a (possibly infinite) set of propositional rules.

Definition 3.1 A *positive* (resp. *negative*) interpretation I is a set of positive (resp. negative) literals from Lit(Π).

Definition 3.2 A *context* C_n is an ordered set of positive or negative interpretations. Let S_i be a positive or a negative interpretation; C_n denotes the context $S_1 S_2 \ldots S_n$. C_n is a *negative* (resp. *positive*) *context* if S_n is a negative interpretation (resp. positive interpretation). $C_n + G$ denotes the concatenation $S_1 S_2 \ldots S_n G$. A literal G is in context C_n ($G \in C_n$ for short) $G \in S_n$.

A context C_n implicitly defines an interpretation $I_n(C_n)$ which is the set of literals in partial interpretations S_n i.e. $I_n(C_n) = \cup_{(i \leq n)} S_i$, and for no atom A both A and $\sim A$ belong to it.

Definition 3.3 A *contextual formula* (C-formula) is a pair $C\#F$, where C is a context and F is an expression built from atoms with conjunctions and negations. An empty C-formula is the C-formula $C\#\mathbf{t}$.

By the interpretation $I(C\#F)$ we mean the interpretation $I(C)$ associated with context C.

Definition 3.4 (WFM-derivation) Let $R_j = \langle C_j\#F_j; I_j\rangle$ where C_j is a context, and I_j a set of literals. A WFM-*derivation* from R_i to R_n is a sequence from $< C_i\#F_i; I_i >$ to $< C_n\#F_n; I_n >$ such that for any $\langle C_k\#F_k; I_k\rangle$ ($i \leq k \leq n$), the following derivation rules apply (where we assume $C_{k+1} = C_k$ and $I_{k+1} = I_k$ unless stated otherwise).

D1.1. if $F_k = \sim G$ and there is no rule $G \leftarrow B$ then $R_{k+1} = \langle C_k + \sim G\#\mathbf{t}; I_k \cup \{\sim G\}\rangle$.

D1.2. if $F_k = \sim G$ and $\sim G \in C_k$ then $F_{k+1} = \mathbf{t}$.

D2.1. if $F_k = \sim G$ and there are r rules for G with G_i ($1 \leq i \leq r$) as head, $\sim G \notin C_k$, $G \notin I_k$

$$\begin{array}{rl} G_1 \leftarrow & B_{11}, \ldots, B_{m1} \\ & \ldots \\ G_r \leftarrow & B_{1r}, \ldots, B_{m'r} \end{array}$$

in Π and $G \notin I_k$, then $R_{k+1} = \langle C_k + \sim G\#\ \tilde{G}_1, \ldots, \tilde{G}_r; I_k \cup \{\sim G\}\rangle$ where $\tilde{G}_i$ is a short hand for $\sim(B_{1i}, \ldots, B_{mi})$

D2.2. if $F_k = \sim(G_1, \ldots, G_m)$ then $R_{k+1} = \langle C_k \# \sim G_i; I_k \rangle$ for some $1 \leq i \leq m$.

D3. if $F_k = G$ and $G \notin I_k$ then for some rule $G \leftarrow B_1, \ldots, B_m \in \Pi$, $R_{k+1} = \langle C_k + G\#(B_1, \ldots, B_m); I_k \cup \{G\} \rangle$

D4. if $F_k = (g, G)$ then $R_{k+1} = \langle C_k \# G; I_{gk} \rangle$ if there is a derivation from $\langle C_k \# g; I_k \rangle$ to $\langle L \# t; I_{gk} \rangle$.

There is a WFM-derivation for G in Π iff there is a sequence from $\langle \{\} \# G; \{\} \rangle$ to $\langle L \# \mathbf{t}; I \rangle$, for some I.

We argue these rules are intuitive when one recalls the definition of 3-valued model [cf. section 2]: rule D1.1 establishes the CWA[3]. Rule D1.2 says that a literal may support on itself when proving its falsity[4]. Rule D2.1 says that for an atom to be interpreted as false it has to be proven false in all definitions for it. Rule D2.2 says that for a body of a rule to be false it is enough to prove some literal in the body to be false[5]. Rule D3 says that for a literal to be true it is enough to have a rule with all body literals true[6]. Rule D4 says that a conjunction of formulas is true if each element is true. Note that $I_k = \bigcup_{i \leq n} I_i \cup \{F_k\}$ if F_k is a literal.This means we don't need to explicitly record I_k at each step k but simply consider all the C_j in $(j < k)$. Note that rule D4 introduces a notion of sub-derivation. We now present examples illustrating the application of the derivation rules.

Example 1 For the program $\Pi = \{p \leftarrow \sim q\}$, a WFM-derivation for p is:

$$
\begin{array}{ll}
\{\}\#p & \\
\{p\}\# \sim q & D3 \\
\{p, \sim q\}\#\mathbf{t} & D1.1
\end{array}
$$

Example 2 Let Π be the "work-tired" example [Przymusinski, 1990] with the obvious abbreviations:

$$
\begin{array}{ll}
w \leftarrow t & (1) \\
t \leftarrow \sim s & (2) \\
s \leftarrow \sim w & (3) \\
a \leftarrow w, \sim p & (4) \\
p &
\end{array}
$$

A failed derivation for $\sim a$ is:

$$
\begin{array}{ll}
\{\}\# \sim a & \\
\{\sim a\}\# \sim(w, \sim p) & D2.1 \text{ and } (4) \\
\{\sim a\}\# \sim w & D2.2 \\
\{\sim a \sim w\}\# \sim(\sim t) & D2.1 \text{ and } (1) \\
\{\sim a \sim w\}\#\mathbf{t} & D2.2 \text{ and } \sim(\sim t) = t \\
\{\sim a \sim w, t\}\# \sim s & D3 \text{ and } (2) \\
\{\sim a \sim w, t, \sim s\}\#w & D2.2, D2.1 \text{ and } \sim(\sim w) = w \text{ and } (3) \\
 & \text{failure to continue}
\end{array}
$$

[3]Note that according to the second point of the definition of the Ψ^* operator, if an atom A has no rules in Π, then $\Psi^*(I)(A) = 0$ for all interpretations I of P.

[4]According to the definition of $\Psi^{*\uparrow w}$, we start with $\Psi^{*\uparrow 0} = \langle \{\}, \mathcal{H} \rangle$. So every negative literal may depend on itself. There is no corresponding rule for positive literals

[5]Again these two rules follow exactly the second point of definition 2.6

[6]This is what is stated in the first point of the definition of Ψ^*

and a successful derivation for $\sim a$:

$$
\begin{array}{ll}
\{\}\# \sim a & \\
\{\sim a\}\# \sim(w, \sim p) & D2.1 \text{ and } 4 \\
\{\sim a\}\#p & D2.2, D2.1 \text{ and } \sim(\sim p) = p \text{ and } (4) \\
\{\sim a, p\}\#\mathbf{t} & D3
\end{array}
$$

Example 3 There is no derivation for (p, q) in program $\Pi = \{p \leftarrow \sim q,\ q \leftarrow \sim p\}$.

$$
\begin{array}{lll}
\{\}\#(p, q) & I_0 = \emptyset & \\
\{\}\#p & I_1 = \emptyset & \\
\{p, \sim q\}\#p & I_3 = \{p, \sim q\} & \text{failure}
\end{array}
$$

Note that even the selected literal order differed the derivation definition is still applicable because we keep a record of I_k at each derivation step.

3.1 WFM-Trees

A derivation from $\langle \emptyset \# G; \emptyset \rangle$ to $\langle L \# \mathbf{t}; I \rangle$ may be interpreted as the construction of certain trees to be introduced now. These trees are obtained from the derivation rules with the following in mind: at each derivation step $\langle C_k \# L_k; I_k \rangle$, C_k is the ordered ancestor list of literal L_k which is a node of the tree, and I_k is the set of all literals in the nodes already visited by the derivation procedure. In the following we will omit the special symbol $\mathbf{t}$.

Definition 3.5 (WFM-Tree) A *WFM-tree* for G given program Π $WFM(G, \Pi)$ is a finite tree with root G, such that if N is the literal of a node of the tree then:

WFM-I If N is negative, let $N = \sim L$ and:

1) if there are no rules for L then N is a leaf (rule D1.1)
2) if $\sim L$ has an identical ancestor A and all literals in the branch from $\sim L$ to A are negative then N is a leaf (rule D1.2)
3) if there are r rules for L: (rules D2.1+D2.2)

$$
\begin{array}{ll}
L_1 \leftarrow & B_{11}, \ldots, B_{k1} \\
 & \ldots \\
L_r \leftarrow & B_{1r}, \ldots, B_{k'r}
\end{array}
$$

in Π, then node $\sim L$ has r immediate descendents $\sim B_{j1}, \ldots, \sim B_{hr}$, each one selected from the body of a different rule.

WFM-II If N is positive then:

1) If there is a fact N in Π, then N is a leaf (rule D3)
2) the n immediate descendents are those literals $B_1 \ldots B_n$, such that a rule $N \leftarrow B_1, \ldots B_n$ exists in Π (rule D4)

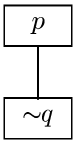
By considering all possible choices of rules and literals all WFM-trees are obtained.

Proposition 3.1 For every program Π and goal G there is a WFM-derivation for G in Π iff there is a WFM-Tree(G,Π).

Note that a context C in a C-formula $C\#L$, expresses a branch from the root of the tree to node L. Given a tree and a node, its context is implicitly defined by the ordered set of its ancestors, with its father being the most recent one in C (the rightmost). The condition $G \notin I_k$ in D2.1 and D3 means that if a tree has a node L (resp. $\sim L$) then it may have no node $\sim L$ (resp. L).

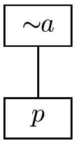
We may think of the derivation rules as stating conditions for re-writing a literal, possibly by cancelation with a previous ancestor in the tree. For the well founded model derivation procedure the only possible cancelation of a literal with an ancestor is provided by rule D1.2.

Example 4 The WFM-Tree for p given program $\{p \leftarrow \sim q\}$ is:



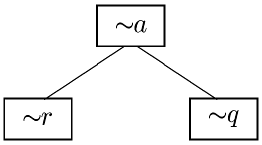
Example 5 The WFM-Tree for $\sim a$ for the program below is:

$$\begin{array}{ll} w \leftarrow t & (1) \\ t \leftarrow \sim s & (2) \\ s \leftarrow \sim w & (3) \\ a \leftarrow w, \sim p & (4) \\ p & \end{array}$$



Example 6 The WFM-Tree for $\sim a$ for the program below is:

$$\begin{array}{ll} a \leftarrow \sim p, r & (1) \\ a \leftarrow \sim q & (2) \\ p \leftarrow \sim p & (3) \end{array}$$



Note the WFM of this program is $\{\sim a, \sim r, \sim q\}$.

3.2 Properties of WFM-Trees

By definition WFM-Trees are finite. The only type of leaf nodes in a WFM-tree are:

1) a positive literal with no identical ancestor
2) a negative literal with no identical ancestor
3) a negative literal with an identical ancestor

Proposition 3.2 Every branch from the root literal has at most one node with an identical ancestor.

Proposition 3.3 If a leaf node for literal L has an identical ancestor then L is a negative literal.

Proposition 3.4 If a leaf node for literal L has no identical ancestor and is a negative literal $L = \sim H$ then there are no rules for H in Π.

Proposition 3.5 If a leaf node has a positive literal L then no node in the branch from the root literal to L has an identical ancestor literal, and there is a fact L in program Π.

Lemma 3.1 *(Leaves are in the WFM)* If T is a $WFM(G,\Pi)$ tree for G in Π then:

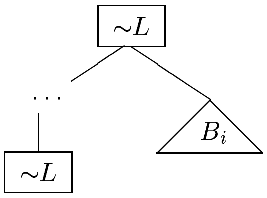
1) a positive literal which is in a leaf node, and has no identical ancestor, is in $WFM(\Pi)$
2) a negative literal which is in a leaf node, and has no identical ancestor, is in $WFM(\Pi)$
3) a negative literal which is in a leaf node, and has an identical ancestor is in $WFM(\Pi)$

Proof:

3.1.1) if a positive literal L is in a leaf node, and has no identical ancestor, then there is a fact L in Π and facts are always in $WFM(\Pi)$.

3.1.2) if a negative literal $\sim L$ is in a leaf node, and has no identical ancestor, then there are no rules for L in Π. For an atom L such that no rules for it exist in Π, $\sim L$ is in $WFM(\Pi)$.

3.1.3) if a negative literal $\sim L$ is in a leaf node and has an identical ancestor, we have:



i.e. there is at least one chain from L to L, and possibly a rule for L, such as (3) in the program below:

$$\begin{array}{ll} (1) & L \leftarrow \ldots H \ldots \\ (2) & H \leftarrow \ldots L \ldots \\ (3) & L \leftarrow B_1 \ldots B_n \end{array}$$

i.e. in Π there is a chain of positive literals $L \ldots H \ldots L$, and the well founded model contains $\sim L \ldots \sim H \ldots \sim L$.

Note that for the model not to contain $\sim L$ by (3) all B_i would have to differ from false. But in that case the tree named B_i in the figure above could not exist, nor the father node $\sim L$.

Lemma 3.2 Given a WFM-tree for G then, for any internal node literal H, if its immediate descendents $D_1 \ldots D_n$ are in the well founded model, then H is in the WFM.

Proof: The proof follows easily from the observation that the only tree formation rules introducing descendents nodes are WFM-I.3 and WFM-II.2 and the definition of 3-valued interpretation (cf. footnotes 5 and 6).

Theorem 3.3 *(Soundness of WFM-Trees)* Let Π be a program and L a literal. If there is a WFM-tree for L then L is in $WFM(\Pi)$.

Proof: Follows directly from lemma 3.1 and lemma 3.2 above.

Corollary 1 Given program Π and literal G such that $G \in WFM(\Pi)$ then all literals in the $WFM(G,\Pi)$ are in $WFM(\Pi)$.

Theorem 3.4 *(Completeness of WFM trees)* Let Π be a program and L a literal. If L is in $WFM(\Pi)$ there is a WFM-tree for L.

Proof: Appears in the extended version.

4 Derivation Procedure for Extended Stable Models

We present now a derivation procedure such that, given a program Π and a literal G, the derivation succeeds if G is in some $XSM(\Pi)$.

Definition 4.1 (**XSM-Derivation)**
Let $R_j = \langle C_j \# F_j; I_j \rangle$ where C_j is a context and I_ja set of literals. A XSM-derivation from R_i to R_n is a sequence from $\langle C_i \# F_i; I_i \rangle$ to $\langle C_n \# F_n; I_n \rangle$ such that for any $\langle C_k \# F_k; I_k \rangle$ $(i \leq k \leq n)$ the following derivation rules apply (where we assume $C_{k+1} = C_k$ and $I_{k+1} = I_k$ unless stated otherwise):

D1.1 if $F_k = \sim G$ and there is no rule $G \leftarrow B$ then $R_{k+1} = \langle C_k + \sim G \# t; I_k \cup \{\sim G\} \rangle$

D1.2 if $F_k = \sim G$ and $\sim G \in C_m$, $m \leq k$ then $F_{k+1} = \mathbf{t}$

D2.1 if $F_k = \sim G$, $\sim G \notin C_k$, $G \notin I_k$, and there are r rules for G with G_i as head

$$\begin{array}{rl} G_1 \leftarrow & B_{11}, \ldots, B_{m1} \\ & \ldots \\ G_r \leftarrow & B_{1r}, \ldots, B_{m'r} \end{array}$$

in Π then $C_{k+1} = C_k + \sim G \# \ \widetilde{G}_1, \ldots, \widetilde{G}_r$, and $I_{k+1} = I_k \cup \sim G$ where $\widetilde{G}_i$ is a short hand for $\sim(B_{1i}, \ldots, B_{mi})$

D2.2 if $F_k = \sim(G_1, \ldots, G_m)$ then $R_{k+1} = \langle C_k \# \sim G_i; I_k \rangle$ for some $1 \leq i \leq m$

D3.1 if $F_k = G$, $\sim G \notin I_k$ and $G \notin I_k$ then for some rule $G \leftarrow B_1, \ldots, B_m$ in Π, $R_{k+1} = \langle C_k + G \# (B_1, \ldots, B_m); I_k \cup \{G\} \rangle$

D3.2 if $F_k = G$ and $G \in C_m$ $(m < k)$ then $F_{k+1} = \mathbf{t}$

D4 if $F_k = (g, G)$ then $R_{k+1} = \langle C_k \# G; I_{gk} \rangle$ if there is a derivation from $\langle C_k \# g; I_k \rangle$ to $\langle L \# t; I_{gk} \rangle$.

Note these derivation rules include, as expected, those for well WFM-derivations.

Example 7

$$\begin{array}{ll} a \leftarrow \sim b & (1) \\ b \leftarrow \sim a & (2) \end{array}$$

A XSM-derivation for a is as follows:

$$\begin{array}{ll} \{\}\#a & \\ \{a\}\# \sim b & D3.1 \text{ and } (1) \\ \{a \sim b\}\#a & D2.1 \text{ and } D2.2 \text{ and } \sim(\sim a) = a \text{ and } (2) \\ \{a \sim b\}\#\mathbf{t} & D3.2 \end{array}$$

The XSM-derivation for $\sim a$ is as follows:

$$\begin{array}{ll} \{\}\# \sim a & \\ \{\sim a\}\#b & D2.1 \text{ and } D2.2 \text{ and } \sim(\sim b) = b \text{ and } (1) \\ \{\sim ab\}\# \sim a & D3.1 \text{ and } (2) \\ \{\sim ab\}\#\mathbf{t} & D1.2 \end{array}$$

4.1 XSM-trees

Definition 4.2 (**XSM-tree)** A XSM-tree for G given program Π, $XSM(G, \Pi)$ is a finite tree with root G, such that if N is the literal of a node of the tree then:

XSM-I If N is negative, then $N = \sim L$ and:

1) if there are no rules for L then N is a leaf (rule D1.1)
2) if $\sim L$ has an identical ancestor A and all literals in the branch $\sim L$ to A are negative then N is a leaf (rule D1.2)
3) if there are r rules for L: (rules D2.1+D2.2)

$$\begin{array}{rl} L_1 \leftarrow & B_{11}, \ldots, B_{k1} \\ & \ldots \\ L_r \leftarrow & B_{1r}, \ldots, B_{k'r} \end{array}$$

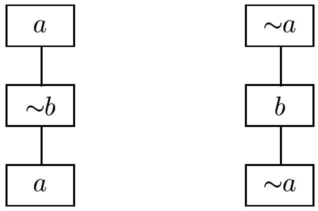
in Π, then node $\sim L$ has r immediate descendents $\sim B_{j1} \ldots \sim B_{hr}$, each one selected from the body of a different rule
4) if $\sim L$ has an identical ancestor A with some positive literal in between, then N is a leaf (rule D1.2)

XSM-II If N is positive then:

1) if there is a fact N in Π, then N is a leaf (rule D3.1)
2) the n immediate descendents are those literals $B_1 \ldots B_n$, such that a rule $N \leftarrow B_1 \ldots B_n$ exists in Π(rule D4)
3) if N has an identical ancestor A, with some negative literal in between, then N is a leaf (rule D3.2)

A XSM-Tree has two types of leaves not appearing in the WFM-Tree, namely: i) positive literal leaf nodes having an identical ancestor and with some negative literal in between; and ii) negative literal leaf nodes having an identical ancestor and with some positive literal in between.

Example 8 Although the program $\{a \leftarrow \sim b; b \leftarrow \sim a\}$ has no WFM-Tree it has the following XSM-Trees (among others):



The soundness proof is similar to that of WFM-Trees, but the equivalent to lemma 3.1 is now:

Lemma 4.1 *(Leaves are in the XSM)*If T is a XSM tree for G then:

4.1.1) a positive literal in a leaf node, having no identical ancestor, is in some $XSM(\Pi)$
4.1.2) a negative literal in a leaf node, having no identical ancestor, is in some $XSM(\Pi)$
4.1.3) a negative literal in a leaf node, having an identical ancestor, is in some $XSM(\Pi)$
4.1.4) a positive literal in a leaf node, having an identical ancestor and some negative literal in between, is in some $XSM(\Pi)$
4.1.5) a negative literal in a leaf node, having an identical ancestor and some positive literal in between, is in some $XSM(\Pi)$

Proof: Proof of 1,2 and 3 4.1.i, 4.1.ii and 4.1.iii follows from lemma 3.1 and the fact that all literals in the $WFM(\Pi)$ are in all $XSM(\Pi)$.

4.2 Completeness of XSM-Tree

The completeness proof may be found in the full version.

5 A Prolog implementation

We present here a Prolog implementation of the above procedures which provides them with an operational semantics. Lines are numbered for referencing. The interpreter is basically the implementation of the derivation rules of the procedures [cf. section 3 and 4] plus loop checking. Thus the execution always terminates. The code may be used for WFM-derivations as below, or for XSM-derivations by deleting line (7.4).`mb` is the `member` predicate.

```
s(~((G1,G)),I,O,An) :- s(~G1,I,O ,An).  (1)
s(~((G1,G)),I,O,An) :-!,s(~(G) ,I,O ,An).  (2)
s((G1,G),I,O,An):-!,s(G1,I,II,An),s(G,II,O,An).  (3)
s(~(~G) ,A ,B ,C ) :- !, s(G,A,B,C). (4)
s(~G,I,_,An) :- mb(G ,I), !, fail.  (5)
s( G,I,_,An) :- G ~(_), mb(~G,I), !, fail.  (6)
s(G,I,[G|I],An) :- d(G,An,D), (7)
     ( D = z, G = (~_), !; D = z, !, fail;
     D = e, !, fail; % wfm (7.4)
     D = e, !).
s(~G,_ ,_ ,_ ) :- (G <<- ), !, fail.  (8)
s(~G,I,O,An) :- !, findall((G<<-B),(G<<-B) ,L), (9)
     all_out(~G,L,[~G|I],O,[~G|An]).
s(G,I,[G|I],An) :- (G <<- ).  (10)
s(G,I,O ,An) :- (G<<-B),s(B,[G|I],O,[G|An]).(11)

all_out(~G,[] ,I,I,An).  (12)
all_out(~G,[(G<<-(B1))|Gn],I,O,An) :- (13)
     s(~(B1),I,II,An), all_out(~G,Gn,II,O,An).

d(~G,[(~X)|An],D) :- !, d(~G,[(~X)|An],e,ZNZ,D).
d(~G,[ X |An],D):-!,d(~G,[ X |An],o,ZNZ,D).
d(G,[(~X)|An],D):-!,d( G,[(~X)|An],o,ZNZ,D).
d(G,[ X|An],D):- d( G,[ X |An],e,ZNZ,D).
d(G,[G|T] ,e ,Z ,z) :- var(Z), !.
d(G,[G|T] ,D ,nz,D) :- !.
d(G,[(~_),(~X)|T],EO,Z,D):-!,d(G,[(~X)|T],EO,Z,D).
d(G,[(~_),(X) |T],e,_,D) :-!,d(G,[X|T] ,o,nz,D).
d(G,[(~_),(X) |T],o,_,D) :-!,d(G,[X|T] ,e,nz,D).
d(G,[_ ,(~X)|T],e,_,D) :-!,d(G,[(~X)|T],o,nz,D).
d(G,[_ ,(~X)|T],o,_,D) :-!,d(G,[(~X)|T],e,nz,D).
d(G,[_ ,X |T],EO,Z ,D) :- d(G,[X|T],OE,Z,D). (25)
```

Lines (1)-(2) correspond to rule D2.2. Line (3) corresponds to rule D4. Line (4) is obvious. Lines (5)-(6) ensure the condition $G \notin I_k$. Lines (9), (12) and (13) correspond to D2.1. Lines (14) - (25) do an ancestor search for literal G: the result of this search can be: the literal G has an ancestor node in the current context `(D=z)`;the literal G has an ancestor node in some context other than the current one; the literal G has no ancestor goal and the predicate `dist/3` in rule (7) fails. Line (7.2) corresponds to derivation rule D1.2. Line (7.5) corresponds to rules D1.2 and D3.2. Line (12) accounts for rule D1.1. Line (7.3) avoids the potential loop of a positive literal G with an identical ancestor G with no negative literals in between. A literal G is in $WFM(\Pi)$ (resp. $XSM(\Pi)$) if the query `?- s(G,[],L,[])` succeeds. The procedure returns in L a set of literals in the $WFM(\Pi)$ (resp. $XSM(\Pi)$).

Acknowledgements

We thank ESPRIT BRA COMPULOG (no. 3012), INIC, JNICT and Gabinete de Filosofia do Conhecimento.

References

[Fitting, 1985] M. Fitting. A Kripke-Kleene semantics for logic programs. *Journal of Logic Programming*, 2(4):295–312, 1985.

[Gelfond and Lifschitz, 1988] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *International Conference on Logic Programming*, 1988.

[Pereira *et al.*, 1990] Luís M. Pereira, Joaquim N. Aparício, and José J. Alferes. A derivation procedure for extended stable models. Technical report, AI Centre/Uninova, 1990.

[Pereira *et al.*, 1991a] L. M. Pereira, J. N. Aparício, and J. J. Alferes. Counterfactual reasoning based on revising assumptions. Technical report, AI Centre/Uninova, 1991.

[Pereira *et al.*, 1991b] Luís M. Pereira, José J. Alferes, and Joaquim N. Aparício. Contradiction removal within Well Founded Semantics. In *First Logic Programming and Non-Monotonic Reasoning Workshop*. MIT Press, 1991.

[Pereira *et al.*, 1991c] Luís M. Pereira, Joaquim N. Aparício, and José J. Alferes. Hypothetical reasoning with well founded semantics. In *Third Scandinavian Conference on Artificial Intelligence*. IOS, 1991.

[Pereira *et al.*, 1991d] Luís M. Pereira, Joaquim N. Aparício, and José J. Alferes. Nonmonotonic reasoning with well founded semantics. In *International Conference on Logic Programming*. MIT Press, 1991.

[Przymusinska and Przymusinski, 1990] H. Przymusinska and T. Przymusinski. *Semantic Issues in Deductive Databases and Logic Programs*. Formal Techniques in Artificial Intelligence. North Holland, 1990.

[Przymusinski, 1989] T. C. Przymusinski. Every logic program has a natural stratification and an iterated fixed point model. In *Eight Symposium on Principles of Database Systems*, pages 11–21. ACM SIGACT-SIGMOD, 1989.

[Przymusinski, 1990] T. Przymusinski. Extended stable semantics for normal and disjunctive programs. In *International Conference on Logic Programming*, pages 459–477, 1990.

[Van Emden and Kowalski, 1976] A. Van Emden and R. Kowalski. The semantics of predicate logic as a programming language. *Journal of the ACM*, 23(4):733–742, 1976.

[Van Gelder *et al.*, 1990] A. Van Gelder, K. A. Ross, and J. S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 1990.