SLX – A top-down derivation procedure for programs with explicit negation

José Júlio Alferes, Carlos Viegas Damásio, and Luís Moniz Pereira
CRIA, Uninova and DCS, U. Nova de Lisboa
2825 Monte da Caparica, Portugal
{jja|cd|lmp}@fct.unl.pt

Abstract

In this paper we define a sound and (theoretically) complete top-down derivation procedure for a well-founded semantics of logic programs extended with explicit negation (*WFSX*). By its nature, it is amenable to a simple interpreter implementation in Prolog, and readily allows pre-processing into Prolog, showing promise as an efficient basis for further development.

The derivation method is directly inspired on the semantic AND-tree characterization of *WFSX* in our paper [2], but does not require knowledge of it. In fact, that work can be seen as a preliminary step towards the definition of the procedure presented here.

Because of its properties, which are paramount for top-down query evaluation and other approaches do not fully enjoy, *WFSX* is a natural candidate to being the semantics of choice for logic programs extended with explicit negation. Moreover, *WFSX* is sound wrt to the answer-sets semantics, and it is a better aproximation to answer-sets than simply using the well-founded semantics of normal programs, plus a renaming of explicitly negated literals. Thus, our top-down procedure can be used as a sound one for answer-sets, that provides less incompleteness than others.

Since *WFSX* coincides with the well-founded semantics on normal programs, our method is applicable to it and, for ground programs, compares favourably with previous approaches [2].

1 Introduction

The evolution of Logic Programming semantics has included the introduction of an explicit form of negation, beside the older implicit (or default) negation typical of Logic Programming, cf. [10, 13, 21]. The widespread use of such extended programs requires the definition of a correct top-down querying mechanism, as for Prolog. The purpose of this paper is to present one such SLDNF-like derivation procedure, SLX, for programs with explicit negation under well founded semantics (*WFSX*) [13], and prove its soundness and completeness. (Its soundness wrt the answer-sets semantics is also shown.)

The derivation method is directly inspired on the semantic AND-tree characterization of *WFSX* in our paper [2], but does not require knowledge of it. In fact, that work can be seen as a preliminary step towards the definition of the procedure presented here, in which there was no concern with the construction of the trees (they are assumed to exist). The attribution of failure and success was made "a posteriori" and assuming all the required trees simultaneoulsy present. Here not only do we define how the derivations are constructed in a top-down way, using an arbitrary selection rule, but also do we attribute success and failure of literals incrementally as the derivation develops. The characterization in [2] is a declarative semantics, and a first step towards the present one, which is procedural.

Our choice of *WFSX* as the base semantics is justified by the structural properties it enjoys, which are paramount for top-down query evaluation. Indeed, because of its properties, which other approaches do not fully enjoy, *WFSX* is a natural candidate to being the semantics of choice for logic programs extended with explicit negation. Namely, it exhibits the structural properties: well-foundedness, cumulativity, rationality, relevance, partial evaluation. By well-foundedness we mean that it can be characterized (without recourse to three-valued logic) by the least fixpoint of two iterative operators (cf. below), which justify our top-down procedure. By cumulativity [7], we refer to the efficiency related ability of using lemmas, i.e. the addition of lemmas does not change the semantics of a program. By rationality [7], we refer to the ability to add the negation of a non-provable conclusion without changing the semantics; this is important for efficient default reasoning. By relevance [8], we mean that the top-down evaluation of a literal's truth-value is made possible because it requires only the call-graph below it. By partial evaluation [8] we mean that the semantics of a partially evaluated program keeps to that of the original[1]. Additionally, it is amenable to both top-down and bottom-up procedures, and its complexity for finite DATALOG programs is polynomial. These and other properties of *WFSX* are detailed and proven in [1].

As proven in [1], *WFSX* is sound wrt to the answer-sets semantics of [10], and it is a better aproximation to answer-sets than simply using WFS plus the renaming of $\neg$-literals (cf. example in [2]). Thus, our top-down method can be used as a sound one for answer-sets, that provides less incompleteness than others. Since *WFSX* coincides with WFS on normal programs, our method is applicable to it and, for ground programs, compares favourably with previous approaches [2].

To the best of our knowledge paper [19] is the only in the literature that addresses the topic of proof procedures for extended logic programs. The author uses the notion of conservative derivability [22] as the proof-theoretic semantics for extended programs. The paper provides a program transformation from such programs to normal ones. Then it is proved that Kunen semantics [11] applied to the transformed program is sound and complete wrt conservative derivability. This approach has several problems mainly motivated by the interpretation of default negation as finite failure as rec-

[1]Stable model based approaches, such as answer-sets, enjoy neither cumulativity, nor rationality, nor relevance.

ognized by the author. For instance, in the program $\{a \leftarrow a\}$ the literal $\neg a$ is false but a is undefined, contrary to the results obtained by answer sets and *WFSX* where *not* a is true. As a final remark, conservative derivability is not defined for programs with functor symbols. Therefore the approach is only applicable to extended programs without functor symbols. *WFSX* solves all these problems properly.

The paper commences with a *WFSX* review, followed by the definition of SLX derivation, and continues with proofs of its soundness and (theoretical) completeness. The paper concludes with discussion.

2 *WFSX* overview

Here we recall the language of logic programs extended with explicit negation, or extended logic programs for short, and briefly review the *WFSX* semantics [13]. An extended program is a (possibly infinite) set of ground 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 $(0 \leq i \leq n)$. 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[2]. *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 disjoint 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 must also be entailed"*. This requirement states that whenever some literal is explicitly false then it must be assumed false by default.

Here we present *WFSX* in a distinctly different manner with respect to its original definition. This presentation is based on the alternating fixpoints of Gelfond-Lifschitz Γ-like operators. The equivalence between both definitions is proven in [1]. For lack of space we do not present here the definition of Γ (see [10]). To impose the coherence requirement [3] we introduce:

Definition 2.1 (Seminormal version of a program) *The seminormal 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.*

[2]This designation has been used in the literature instead of the more operational *"negation as failure (to prove)"*.

[3][1] shows how the use of semi-normal programs in fact imposes coherence.

Below we use $\Gamma(S)$ to denote $\Gamma_P(S)$, and $\Gamma_s(S)$ to denote $\Gamma_{P_s}(S)$.

Definition 2.2 (Partial stable model) *An interpretation $T \cup not\ F$ is called a partial stable model of P iff $T = \Gamma\Gamma_s T$ and $F = \mathcal{H}(P) - \Gamma_s T$.*

Not all extended programs have partial stable models (e.g. $P = \{a; \quad \neg a\}$ has no partial stable models). Programs without partial stable models are said contradictory.

Theorem 2.1 (*WFSX* **semantics)** *Every noncontradictory program P has a least (wrt $\subseteq$) partial stable model, the well-founded model of P ($WFM(P)$).*

To obtain an iterative "bottom-up" definition for $WFM(P)$ we define the following transfinite sequence $\{I_\alpha\}$ (where δ is a limit ordinal):

$$I_0 = \{\}; \qquad I_{\alpha+1} = \Gamma\Gamma_s I_\alpha; \qquad I_\delta = \bigcup \{I_\alpha \mid \alpha < \delta\}$$

There exist a smallest ordinal λ for the sequence above, such that I_λ is the smallest fixpoint of $\Gamma\Gamma_s$. Then $WFM(P) = I_\lambda \cup not\ (\mathcal{H}(P) - \Gamma_s I_\lambda)$.

If the program does not contain explicit negation then WFSX *coincides with the well-founded semantics of [9].*

In this constructive definition, literals obtained after an application of $\Gamma\Gamma_s$ (i.e. in some I_α) are true in $WFM(P)$, and literals not obtained after an application of Γ_s (i.e. not in $\Gamma_s I_\alpha$, for some α) are false by default in $WFM(P)$.

Note that, like the alternating fixpoint definition of WFS [20], this definition of *WFSX* also relies on the application of two anti-monotonic operators. However, unlike the definition of WFS, these operator are distinct. As we shall see, this points to the definition of two different kinds of derivation procedures, one reflecting applications of Γ and proving verity of objective literals, and the other reflecting applications of Γ_s and proving nonfalsity of objective literals. The two become meshed when the proof of verity of *not* L is translated into the failure to prove the nonfalsity of L.

The theorem below relates *WFSX* with the answer-sets semantics of [10], and its proof can be found in [1]:

Theorem 2.2 (Soundness wrt Answer-sets) *Let P be an extended logic program with at least one answer-set. Then P is noncontradictory, i.e. has partial stable models. Moreover, for any objective literal L, if $L \in WFM(P)$ then L belongs to all answer-sets of P; if not $L \in WFM(P)$ then L does not belong to any answer-set of P.*

3 SLX definition

In this section we define SLX (where X stands for eXtended programs, and SL stands for Selected Linear), a top-down derivation procedure for extended logic programs under *WFSX* [13]. Its correctness is proven in section 4.

The definition of SLX is directly inspired on the semantic tree characterization of [2], and relies on SLDNF-like definitions of derivation, refutation, and failure. For the sake of simplicity we present the definition only for ground (but possibly infinite) programs. However, its generalization for programs with variable is straightforward.

In order to informally motivate the definitions, we start with the simpler problem of programs without explicit negation. It is well known [4, 5, 6, 14, 18] that the main issues in the definition of top-down procedures for WFS are infinite positive recursion, and infinite recursion through negation by default. The former gives rise to the truth value false (so that, for some L involved in the recursion, there should exist a refutation for *not* L, and no refutation for L), and the latter to the truth value undefined (so that both L and *not* L should have no refutation).

Apart from these additional issues, we mainly follow the ideas of SLDNF [12] where refutations are derivations ending with the empty goal, derivations are constructed via resolution, and the negation as failure rule (stating that a selected *not* L is removed from the goal if L has no refutation and is the last goal of the derivation if L has one).

To solve the problem of positive recursion, as in SLS-resolution [16], we introduce a failure rule for not necessarily finite derivations.

Example 1 Let $P = \{p \leftarrow p\}$. The only derivation for p $(\leftarrow p, \leftarrow p, \ldots)$ is infinite. By stipulating that there is no refutation for $\leftarrow p$ we have that $\leftarrow not\ p, \leftarrow \Box$ is a refutation.

For recursion through negation by default we want both L and *not* L to have no refutations, which seems to violate the negation as failure rule. In fact that rule in SLDNF states that if there is no refutation for L, *not* L should succeed (i.e. removed from the goal), and if L has a refutation than *not* L should fail (i.e. be the last goal in the derivation). This is so because SLDNF relies on a two-valued semantics, and so failure of verity means falsity and vice-versa. In WFS, it being a three valued semantics, the same cannot apply. In fact, for WFS a failure of L simply means that L is not true, i.e. it can be false or undefined.

To solve this problem, others [15, 16, 17, 18] have defined derivation procedures that consider the extra status of "undefined" or "undetermined" assigned to goals. Literals involved in an infinite recursion through negation are assigned that status. As we shall see below, this approach does not generalize to extended programs.

Instead of considering an extra status we distinguish two kind of derivations: SLX-T-derivations that prove verity in the WFM, and SLX-TU-derivations (where TU stands for "true or undefined") that prove nonfalsity in the WFM. Now, for any L, the verity of *not* L succeeds iff there is no SLX-TU-refutation for L (i.e. L is false), and the nonfalsity of *not* L succeeds iff there is no SLX-T-refutation for L (i.e. L is not true).

Having these two kinds of derivations, it becomes easy to define the derived goal of a goal G, even with the selected literal L involved in recursion through negation by default. Indeed, since in this case, and according to WFS, L is to be undefined, it must be failed (i.e. the last goal in the derivation) if it occurs in a SLX-T-derivation, and refuted (i.e. removed from the goal) if it occurs in a SLX-TU-derivation.

Example 2 Let $P = \{p \leftarrow \mathit{not}\ p\}$. In order to prove the verity of p we start by building a SLX-T-derivation for $\leftarrow p$. By resolving the goal with the program rule we obtain $\leftarrow p, \leftarrow \mathit{not}\ p$. Now, $\leftarrow \mathit{not}\ p$ is the last goal in the derivation if there is a SLX-TU-refutation for $\leftarrow p$, and is replaced by the empty goal otherwise.

So we start building a SLX-TU-derivation for $\leftarrow p$. Resolving the goal with the program rule we obtain $\leftarrow p, \leftarrow \mathit{not}\ p$, and so there is a recursion in $\mathit{not}\ p$ through negation by default. Thus, as we are in a derivation for "true or undefined", $\mathit{not}\ p$ is removed from the goal, and its derived goal is the empty goal. This gives a SLX-TU-refutation for $\leftarrow p$, which in turn determines $\leftarrow \mathit{not}\ p$ to be the last goal in the SLX-T-derivation above, making it a failed one.

Now we show how to generalize this procedure to deal with explicit negation in *WFSX*. In a lot of points the treatment of extended programs is akin to that of normal ones, where instead of atoms we refer to objective literals, namely because, as expected, objective literals are treated exactly like atoms are in WFS.

The main difference in the generalization to extended programs resides in the treatment of negation by default. In order to fulfill the coherence requirement there must be an additional way to refute a literal $\mathit{not}\ L$. In fact $\mathit{not}\ L$ is true if $\neg L$ is also true.

Example 3 Consider $P = \{a \leftarrow \mathit{not}\ b;\ b \leftarrow \mathit{not}\ a;\ \neg a \leftarrow\ \}$, whose WFM is $\{\neg a, b, \mathit{not}\ a, \mathit{not}\ \neg b\}$.

In order to prove the verity of b, one starts to build one SLX-T-derivation for goal $\leftarrow b$. By solving it with the only rule for b in P we obtain $\leftarrow b, \leftarrow \mathit{not}\ a$. Now, according to what was said above, one has to build SLX-TU-derivations for $\leftarrow a$.

Solving $\leftarrow a$ with the first program rule we obtain $\leftarrow a, \leftarrow \mathit{not}\ b$. Since this is a case of recursion through negation by default, and we are in a TU-derivation, the derived goal is $\leftarrow \Box$. Consequently there is no refutation for $\leftarrow b$. However, since $\neg a$ is true, by coherence $\leftarrow \mathit{not}\ a$ (and thus $\leftarrow b$ also) must be refuted, by the additional method of proof for default literals via coherence.

In SLX-T-derivations, in order to obtain coherence, we introduce an extra possibility of removing a $\mathit{not}\ L$ from a goal: "In a SLX-T-derivation, a selected literal *not* L is removed from a goal if there is no SLX-TU-refutation for L *or if there is one SLX-T-refutation for* $\neg L$".

Care must also be taken in nonfalsity derivations because the coherence requirement can override undefinedness (cf. [13]):

Example 4 Consider $P = \{a \leftarrow b;\ b \leftarrow not\ c;\ c \leftarrow not\ c;\ \neg b \leftarrow\ \}$, whose WFM is $\{\neg b, not\ a, not\ b, not\ \neg b, not\ \neg c\}$.

In trying to prove verity of *not* a one starts building SLX-TU-derivations for $\leftarrow a$ to check whether they fail. The only possible such derivation starts with $\leftarrow a, \leftarrow b, \leftarrow not\ c$. To determine the derived goal of *not* c one tries to find a SLX-T-refutation for $\leftarrow c$ which, as the reader can easily check, does not exist. Thus there is a SLX-TU-refutation for $\leftarrow a$ $(\leftarrow a, \leftarrow b, \leftarrow not\ c, \leftarrow \Box)$, so that there is no SLX-T-refutation for $\leftarrow not\ a$.

However, *not* a belongs to the WFM of P. Note that this problem occurs because there is a SLX-TU-refutation for $\leftarrow b$ which should not exist since b is false. Indeed the falsity of b in the WFM is imposed by the coherence principle, since $\neg b$ is true. Note that in the derivations above $\neg b$ is not even used. In fact, in the SLX-TU-derivation for $\leftarrow a$, the goal $\leftarrow b$ should be the last one, because $\neg b$ is true. In other words, the verity of $\neg L$ (stating that L is false) forbids the use of resolution with rules for L in SLX-derivations.

In SLX-TU-derivations coherence imposes the verification of a precondition for the application of resolution on objective literals. Namely: "In a SLX-TU-derivation resolution can only be applied to a selected objective literal L *if there is no SLX-T-refutation for* $\neg L$".

Remark 3.1 *Note that coherence intervenes differently in T and TU-derivations. This is a consequence of having two distinct anti-monotonic operators in the definition of the WFM. At this point it is easier to understand the claim we made before that the usage of an undefined status, instead of having two different kinds of derivation, hampers the generalization to extended programs. In fact, in normal programs one can simply replace the two kinds of derivations by the addition of an undefined status because the single difference between the two derivation types is in the success or failure of literals involved in infinite recursion through default negation. However, in extended programs there are other differences, namely in what regards the applications of the coherence principle, which make the distinction crucial.*

The formalization of the intuitions presented above yields the following definitions of derivations and refutations:

Definition 3.1 (SLX-T-derivation) *Let* P *be an extended program, and* R *an arbitrary but fixed computational rule. A SLX-T-derivation* $G_0, G_1, \ldots$ *for* G *in* P *via* R *is defined as follows:* $G_0 = G$. *Let* G_i *be* $\leftarrow L_1, \ldots, L_n$ *and suppose that* R *selects the literal* L_k $(1 \leq k \leq n)$. *Then:*

- *if* L_k *is an objective literal, and the input rule is* $L_k \leftarrow B_1, \ldots, B_m$, *the derived goal is* $\leftarrow L_1, \ldots, L_{k-1}, B_1, \ldots, B_m, L_{k+1}, \ldots L_n$.

- *if L_k is not A then, if there is a SLX-T-refutation for $\neg A$ in P or there is no SLX-TU-refutation for A in P, the derived goal is:*

$$\leftarrow L_1, \ldots, L_{k-1}, L_{k+1}, \ldots L_n$$

- *otherwise G_i is the last goal in the derivation.*

Definition 3.2 (SLX-TU-derivation) *Let P be an extended program, and R an arbitrary but fixed computational rule. A SLX-T-derivation $G_0, G_1, \ldots$ for G in P via R is defined as follows: $G_0 = G$. Let G_i be $\leftarrow L_1, \ldots, L_n$ and suppose that R selects the literal L_k $(1 \leq k \leq n)$. Then:*

- *if L_k is an objective literal then*
  - *if there exists a SLX-T-refutation for $\neg L_k$ then G_i is the last goal in the derivation.*
  - *otherwise, if the input rule is $L_k \leftarrow B_1, \ldots, B_m$ the derived goal is $\leftarrow L_1, \ldots, L_{k-1}, B_1, \ldots, B_m, L_{k+1}, \ldots L_n$*
  - *if there is no rule for L_k then G_i is the last goal in the derivation.*
- *if L_k is not A then:*
  - *if there is a SLX-T-refutation for $\leftarrow A$ in P then G_i is the last goal in the derivation.*
  - *if all SLX-T-derivations for $\leftarrow A$ are SLX-T-failures then the derived goal is $\leftarrow L_1, \ldots, L_{k-1}, L_{k+1}, \ldots L_n$.*
  - *due to infinite recursion through default negation, it might happen that the previous points are not enough to determine the derived goal. In such a case, by definition, the derived goal is also*

$$\leftarrow L_1, \ldots, L_{k-1}, L_{k+1}, \ldots L_n.$$

Definition 3.3 (SLX refutation and failure) *A SLX-T-refutation (resp. SLX-TU-refutation) for G in P is a finite SLX-T-derivation (resp. SLX-TU-derivation) which ends in the empty goal ($\leftarrow \Box$).*

A SLX-T-derivation (resp. SLX-TU-derivation) for G in P is a SLX-T-failure iff it is not a refutation, i.e. it is infinite or it ends with a goal other than the empty goal.

4 Correctness of SLX

In this section we prove the soundness, and theoretical completeness[4] of the SLX derivation procedure. To do so we assign ranks to derivations. The proofs of correctness essentially rely on two lemmas proven by transfinite induction on the rank of derivations. In order to trim the proof we begin by making some simplifications in the above definitions of derivations:

In definition 3.1 of SLX-T-derivation one possible way of removing a selected default literal *not A* from a goal is to find a SLX-T-refutation for $\leftarrow$

[4]In practice completeness cannot be achieved because in general the WFM is not computable [9]. However, in theory, and with the possible need of constructing more than ω derivations, completeness is obtained.

$\neg A$. However this case is redundant. Note that the other case for removing *not* A is when there is no SLX-TU-refutation for $\leftarrow A$. But definition 3.2 states that in a SLX-TU-derivation, if there is a SLX-T-refutation for the explicit complement of a selected objective literal then the goal is the last in the derivation. Thus, if there is a SLX-T-refutation for $\leftarrow \neg A$, the only SLX-TU-derivation for $\leftarrow A$ is this single goal and is a failure, and so, even when not considering the first possibility, *not* A is nevertheless removed from the goal. Thus, in definition 3.1 the case $L_k = not\ A$ can be simplified to: *if there is no SLX-TU-refutation for* A *in* P *then the derived goal is*

$$\leftarrow L_1, \ldots, L_{k-1}, L_{k+1}, \ldots L_n$$

Now let's look at the cases for a selected objective literal L_k in definition 3.2. Clearly the first one corresponds to introducing *not* $\neg L_k$ in the derived goal. This is so because if there is a SLX-T-refutation for $\leftarrow \neg L$ the derivation will become a failure (and this is equivalent to the first case), and if there is no such refutation it is simply removed (and this is equivalent to the second case). Consequently, in definition 3.2 we remove the first case for a selected objective literal, keep the third, and modify the second to: *if the input rule is* $L_k \leftarrow B_1, \ldots, B_m$ *the derived goal is*

$$\leftarrow L_1, \ldots, L_{k-1}, not\ \neg L_k, B_1, \ldots, B_m, L_{k+1}, \ldots L_n$$

Now we assign ranks to these simplified derivations. As the proofs shall show, we do not need to assign a rank neither to SLX-T-failures nor to SLX-TU-refutations. These do not contribute towards proving literals that belong to the WFM[5].

Intuitively, the rank of a SLX-T-refutation reflects the depth of "calls" of subsidiary derivations that are considered in the refutation. Its definition, below, can be seen as first assigning to each literal removed from a goal an associated rank. When removing an objective literal no subsidiary derivation is considered, and so the rank is not affected. The empty goal has rank 0. When removing a default literal, the depth of subsidiary derivations that has to be considered is the maximum (more precisely, the least upper bound for the infinite case) of the depth of all SLX-TU-failures[6]. The depth needed for finally removing all literals from a goal is the maximum of the ranks associated with each of the literals in the goal.

Definition 4.1 (Rank of a SLX-T-refutation) *The rank of a SLX-T-refutation is the rank of its first goal. Ranks of goals in the refutation are:*

- *The rank of* $\leftarrow \Box$ *is 0.*
- *Let* G_i *be a goal in a refutation whose next selected literal is objective. The rank of* G_i *is the rank of* G_{i+1}.
- *Let* G_i *be a goal in a refutation whose next selected literal is a default one, not* L, *and let* α *be the least ordinal upper bound (i.e. maximum in the finite case) of the ranks of the SLX-TU-failures for* $\leftarrow L$[7]. *The*

[5]This is tantamount to having no need to assign a rank to indetermined nodes in [18].

[6]Note that for removing a default literal all SLX-TU-failures must be considered. This is the reason behind "maximum".

[7]Since we are in a SLX-T-refutation, all SLX-TU-derivations for $\leftarrow L$ are failures.

rank of G_i is the maximum of α and the rank of G_{i+1}.

Ranks of SLX-TU-failures reflect the depth of "calls" that is needed to fail the subsidiary derivations. Note that the failure of a derivation is uniquely determined by the last goal in it, and more precisely by its selected literal. If that literal is objective then no subsidiary derivation is needed to fail it, and thus its rank is 0. For failing a default literal *not* L one has to find a SLX-T-refutation for $\leftarrow L$. Several might exist, but it is enough to consider the one with minimum depth. Moreover, in this case one has to increment the rank, since the default literal *not* L was failed, and caused an extra "call". Note that, for SLX-T-refutations this increment is not considered. The issue of incrementing the rank only for one kind of derivations is tantamount to that of considering the increment of levels of I_is in the sequence for constructing the WFM only after the application of the two operators, Γ and Γ_s.

Definition 4.2 (Rank of a SLX-TU-failure) *An infinite SLX-TU-failure has rank 0. The rank of a finite SLX-TU-failure is the rank of its last goal. Let G_n be the last goal of the derivation, and L_k be its selected literal:*

- *if L_k is an objective literal then the rank is 0.*
- *if L_k is a default literal, not A, then the rank is $\alpha + 1$, where α is the minimum of the ranks of all SLX-T-refutations for $\leftarrow A$.*

The following lemma is used in the proofs of correctness. This lemma relates the existence of sequences where some default literals are removed to the Γ operator by which some default literals are removed from the body of rules:

Lemma 4.1 *Let I be an interpretation, and let $(\leftarrow L), G_1, \ldots$ be a sequence of goals constructed as per definition 3.2 (resp. definition 3.1), except that selected default literals not L_k such that $L_k \notin I$ are immediately removed from goals. Then: $L \in \Gamma_s I$ (resp. $L \in \Gamma I$) iff the sequence is finite and ends with the empty goal.*

Proof (sketch): Here we omit the proof for $L \in \Gamma I$ with definition 3.1, which is similar. If $L \in \Gamma_s I$ then, as per the definition of Γ_s, there must exist a finite set of rules in $\frac{P_s}{I}$ such that L belongs to its least model. According to the definition of $\frac{P_s}{I}$ and of semi-normal program, there is a finite set of rules in P such that for each default literal *not* L in their bodies $L \notin I$, and for each such rule with head H, $\neg H \notin I$. Let P^* be the subset of P formed by those rules. The only default literals to be considered by definition 3.2 will be those in the bodies, plus the default negations of $\neg$-complements of the heads of rules used in the derivation. So, given the completeness of SL-resolution[8] [12], and the fact that all these introduced literals are not in I (as shown above), a sequence of goals considering only the rules in the finite P^* exists and ends in $\leftarrow \square$. Thus the least model of $\frac{P^*_s}{I}$ contains L.

[8]For definite programs both T and TU derivations reduce to SL-derivation.

Lemma 4.2 *Let P be a noncontradictory extended logic program, L an objective literal, and $\{I_\alpha\}$ be the sequence constructed for the WFM of P, as per theorem 2.1. In that case:*

1. *if there is a SLX-T-refutation for $\leftarrow L$ in P with rank $< i$ then $L \in I_i$.*
2. *if all SLX-TU-derivations for $\leftarrow L$ in P are failures with rank $\leq i$ then $L \notin \Gamma_s I_i$.*

Proof: For point 1, by transfinite induction on i(for point 2 the proof is similar, and omitted here for brevity):

i is a limit ordinal δ: The case where $\delta = 0$ is trivial. For $\delta \neq 0$, for point 1, assume that there is a SLX-T-refutation for $\leftarrow L$ with rank $< \delta$. Thus, there is a $\alpha < \delta$ for which such a refutation exists with rank $< \alpha$. Then, $\exists_{\alpha<\delta} L \in I_\alpha$. Thus, $L \in \bigcup_{\alpha<\delta} I_\alpha$, i.e. $L \in I_\delta$.

Induction step: If there is a SLX-T-refutation for $\leftarrow L$ with rank $< i+1$ then, by definition of ranks for these refutations, all subsidiary derivations for default literals *not* L_j in the refutation are failed and of rank $< i+1$ (and thus $\leq i$) and are simply removed. So, given point 2, $\forall j, L_j \notin \Gamma_s I_i$. From lemma 4.1, by tacking there the interpretation $I = \Gamma_s I_i$, and by removing all *not* L_j literals, it follows that $L \in \Gamma\Gamma_s I_i$, i.e. $L \in I_{i+1}$.

Theorem 4.3 (Soundness of SLX) *Let P be a noncontradictory extended logic program, L an arbitrary literal from P. If there is an SLX-T-refutation for $\leftarrow L$ in P then $L \in WFM(P)$.*

Proof: If L is an objective literal, then the result follows immediately from lemma 4.2, and the monotonicity of $\Gamma\Gamma_s$.

Let $L = not\ A$. If there is a SLX-T-refutation for $\leftarrow not\ A$ with rank i then, by definition of SLX-T-refutation, all SLX-TU-derivations for $\leftarrow A$ are failures of rank $\leq i$. By point 2 of lemma 4.2, $A \notin \Gamma_s I_i$.

Let M be the least fixpoint of $\Gamma\Gamma_s$. Given that $\Gamma\Gamma_s$ is monotonic, $I_i \subseteq M$, i.e. for any objective literal A, $A \in I_i \Rightarrow A \in M$. By antimonotonicity of Γ_s, $A \in \Gamma_s M \Rightarrow A \in \Gamma_s I_i$. Thus, since $A \notin \Gamma_s I_i$, $A \notin \Gamma_s M$ i.e., by definition of the WFM, $not\ A \in WFM(P)$.

Given the soundness of *WFSX* wrt the answer-sets semantics (theorem 2.2) and the soundness of SLX wrt *WFSX*, it follows easily that SLX can be used as a sound derivation procedure for the answer-sets semantics.

Corollary 4.1 (Soundness wrt answer-sets) *Let P be an extended logic program with at least one answer-set, and L an arbitrary objective literal from P. If there is an SLX-T-refutation for $\leftarrow L$ in P then L belongs to all answer-sets of P. If there is an SLX-T-refutation for $\leftarrow$ not L in P then there is no answer-set of P with L.*

Now we prove theoretical completeness of SLX. To do so we begin by presenting a lemma that, like lemma 4.1 (and with a similar proof), relates sequences with the Γ operator. Then we prove completeness for objective literals by transfinite induction on the ranks for a particular class of selection rules. Finally we lift this restriction, and prove completeness also for default literals.

Lemma 4.4 *Let I be an interpretation, and L an objective literal. If $L \notin \Gamma_s I$ (resp. $L \notin \Gamma I$) then each possible sequence of goals starting with $\leftarrow L$ and constructed as per definition 3.2 (resp. definition 3.1), except that selected default literals not L_k such that $L_k \notin I$ are immediately removed from goals, is either: infinite; ends with a goal where the selected literal is objective; ends with a goal where the selected literal is not A and $A \in I$.*

Lemma 4.5 *Let P be an extended logic program, L an objective literal, and $\{I_\alpha\}$ be the sequence constructed for the WFM of P. Then, there exists a selection rule R such that:*

1. *if $L \in I_i$ then there is a SLX-T-refutation for $\leftarrow L$ in P with rank $< i$.*
2. *if $L \notin \Gamma_s I_i$ then all SLX-TU-derivations for $\leftarrow L$ in P are failures with rank $\leq i$.*

Proof: Let R be a selection rule that begins by selecting all objective literals, and then default ones subject to that it selects a *not* L before a *not* L' if there is a j in the sequence of the $\{I_\alpha\}$ such that $L \notin \Gamma_s I_j$ and $L' \in \Gamma_s I_j$.

By transfinite induction on i:

i is a limit ordinal δ: The case where $\delta = 0$ is trivial. For point 1 and $\delta \neq 0$, the proof is similar to the one presented in lemma 4.2 when $i = \delta$.

For point 2 and $\delta \neq 0$, assume that $L \notin \Gamma_s I_\delta$. By lemma 4.4, making the I in that lemma equal to I_δ, each SLX-TU-derivation for $\leftarrow L$ is either: infinite, and in this case a failure of rank 0; ends with a goal where the selected literal is objective, i.e. a failure of rank 0; ends with a goal where the selected literal is *not* A and $A \in I_\delta$. In this case, and given that point 1 is already proven for $i = \delta$, there is a SLX-T-refutation for $\leftarrow A$ with rank $< \alpha$ such that $\alpha < \delta$. Thus, and according to the definition of ranks, the rank of this derivation is $\leq \delta$.

Note that, by considering the special selection rule R in the sequences mentioned in lemma 4.4, these become indeed equal to derivations, where the *not* L_k such that $L_k \notin I_\delta$ are never selected.

Induction step: Assume points 1 and 2 of the lemma hold. We begin by proving that point 1 also holds for $i + 1$.

Assume that $L \in \Gamma\Gamma_s I_i$. By lemma 4.1, there exists a sequence ending with the empty goal, constructed as per definition 3.1, except that selected default literals *not* L_k such that $L_k \notin \Gamma_s I_i$ are immediately removed from goals. By point 2, for any L_k, all SLX-TU-derivations for $\leftarrow L_k$ are failures with rank $\leq i$. Therefore the sequence is a refutation. Moreover its rank is $\leq i$ and thus also $< i$. This proves point 1.

Now we prove that point 2 also holds for $i + 1$. Assume that $L \notin \Gamma_s I_{i+1}$. By lemma 4.4, considering the I in that lemma equal to I_{i+1}, each SLX-TU-derivation for $\leftarrow L$ is either: infinite, and in this case a failure of rank 0;

ends with a goal where the selected literal is objective, i.e. a failure of rank 0; ends with a goal where the selected literal is *not* A and $A \in I_{i+1}$. In this case, and given that point 1 is already proven, there is a SLX-T-refutation for $\leftarrow A$ with rank $< i+1$. Thus, and according to the definition of ranks, the rank of this derivation is $< i+2$, i.e. $\leq i+1$.

The argument for saying that the sequences of lemma 4.4 are derivation is similar to the one used above for limit ordinals.

Note that, in the proof of point 1 above, we never use the special selection rule R. Thus, for SLX-T-derivations an arbitrary selection rule can be used.

Moreover, in point 2, the only usage of R is to guarantee that the rank of all SLX-TU-failures is indeed $\leq i$. This is needed for proving the lemma by induction. However, it is clear that if by using R all SLX-TU-derivations are failures, although with a possibly greater rank, the same happens with an arbitrary selection rule[9]. This is why there is no need to consider the special selection rule in the theorem below.

Theorem 4.6 (Theoretical completeness of SLX) *Let P be a noncontradictory extended program, and L an arbitrary literal from P. If $L \in WFM(P)$ then there exists a SLX-T-refutation for $\leftarrow L$ in P.*

Proof: If L is an objective literal the proof follows from lemma 4.5.

Let $L = not\ A$. By definition of WFM, there exists an ordinal λ such that I_λ is the least fixpoint of $\Gamma\Gamma_s$. Thus, again by definition of WFM, $A \notin \Gamma_s I_\lambda$, and by point 2 of lemma 4.5 all SLX-TU-derivations for $\leftarrow A$ in P are failures. Consequently, the SLX-T-derivation consisting of the single goal $\leftarrow$ *not* A is a refutation.

This theorem requires one to know "a priori" whether the program is contradictory. However this is not problematic since SLX can detect contradictions:

Theorem 4.7 (Contradictory programs) *If P is contradictory, there exists a $L \in \mathcal{H}$ for which there are SLX-T-refutations for both $\leftarrow L$ and $\leftarrow \neg L$.*

5 Discussion

Although sound and complete, the method described in the previous section is not effective (even for finite ground programs). In fact, and because it furnishes no mechanism for detecting loops, termination is not guaranteed. To ensure the latter (at least) for finite ground programs, we've studied in [2] rules that prune the AND-trees search space, and eliminate both cyclic positive recursion and cyclic recursion through negation by default.

Now, for SLX to detect both kinds of cyclic recursions we need to use two types of ancestors: *local ancestors* are assigned to all literals appearing

[9]Literals involved in infinite recursion through negation do not give rise to SLX-TU-failures.

in derivations, and are used for detecting cyclic positive recursion; *global ancestors* are assigned to whole derivations, and are used to detect cyclic negative recursion. We've shown that two of the pruning rules there are necessary and sufficient to guarantee termination for all finite (and ground) extended logic programs. More sound pruning rules exist and can be used by a particular implementation to improve efficiency, thereby reducing the size and number of derivations searched. Due to its SLDNF-resemblance, it has been rather easy to implement a pre-processor that compiles[10] *WFSX* programs into Prolog.

A straightforward generalization method for nonground programs faces two problems: first, as shown in [3], the loop detection methods do not guarantee termination in the nonground case, even for term-bounded programs; second, the procedure flounders on nonground default literals. Being cumulative, *WFSX* is amenable to the use of tabulation methods to guarantee termination for nonground term-bounded programs. With the introduction of constructive negation techniques in SLX we hope to abate the floundering problems.

Another item of discussion is whether there is some program transformation, from extended programs into normal programs, such that available top-down procedures for WFS could be then applied to obtain an equivalent semantics to that of *WFSX* (where if the program is contradictory this must be detected). In fact there is one such transformation (lack of space prevents us to present it here). However, the resulting programs has a specific form, which makes any general WFS procedure less than optimal because such specificity is not exploited. In contradistinction, our SLX procedure takes into account the *WFSX* particulars to achieve better performance, namely certain steps (already mentioned) which are required in one of the two types of derivation are not in the other, and vice-versa.

Additionally, in our paper [2] we contrast our approach to other WFS top-down procedures, in the case of normal programs (where no transformation is needed), and find it compares favourably.

Acknowledgements

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

References

[1] J. J. Alferes. *Semantics of Logic Programs with Explicit Negation.* PhD thesis, Universidade Nova de Lisboa, 1993.

[2] J. J. Alferes, C. V. Damásio, and L. M. Pereira. Top-down query evaluation for well-founded semantics with explicit negation. In *ECAI'94*. Morgan Kaufmann, 1994.

[10] This implementation is available on request.

[3] K. Apt, R. Bol, and J. Klop. On the safe termination of Prolog programs. In *Proc. ICLP'89*. MIT Press, 1989.

[4] R. Bol and L. Degerstedt. Tabulated resolution for well founded semantics. In *ILPS'93*. MIT Press, 1993.

[5] W. Chen and D. S. Warren. A goal–oriented approach to computing well–founded semantics. In *IJCSLP'92*. MIT Press, 1992.

[6] W. Chen and D. S. Warren. Query evaluation under the well founded semantics. In *PODS'93*, 1993.

[7] J. Dix. Classifying semantics of logic programs. In *1st LP & NMR*. MIT Press, 1991.

[8] J. Dix. A framework for representing and characterizing semantics of logic programs. In *3rd KR*. Morgan Kaufmann, 1992.

[9] A. Van Gelder, K. A. Ross, and J. S. Schlipf. The well-founded semantics for general logic programs. *J. of the ACM*, 38(3), 1991.

[10] M. Gelfond and V. Lifschitz. Logic programs with classical negation. In *ICLP'90*. MIT Press, 1990.

[11] K. Kunen. Negation in logic programming. *Journal of LP*, 4, 1987.

[12] J. Lloyd. *Foundations of Logic Programming*. Springer–Verlag, 1987.

[13] L. M. Pereira and J. J. Alferes. Well founded semantics for logic programs with explicit negation. In *ECAI'92*. Morgan Kaufmann, 1992.

[14] L. M. Pereira, J. N. Aparício, and J. J. Alferes. A derivation procedure for extended stable models. In *IJCAI*. Morgan Kaufmann, 1991.

[15] H. Przymusinska, T. C. Przymusinski, and H. Seki. Soundness and completeness of partial deductions for well–founded semantics. In *Int. Conf. on Automated Reasoning*. LNAI 624, 1992.

[16] T. Przymusinski. Every logic program has a natural stratification and an iterated fixed point model. In *8th Symp. on Principles of Database Systems*. ACM SIGACT-SIGMOD, 1989.

[17] T. Przymusinski and D.S. Warren. Well–founded semantics: Theory and implementation. Draft, 1992.

[18] K. A. Ross. A procedural semantics for well-founded negation in logic programs. *Journal of LP*, 13, 1992.

[19] F. Teusink. A proof procedure for extended logic programs. In *ILPS'93*. MIT Press, 1993.

[20] A. van Gelder. The alternating fixpoint of logic programs with negation. In *Symp. on Principles of Database Systems*. ACM SIGACT-SIGMOD, 1989.

[21] G. Wagner. Logic programming with strong negation and innexact predicates. *J. of Logic and Computation*, 1(6), 1991.

[22] G. Wagner. Neutralization and preeemption in extended logic programs. Technical report, Freien Universitat Berlin, 1993.