# Approximation of the Well-Founded Semantics for Normal Logic Programs using Abstract Interpretation

Roberta Gori, Ernesto Lastres, René Moreno, Fausto Spoto

**Abstract**

The well-founded semantics for normal logic programs is defined in a constructive way through the use of a fixpoint operator and in general is not computable. We propose an approach for approximating through abstract interpretation the concrete behavior of normal logic programs described by the well-founded semantics. We provide also an abstract well-founded semantics scheme, based on an abstract immediate consequences operator and apply this scheme to obtain an approximated semantics on the *depth(k)* domain.

*Keywords: Logic Programming, Abstract Interpretation, Well-Founded Semantics.*

## 1 Introduction

Abstract interpretation is a theory developed to reason about the abstraction relation between two different semantics. In general it is used to design abstract semantics useful for static analysis, but it can also be used to approximate the concrete semantics of programs. This last aspect becomes relevant when the concrete semantics is not finitely computable, which is the case of the well-founded semantics for normal logic programs.

The well-founded semantics is one of the well known semantics for normal logic programs. It is defined in a constructive way through the use of a fixpoint operator, allowing a direct approximation through abstract interpretation. Moreover, the well-founded semantics is applicable to every normal logic program, without any syntactical restriction.

This paper presents an approach for approximating through abstract interpretation the concrete behavior of normal logic programs described by the well-founded semantics. We provide an abstract well-founded semantics scheme, based on an abstract immediate consequences operator. If the abstract domain is finite we get the abstract fixpoint of this operator in a finite number of steps. We provide sufficient conditions on the abstract domain and the abstract operators which ensure that the fixpoint is correct w.r.t. the concrete well-founded semantics of the program, in the sense that it provides upward approximations of both, the positive and the negative facts of the semantics. We apply then our scheme to the *depth(k)* abstract domain.

The paper is organized as follows. In section 3 we recall the basic definitions of the well-founded semantics, in section 4 we define our abstraction scheme and we prove its correctness w.r.t the concrete semantics. Section 5 presents the application of our approximation scheme to the *depth(k)* domain and finally in 5.1 we show some examples of approximations.

Dipartimento di Informatica, Università di Pisa Corso Italia 40, 56125 Pisa, Italy. Phone : +39 (+50) 887279 E-mail: {gori,lastres,moreno,spoto}@di.unipi.it

# 2 Preliminaries

The reader is assumed to be familiar with standard logic programming definitions as presented in [1, 4].

In the paper we will consider the equational version of logic programs. With $c, p(\mathbf{x})$ we denote a constrained atom, by $U_H$ and $B_H$ we denote the Herbrand Universe and Base respectively. $\mathcal{P}$ denotes the set normal logic programs, $\mathcal{P}_{\mathcal{H}}$ denotes the set of ground normal logic programs. By $T_P$ we denote the usual immediate consequences operator for positive logic programs as defined in [4]. If $S$ is a set then $\mathcal{P}(S)$ denotes the powerset of $S$.

We assume also familiarity with the theory of abstract interpretation as presented in [2].

# 3 Well-Founded Semantics

The well-founded semantics (*wfs* ) [7] is based on the idea that a given program may not necessarely provide information on the truth or falsehood of every fact. Instead some facts may simply be indifferent to it, and the answer should be allowed to say that the truth value of those facts is unknown. The price is that the answer is no longer guaranteed to provide total information. A key point of this approach is that all normal programs can be considered without any syntactical restriction as in the case of stratified and locally stratified models. Another aspect of this approach is that it puts negative and positive facts on a more equal footing: both must be inferred.

To formalize this, we will introduce a 3-valued interpretation, which is a pair of Herbrand interpretations: the first one contains the positive facts and the second one contains the negative facts. The other facts are considered unknown. Hence we will write an interpretation as $I = \langle I^+, I^- \rangle$, where $I^+$ is the set of positive facts and $I^-$ is the set of negative facts. This set of interpretations is made into a cpo $\langle I, \leq \rangle$ defining $I \leq J$ iff $I^+ \subseteq J^+$ and $I^- \subseteq J^-$.

While many formulations of *wfs* have been given, we use here a presentation based on a fixpoint calculation: The interpretations are refined with an immediate consequence operator $W_P$ .
Given a program $P$ consider the program $P'$ formed by all ground instances of the clauses in $P$. Let $I$ be an interpretation; $W_P(I) = \left\langle M_{P'/_{I^+}}, \overline{M_{P'/_{I^-}}} \right\rangle$ , where $M_R$ stands for the least fixpoint of the Herbrand immediate consequence operator $T_R$ and $\overline{M_R}$ stands for $B_H \setminus M_R$ and

1. $P'/_{I^+}$ is computed in the following three steps:
   (a) Remove all clauses having at least a literal in their body which is false in $I$;
   (b) Remove all literals (from the body of the program clauses) which are true in $I$.
   (c) $P'/_{I^+}$ is obtained by removing all clauses having at least a negative literal in their body.
2. $P'/_{I^-}$ is computed in the following three steps:
   (a) Remove all clauses having at least a literal in their body which is false in $I$;

(b) Remove all literals (from the body of the program clauses) which are true in $I$.

(c) $P'/_{I^-}$ is obtained by removing all negative literals.

It was shown in [7] that the $W_P$ operator is monotone on $\langle I, \leq \rangle$. Hence the semantics of the program $P$ is defined as the least fixpoint of the $W_P$ operator. Since $W_P$ is not continuous the computation of its fixpoint could require more than $\omega$ iterations.

Note that this definition is based on three basic operations, namely

1. The operator $G : \mathcal{P} \rightarrow \mathcal{P}_{\mathcal{H}}$, which computes the set of ground instances of a program,
2. The operator $Comp : \mathcal{P}(U_H) \rightarrow \mathcal{P}(U_H)$, which computes the complement w.r.t. the Herbrand Base of a set of ground atoms,
3. The usual membership operator $\in$: $U_H \times \mathcal{P}(U_H) \rightarrow \{\mathbf{true}, \mathbf{false}\}$.

# 4 Abstract Semantics

In this section we show how an abstract *wfs* can be defined for an abstract program given a Galois-insertion $\langle \alpha, \gamma \rangle$ from the concrete domain $\langle I, \leq \rangle$ to an abstract domain $\langle I^\alpha, \leq^\alpha \rangle$. Actually this function $\alpha$ is induced by an abstraction function from the Herbrand Universe to an abstract Herbrand domain, which with abuse of notation we will call $\alpha$. Note that this abstraction function determines univocally the abstract functions on atoms and programs.

We have to specify the abstract counterparts of the three concrete operators described in the previous section. We define the following abstract (optimal) operators:

1. *Abstract optimal grounding*: $G^\alpha : \alpha(\mathcal{P}) \rightarrow \alpha(\mathcal{P}_{\mathcal{H}})$, as $G^\alpha(P_\alpha) = \alpha\left(G\left(\gamma(P_\alpha)\right)\right)$, where $P_\alpha$ is an abstract program.
2. *Abstract optimal complement*: $Comp_\alpha : \mathcal{P}(\alpha(U_H)) \rightarrow \mathcal{P}(\alpha(U_H))$ , as $Comp_\alpha(S_\alpha) = \bigcap \{S'_\alpha \mid Comp(g) \subseteq \gamma(S'_\alpha),\ \forall g.\alpha(g) \in S_\alpha\}$ which can be rewritten as $Comp_\alpha(S_\alpha) = \bigcap_{g \in \gamma(S_\alpha)} \alpha(Comp(g))$.
3. *Abstract membership*: $\in^\alpha$: $\alpha(U_H) \times \mathcal{P}(\alpha(U_H)) \rightarrow \{\mathbf{true}, \mathbf{false}\}$, as $\alpha(c,\ p(\mathbf{x})) \in^\alpha S_\alpha$ iff $\gamma\left(\alpha(c,\ p(\mathbf{x}))\right) \cap \gamma(S_\alpha) \neq \emptyset$

Given an abstract interpretation $I_\alpha = \langle I^+_\alpha, I^-_\alpha \rangle$, the abstract immediate consequences operator for an abstract program $P' = G^\alpha(\alpha(P))$ is defined as $W^\alpha_{\alpha(P)}(I_\alpha) = \left\langle M_{P'/_{I^+_\alpha}}, \overline{M_{P'/_{I^-_\alpha}}} \right\rangle$, where $M_R$ stands for the least fixpoint of the abstract immediate consequences operator, $\overline{M_R}$ stands for the complement of $M_R$ and

1. $P'/_{I^+_\alpha}$ is obtained through the following three steps:
   (a) Do nothing
   (b) Remove all literals which are true in $I_\alpha$, using the abstract membership operator.

(c) $P'/_{I_\alpha^+}$ is obtained by removing all clauses having at least a negative literal in their body,

2. $P'/_{I_\alpha^-}$ is obtained through the following three steps:

   (a) Remove all clauses having at least a literal in their body which is false in $I_\alpha$, using the abstract membership operator,

   (b) Do nothing

   (c) $P'/_{I_\alpha^-}$ is obtained by removing all negative literals.

Now we have to find sufficient conditions that guarantee the correctness of the abstract semantics w.r.t. the concrete one. Formally, we will prove that the following conditions will entail the correctness of the $W^\alpha_{\alpha(P)}$ operator w.r.t. $W_P$, namely $W_P(I) \subseteq \gamma(W^\alpha_{\alpha(P)}(\alpha(I)))$:

1. $\alpha\left(G(P)\right) = G^\alpha\left(\alpha(P)\right)$
2. $\alpha(Comp(S)) \subseteq Comp_\alpha(\alpha(S))$
3. $c,\ p(\mathbf{x}) \in S \Rightarrow \alpha(c,\ p(\mathbf{x})) \in^\alpha \alpha(S)$
4. The abstract immediate consequences operator for positive programs is correct w.r.t. the concrete one.

Actually we will prove a stronger property:

**Proposition 1** *Let $I$ be an interpretation and $I_\alpha$ be an abstract interpretation. Then we have, for every program* P $\alpha(I) \leq I_\alpha \Rightarrow \alpha\left(W_P(I)\right) \leq W^\alpha_{\alpha(P)}(I_\alpha)$

**Proof**:
The correctness condition 1 on the abstract grounding allows us to consider the abstract ground program over which $W^\alpha_{\alpha(P)}$ works, as the abstraction of the ground program over which $W_P$ works. We split the proof in two parts, the first one for the positive facts of the interpretation and the second one for the negative facts.

1. The concrete positive facts are obtained as the least fixpoint of the program obtained through steps 1.a, 1.b and 1.c of the definiton of $W_P$. In the definition of $W^\alpha_{\alpha(P)}$ the step 1.a is no more executed, hence this step removes some clauses only from the concrete program. Step 1.b is executed in both operators: Since we assume $I^+ \subseteq I^+_\alpha$ and by correctness condition 3 on the membership operator, literals are more often removed from the abstract program than from the concrete one. Step 1.c either removes from the abstract program a clause which is the abstraction of a concrete clause which was already removed in step 1.a of $W_P$, or removes a clause which is the abstraction of a concrete clause which gets removed in step 1.c of $W_P$. This is because step 1.b removes more literals from the abstract program than from the concrete one. From these considerations we conclude that the abstraction of the fixpoint of the concrete program computed in $W_P$ contains less facts than the fixpoint of the abstract program computed in $W^\alpha_{\alpha(P)}$, since we assume to use an abstract fixpoint operator on positive programs which is correct on w.r.t. the concrete fixpoint operator on positive programs.

2. The concrete negative facts are obtained as the complementation of the least fixpoint of the program obtained through steps 2.a, 2.b and 2.c of the definition of $W_P$. Since complementation is antimonotonic, we have to show that the abstraction of the fixpoint of this program contains all facts contained in the fixpoint of the abstract program computed in the definition of $W^{\alpha}_{\alpha(P)}$. Step 2.a removes a clause from the concrete program only if its abstraction is removed from the abstract program. This is because we assume $\alpha(I) \subseteq I_\alpha$ and by correctness condition 2. Step 2.b is executed only on the concrete program. Step 2.c removes a literal from the concrete program only if it removes the abstraction of the literal from the abstract clause which is the abstraction of the concrete clause the first literal belongs to. These considerations, together with the correctness condition 4, imply that the abstraction of the fixpoint of the concrete program obtained through the algorithm for $W_P$ contains the fixpoint of the abstract program obtained through the algorithm for $W^{\alpha}_{\alpha(P)}$.

□

**Corollary 2** *Let $I$ be an interpretation. Then we have, for every program $P$ ,*

$$\alpha\left(W_P(I)\right) \leq W^{\alpha}_{\alpha(P)}(\alpha(I))$$

# 5 Approximation using the $depth(k)$ domain

We show now how the concrete well-founded semantics can be approximated using the *depth(k)* domain [6]. The idea is to approximate the infinite ground success set by means of a *depth(k)* cut, i.e., by cutting terms which have a depth greater than $k$. Terms are cut by replacing each sub-term rooted at depth $k$ with a new variable taken from a set $\widetilde{V}$ disjoint from $V$ (the set of variables of the program). A *depth(k)* term represents all the terms obtained by instantiating the variables of $\widetilde{V}$ with groundterms.

We have to define the abstraction functions $\alpha_k$ on terms and $\alpha^c_k$ on constraints.

- If $k = 0$ we define $\alpha_0(t) = Z$, where $Z \in \widetilde{V}$ is a new fresh variable
- If $k > 0$ we define
  - $\alpha_k(a) = a$, if $a$ is a constant.
  - $\alpha_k(X) = X$, if $X$ is a variable.
  - $\alpha_k(f(t_1, ..., t_n)) = f(\alpha_{k-1}(t_1), ..., \alpha_{k-1}(t_n))$, if $f$ is a functor.
  - $\alpha^c_k(\{X = t\}) = \{X = \alpha_k(t)\}$.
  - $\alpha^c_k(C_1 \cup C_2) = \alpha^c_k(C_1) \cup \alpha^c_k(C_2)$.

We show now how the abstract operations can be effectively computed for the *depth(k)* domain

The abstract grounding $G^{\alpha_k}$ of a program on *depth(k)* can be computed through the following algorithmic definition $G^{\alpha_k}_e$:

- If $k = 0$: $G^{\alpha_k}_e(t) = Z$ for all abstract terms $t$.

- If $k > 0$:
  - $G_e^{\alpha_k}(a) = a$
  - $G_e^{\alpha_k}(X) = U_H^k$, where $U_H^k$ is the abstraction of the Herbrand Universe.
  - $G_e^{\alpha_k}(f(t_1, ...t_n)) = \left\{ f(\widetilde{t_1}, ..., \widetilde{t_n}) \mid \widetilde{t_i} \in G_e^{\alpha_{k-1}}(t_i) \right\}$
  - $$G_e^{\alpha_k}(X = t^\alpha) = \{\{X = t_X^\alpha, Y = t_Y^\alpha, ...\} \mid X, Y, ... \in V, t_X^\alpha, t_Y^\alpha, ... \in \alpha_k(U_H) \text{ and } (t^\alpha [Y = t_Y^\alpha, ...])_k = t_X^\alpha\}$$
  - $G_e^{\alpha_k}(C_1 \cup C_2) = G_e^{\alpha_k}(C_1) \cap G_e^{\alpha_k}(C_2)$

The following lemma shows that the algorithmic definition for the abstract grounding agrees with the generic definition given in section 4. Moreover, it shows that the correctness condition for the abstract grounding holds in the *depth(k)* domain.

**Lemma 3** *The following two properties hold:*

- $G^{\alpha_k}(tc) = G_e^{\alpha_k}(tc)$ *for every term or constraint* $tc$.
- $\alpha_k(G(P)) = G^{\alpha_k}(\alpha_k(P))$ *for every program* $P$ .

If the Herbrand Universe is infinite it can be shown that the abstract complement can be effectively computed as $comp_\alpha(I_\alpha) = B_P^k \setminus \left\{ p(\widetilde{t}) \in I_\alpha \mid p(\widetilde{t}) \textit{ is not cut} \right\}$

Since the concretization of different abstract constrained atoms are disjoint, the abstract membership operator becomes the usual membership operator, which is effective because the set of abstract constrained atoms is finite.

Finally, correctness condition 4 holds, as shown in [5].

Note that the $depth(k)$ domain, for a given $k$ and a finite number of function symbols, is finite. Hence the abstract fixpoint is always reachable in a finite number of steps.

## 5.1 Examples

Consider the following program $P$ defining the predicates `odd`, which computes the set of odd natural numbers, `mult3` for computing the set of natural numbers which are multiple of three and `oddnotmult3` for computing the set of odd natural numbers which are not multiple of three.

```
odd(X) : −X = s(0).
odd(X) : −X = s(s(Y)), odd(Y).
mult3(X) : −X = s(s(s(0))).
mult3(X) : −X = s(s(s(Y))), mult3(Y).
oddnotmult3(X) : −X = Y, X = Z, odd(Y), ¬mult3(Z).
```

Consider now the abstraction $\alpha_k(P)$ for $k = 4$, which is equal to $P$. The grounding of $\alpha_k(P)$ yields:

$$
\begin{array}{l}
\mathtt{odd(X)} : -\mathtt{X} = \mathtt{s(0)}. \\
\mathtt{odd(X)} : -\mathtt{X} = \mathtt{s(s(0))}, \mathtt{Y} = \mathtt{0}, \mathtt{odd(Y)}. \\
\mathtt{odd(X)} : -\mathtt{X} = \mathtt{s(s(s(0)))}, \mathtt{Y} = \mathtt{s(0)}, \mathtt{odd(Y)}. \\
\mathtt{odd(X)} : -\mathtt{X} = \mathtt{s(s(s(s(K))))}, \mathtt{Y} = \mathtt{s(s(0))}, \mathtt{odd(Y)}. \\
\mathtt{odd(X)} : -\mathtt{X} = \mathtt{s(s(s(s(K))))}, \mathtt{Y} = \mathtt{s(s(s(0)))}, \mathtt{odd(Y)}. \\
\mathtt{odd(X)} : -\mathtt{X} = \mathtt{s(s(s(s(K))))}, \mathtt{Y} = \mathtt{s(s(s(s(K1))))}, \mathtt{odd(Y)}. \\
\mathtt{mult3(X)} : -\mathtt{X} = \mathtt{s(s(s(0)))}. \\
\mathtt{mult3(X)} : -\mathtt{X} = \mathtt{s(s(s(0)))}, \mathtt{Y} = \mathtt{0}, \mathtt{mult3(Y)}. \\
\mathtt{mult3(X)} : -\mathtt{X} = \mathtt{s(s(s(s(K))))}, \mathtt{Y} = \mathtt{s(0)}, \mathtt{mult3(Y)}. \\
\mathtt{mult3(X)} : -\mathtt{X} = \mathtt{s(s(s(s(K))))}, \mathtt{Y} = \mathtt{s(s(0))}, \mathtt{mult3(Y)}. \\
\mathtt{mult3(X)} : -\mathtt{X} = \mathtt{s(s(s(s(K))))}, \mathtt{Y} = \mathtt{s(s(s(0)))}, \mathtt{mult3(Y)}. \\
\mathtt{mult3(X)} : -\mathtt{X} = \mathtt{s(s(s(s(K))))}, \mathtt{Y} = \mathtt{s(s(s(s(K1))))}, \mathtt{mult3(Y)}.
\end{array}
$$

$$
\begin{array}{lll}
\mathtt{oddnotmult3(X)} : -\mathtt{X} = \mathtt{0}, \mathtt{Y} = \mathtt{0}, \mathtt{Z} = \mathtt{0}, \mathtt{odd(Y)}, \neg\mathtt{mult3(Z)}. & & \\
\mathtt{oddnotmult3(X)} & : - & \mathtt{X} = \mathtt{s(0)},\ \mathtt{Y} = \mathtt{s(0)},\ \mathtt{Z} = \mathtt{s(0)}, \\
 & & \mathtt{odd(Y)},\ \neg\mathtt{mult3(Z)}. \\
\mathtt{oddnotmult3(X)} & : - & \mathtt{X} = \mathtt{s(s(0))},\ \mathtt{Y} = \mathtt{s(s(0))},\ \mathtt{Z} = \mathtt{s(s(0))}, \\
 & & \mathtt{odd(Y)},\ \neg\mathtt{mult3(Z)}. \\
\mathtt{oddnotmult3(X)} & : - & \mathtt{X} = \mathtt{s(s(s(0)))},\ \mathtt{Y} = \mathtt{s(s(s(0)))},\ \mathtt{Z} = \mathtt{s(s(s(0)))}, \\
 & & \mathtt{odd(Y)},\ \neg\mathtt{mult3(Z)}. \\
\mathtt{oddnotmult3(X)} & : - & \mathtt{X} = \mathtt{s(s(s(K))))},\ \mathtt{Y} = \mathtt{s(s(s(K))))}, \\
 & & \mathtt{Z} = \mathtt{s(s(s(K))))}\ \mathtt{odd(Y)},\ \neg\mathtt{mult3(Z)}.
\end{array}
$$

The abstract fixpoint is

$$
\begin{array}{rcl}
I_2^+ & = & \{\mathtt{X} = \mathtt{s(0)}, \mathtt{odd(X)},\ \mathtt{X} = \mathtt{s(s(s(0)))}, \mathtt{odd(X)},\ \mathtt{X} = \mathtt{s(s(s(s(K))))}, \mathtt{odd(X)}, \\
 & & \mathtt{X} = \mathtt{s(s(s(0)))}, \mathtt{mult3(X)},\ \mathtt{X} = \mathtt{s(s(s(s(K))))}, \mathtt{mult3(X)}, \\
 & & \mathtt{X} = \mathtt{s(0)}, \mathtt{oddnotmult3(X)},\ \mathtt{X} = \mathtt{s(s(s(s(K))))}, \mathtt{oddnotmult3(X)}\}
\end{array}
$$

$$
\begin{array}{rcl}
I_2^- & = & \{\mathtt{X} = \mathtt{0}, \mathtt{odd(X)},\ \mathtt{X} = \mathtt{s(s(0))}, \mathtt{odd(X)},\ \mathtt{X} = \mathtt{s(s(s(s(K))))}, \mathtt{odd(X)}, \\
 & & \mathtt{X} = \mathtt{0}, \mathtt{mult3(X)},\ \mathtt{X} = \mathtt{s(0)}, \mathtt{mult3(X)}, \\
 & & \mathtt{X} = \mathtt{s(s(0))}, \mathtt{mult3(X)},\ \mathtt{X} = \mathtt{s(s(s(s(K))))}, \mathtt{mult3(X)}, \\
 & & \mathtt{X} = \mathtt{0}, \mathtt{oddnotmult3(X)},\ \mathtt{X} = \mathtt{s(s(0))}, \mathtt{oddnotmult3(X)}, \\
 & & \mathtt{X} = \mathtt{s(s(s(0)))}, \mathtt{oddnotmult3(X)}, \\
 & & \mathtt{X} = \mathtt{s(s(s(s(K))))}, \mathtt{oddnotmult3(X)}\}
\end{array}
$$

which is exactly the *depth*(4) approximation of the concrete semantics of the program. Note that in this example, as a consequenece of the abstract complement operator, the intersection between the positive and the negative atoms is not empty. Note also that there are not atoms which do not belong neither to the positive information nor to the negative information, therefore the well founded semantics coincide with the stable models semantics [3] of the program.

This is not the case of the next example, where some facts are left undefined in the fixpoint, i.e. they do not belong neither to the positive information nor to the negative information.

Consider the following program $P$, which is a generalization of the well-known barber's problem.

$$\begin{array}{l}\texttt{citizen(X)} : -\texttt{X} = \texttt{0}.\\ \texttt{citizen(X)} : -\texttt{X} = \texttt{s(Y)},\ \texttt{citizen(Y)}.\\ \texttt{shave(X,Y)} : -\texttt{X} = \texttt{s(s(0))},\ \texttt{citizen(Y)}, \neg\texttt{shave(Y,Y)}.\end{array}$$

The abstract fixpoint is

$$\begin{array}{rcl} I_2^+ & = & \{\texttt{X} = \texttt{0}, \texttt{citizen(X)},\ \texttt{X} = \texttt{s(0)}, \texttt{citizen(X)},\ \texttt{X} = \texttt{s(s(0))}, \texttt{citizen(X)},\\ & & \texttt{X} = \texttt{s(s(s(0)))}, \texttt{citizen(X)},\ \texttt{X} = \texttt{s(s(s(s(K))))}, \texttt{citizen(X)},\\ & & \texttt{X} = \texttt{s(s(0))}, \texttt{Y} = \texttt{0}, \texttt{shave(X,Y)},\ \texttt{X} = \texttt{s(s(0))}, \texttt{Y} = \texttt{s(0)}, \texttt{shave(X,Y)},\\ & & \texttt{X} = \texttt{s(s(0))}, \texttt{Y} = \texttt{s(s(s(0)))}, \texttt{shave(X,Y)},\\ & & \texttt{X} = \texttt{s(s(0))}, \texttt{Y} = \texttt{s(s(s(s(K))))}, \texttt{shave(X,Y)}\}\end{array}$$

$$\begin{array}{rcl} I_2^- & = & \{\texttt{X} = \texttt{s(s(s(s(K))))}, \texttt{citizen(X)},\ \texttt{X} = \texttt{0}, \texttt{Y} = \texttt{0}, \texttt{shave(X,Y)},\\ & & \texttt{X} = \texttt{0}, \texttt{Y} = \texttt{s(0)}, \texttt{shave(X,Y)},\ \texttt{X} = \texttt{0}, \texttt{Y} = \texttt{s(s(0))}, \texttt{shave(X,Y)},\\ & & \texttt{X} = \texttt{0}, \texttt{Y} = \texttt{s(s(s(0)))}, \texttt{shave(X,Y)},\ \texttt{X} = \texttt{0}, \texttt{Y} = \texttt{s(s(s(s(K))))}, \texttt{shave(X,Y)},\\ & & \texttt{X} = \texttt{s(0)}, \texttt{Y} = \texttt{0}, \texttt{shave(X,Y)},\ \texttt{X} = \texttt{s(0)}, \texttt{Y} = \texttt{s(0)}, \texttt{shave(X,Y)},\\ & & \texttt{X} = \texttt{s(0)}, \texttt{Y} = \texttt{s(s(0))}, \texttt{shave(X,Y)},\ \texttt{X} = \texttt{s(0)}, \texttt{Y} = \texttt{s(s(s(0)))}, \texttt{shave(X,Y)},\\ & & \texttt{X} = \texttt{s(0)}, \texttt{Y} = \texttt{s(s(s(s(K))))}, \texttt{shave(X,Y)},\\ & & \texttt{X} = \texttt{s(s(0))}, \texttt{Y} = \texttt{s(s(s(s(K))))}, \texttt{shave(X,Y)},\\ & & \texttt{X} = \texttt{s(s(s(0)))}, \texttt{Y} = \texttt{0}, \texttt{shave(X,Y)},\ \texttt{X} = \texttt{s(s(s(0)))}, \texttt{Y} = \texttt{s(0)}, \texttt{shave(X,Y)},\\ & & \texttt{X} = \texttt{s(s(s(0)))}, \texttt{Y} = \texttt{s(s(0))}, \texttt{shave(X,Y)},\\ & & \texttt{X} = \texttt{s(s(s(0)))}, \texttt{Y} = \texttt{s(s(s(0)))}, \texttt{shave(X,Y)},\\ & & \texttt{X} = \texttt{s(s(s(0)))}, \texttt{Y} = \texttt{s(s(s(s(K))))}, \texttt{shave(X,Y)},\\ & & \texttt{X} = \texttt{s(s(s(s(K))))}, \texttt{Y} = \texttt{0}, \texttt{shave(X,Y)},\ \texttt{X} = \texttt{s(s(s(s(K))))}, \texttt{Y} = \texttt{s(0)}, \texttt{shave(X,Y)},\\ & & \texttt{X} = \texttt{s(s(s(s(K))))}, \texttt{Y} = \texttt{s(s(0))}, \texttt{shave(X,Y)},\\ & & \texttt{X} = \texttt{s(s(s(s(K))))}, \texttt{Y} = \texttt{s(s(s(0)))}, \texttt{shave(X,Y)},\\ & & \texttt{X} = \texttt{s(s(s(s(K))))}, \texttt{Y} = \texttt{s(s(s(s(K))))}, \texttt{shave(X,Y)}\}\end{array}$$

Note that the fact $\texttt{X} = \texttt{s(s(0))}, \texttt{Y} = \texttt{s(s(0))}, \texttt{shave(X,Y)}$ does not belong neither to $I_2^-$ nor to $I_2^+$

# 6 Conclusions

We showed how the well-founded semantics for normal logic programs can be approximated using abstract interpretation theory. We presented an approximation based on the *depth(k)* domain, but it is worth noting that another sensitive approximation can be defined applying our scheme to a type domain.

This approach can be used as alternative or support to other methods for approximating non computable semantics as is the case of set based analysis.

# References

[1] K. R. Apt. Introduction to Logic Programming. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, pages 495–574. Elsevier and The MIT Press, 1990.

[2] P. Cousot and R. Cousot. Abstract Interpretation and Applications to Logic Programs. *Journal of Logic Programming*, 13(2 & 3):103–179, 1992.

[3] M. Gelfond and V. Lifschitz. The Stable Model Semantics for Logic Programs. In R. A. Kowalski and K. A. Bowen, editors, *Proc. Fifth Int'l Conf. on Logic Programming*, pages 1070–1079. The MIT Press, 1988.

[4] J. W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, 1987. Second edition.

[5] K. Marriott and H. Sondergaard. Bottom-up Dataflow Analysis of Normal Logic Programs. *Journal of Logic Programming*, 13(2 & 3):181–204, 1992.

[6] H. Tamaki and T. Sato. Unfold/Fold Transformations of Logic Programs. In Sten-Åke Tärnlund, editor, *Proc. Second Int'l Conf. on Logic Programming*, pages 127–139, 1984.

[7] 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.