Possibilistic Well-Founded Semantics

Mauricio Osorio[1] and Juan Carlos Nieves[2]

[1] Universidad de las Américas - Puebla
CENTIA, Sta. Catarina Mártir, Cholula, Puebla, 72820 México
osoriomauri@googlemail.com
[2] Universitat Politècnica de Catalunya
Software Department (LSI)
c/Jordi Girona 1-3, E08034, Barcelona, Spain
jcnieves@lsi.upc.edu

Abstract. Recently, a good set of logic programming semantics has been defined for capturing possibilistic logic program. Practically all of them follow a credulous reasoning approach. This means that given a possibilistic logic program one can infer a set of possibilistic models. However, sometimes it is desirable to associate just one possibilistic model to a given possibilistic logic program. One of the main implications of having just one model associated to a possibilistic logic program is that one can perform queries directly to a possibilistic program and answering these queries in accordance with this model.
In this paper, we introduce an extension of the Well-Founded Semantics, which represents a sceptical reasoning approach, in order to capture possibilistic logic programs. We will show that our new semantics can be considered as an approximation of the possibilistic semantics based on the answer set semantics and the pstable semantic. A relevant feature of the introduced semantics is that it is polynomial time computable.

1 Introduction

In [10], a possibilistic logic programming framework for reasoning under uncertainty was introduced. It is a combination between Answer Set Programming (ASP) [3] and Possibilistic Logic [7]. This framework is able to deal with reasoning that is at the same time non-monotonic and uncertain. Since this framework was defined for normal programs, it was generalized in [11] for capturing possibilistic disjunctive programs and allowing the encoding of uncertain information by using either numerical values or relative likelihoods.

The expressiveness of this approach is rich enough for capturing sophisticated domains such as river basin systems [1, 2]. In fact, one can suggest that the language's expressiveness of the possibilistic logic programs is rich enough for capturing a wide family of problems where one have to confront with incomplete information and uncertain information.

From the logic programming literature [10–13], we can see that all the possibilistic logic program semantics that have defined until now for capturing the semantics of possibilistic logic programs (to the best of our knowledge) follow a credulous reasoning approach. This means that given a possibilistic logic program one can infer a set of possibilistic models. However, sometimes it is desirable to associate just one possibilistic model to a given possibilistic. This means to perform a skeptical reasoning approach from a possibilistic knowledge base. It is well-known, in the logic programming community, that a skeptical reasoning approach has several and practical implications in order to apply a logic programming approach into real domain applications. Some of these implications are:

- the process of performing a skeptical reasoning approach usually is polynomial time computable.
- to associate a single model to a logic programs helps to define algorithms for performing top-down queries from a knowledge base.

In the literature of logic programming, we can find several logic programming semantics which perform a skeptical reasoning approach [5, 6, 8]. However, the well-accepted logic programming semantics for performing a skeptical reasoning approach is *the well-founded semantics* introduced by Van Gelder in [8]. There are several results which suggest that the well-founded semantics is a strong logic programming semantic for performing a skeptical reasoning approach. For instance, Dix in [5] showed that the well-founded semantics is a *well-behaved semantics*[3]. It worth mentioning that there are few logic programming semantics which are well-behaved. In fact most of them are variations of the well-founded semantics.

Given that a skeptical reasoning approach has important implications for performing non-monotonic reasoning from a possibilistic knowledge base, in this paper, we introduce a possibilistic version of the well-founded semantics. Our so called possibilistic well-founded semantics will be a combination of some features of possibilistic logic and the standard well-founded semantics. We define the possibilistic well-founded semantics for two classes of possibilistic logic programs: Extended Possibilistic Definite logic programs and Extended Possibilistic Normal Logic programs. We show that our extension of the well-founded semantics preserves the important property of being polynomial time computable. Another important property of our possibilistic well founded semantics is that it can be considered as an approximation of the possibilistic answer set semantics [10, 12] and the possibilistic pstable semantics [13].

The rest of the paper is divided as follows: In §2, some basic concepts *w.r.t.* possibilistic logic are presented. Also the syntaxis of extended logic programs is defined and a characterization of the well-founded semantics in terms of rewriting systems is presented. In §3, the syntaxis of the extended possibilistic logic programs is presented. In §4, the definition of the possibilistic well-founded semantics is defined. Finally in the last section our conclusions are presented.

[3] A logic programming semantics is called well-behaved if it satisfies the following properties: Cut, Closure, Weak Model-Property, Isomorphy, M_P-extension, Transformation, Relevance, Reduction, PPE and modularity [5].

2 Background

In this section, we define some basic concepts of possibilistic logic, logic program syntaxis and a characterization of the well-founded semantics in terms of rewriting systems.

We assume familiarity with basic concepts in classic logic and in semantics of logic programs *e.g.,* interpretation, model, *etc*. A good introductory treatment of these concepts can be found in [3, 9].

2.1 Possibilistic Logic

A necessity-valued formula is a pair $(\varphi\ \alpha)$ where φ is a classical logic formula and $\alpha \in (0, 1]$ is a positive number. The pair $(\varphi\ \alpha)$ expresses that the formula φ is certain at least to the level α, *i.e.*, $N(\varphi) \geq \alpha$, where N is a necessity measure modeling our possibly incomplete state knowledge [7]. α is not a probability (like it is in probability theory) but it induces a certainty (or confidence) scale. This value is determined by the expert providing the knowledge base. A necessity-valued knowledge base is then defined as a finite set (*i.e.*, a conjunction) of necessity-valued formulae.

Dubois *et al.*[7] introduced a formal system for necessity-valued logic which is based on the following axioms schemata (propositional case):

(A1) $(\varphi \to (\psi \to \varphi)\ 1)$
(A2) $((\varphi \to (\psi \to \xi)) \to ((\varphi \to \psi) \to (\varphi \to \xi))\ 1)$
(A3) $((\sim \varphi \to \sim \psi) \to ((\sim \varphi \to \psi) \to \varphi)\ 1)$

As in classic logic, the symbols $\neg$ and $\to$ are considered primitive connectives, then connectives as $\vee$ and $\wedge$ are defined as abbreviations of $\neg$ and $\to$. Now the inference rules for the axioms are:

(GMP) $(\varphi\ \alpha), (\varphi \to \psi\ \beta) \vdash (\psi\ GLB\{\alpha, \beta\})$
(S) $(\varphi\ \alpha) \vdash (\varphi\ \beta)$ if $\beta \leq \alpha$

According to Dubois *et al.*, basically we need a complete lattice in order to express the levels of uncertainty in Possibilistic Logic. Dubois *et al.*, extended the axioms schemata and the inference rules for considering partially ordered sets. We shall denote by $\vdash_{PL}$ the inference under Possibilistic Logic without paying attention if the necessity-valued formulae are using either a totally ordered set or a partially ordered set for expressing the levels of uncertainty.

The problem of inferring automatically the necessity-value of a classical formula from a possibilistic base was solved by an extended version of *resolution* for possibilistic logic (see [7] for details).

2.2 Syntaxis: Logic programs

The language of a propositional logic has an alphabet consisting of

(i) proposition symbols: $\bot, \top, p_0, p_1, ...$
(ii) connectives : $\vee, \wedge, \leftarrow, \neg,\ not$

(iii) auxiliary symbols : (,).

where $\vee, \wedge, \leftarrow$ are 2-place connectives, $\neg$, not are 1-place connective and $\bot$, $\top$ are 0-place connective. The proposition symbols, $\bot$, and the propositional symbols of the form $\neg p_i$ ($i \geq 0$) stand for the indecomposable propositions, which we call *atoms*, or *atomic propositions*. Atoms negated by $\neg$ will be called *extended atoms*. We will use the concept of atom without paying attention if it is an extended atom or not. The negation sign $\neg$ is regarded as the so called *strong negation* by the ASP's literature and the negation not as the *negation as failure*. A literal is an atom, a (called positive literal), or the negation of an atom $not\ a$ (called negative literal). Given a set of atoms $\{a_1, ..., a_n\}$, we write $not\ \{a_1, ..., a_n\}$ to denote the set of literals $\{not\ a_1, ..., not\ a_n\}$.

An extended normal clause, *C*, is denoted:

$$a \leftarrow a_1, \ldots, a_j, not\ a_{j+1}, \ldots, not\ a_n$$

where $j + n \geq 0$, a is an atom and each a_i is an atom. When $j + n = 0$ the clause is an abbreviation of $a \leftarrow \top$ such that $\top$ is the proposition symbol that always evaluate to true. An extended normal program P is a finite set of extended normal clauses. When $n = 0$, the clause is called *extended definite clause. An extended definite logic program* is a finite set of extended definite clauses. By $\mathcal{L}_P$, we denote the set of atoms in the language of P. Let $Prog_{\mathcal{L}}$ be the set of all normal programs with atoms from $\mathcal{L}$.

We will manage the strong negation ($\neg$), in our logic programs, as it is done in ASP [3]. Basically, it is replaced each atom of the form $\neg a$ by a new atom symbol a' which does not appear in the language of the program. For instance, let P be the extended normal program:

$$a \leftarrow q. \qquad \neg q \leftarrow r. \qquad q \leftarrow \top. \qquad r \leftarrow \top.$$

Then replacing the atom $\neg q$ by a new atom symbol q', we will have:

$$a \leftarrow q. \qquad q' \leftarrow r. \qquad q \leftarrow \top. \qquad r \leftarrow \top.$$

In order not to allow inconsistent models from logic programs, usually it is added a normal clause of the form $f \leftarrow q, q', f$ such that $f \notin \mathcal{L}_P$. We will omit this clause in order to allow an inconsistent level in our possibilistic-WFS. However the user could add this clause without losing generality.

Sometimes we denote an extended normal clause *C* by $a \leftarrow \mathcal{B}^+,\ not\ \mathcal{B}^-$, where $\mathcal{B}^+$ contains all the positive body literals and $\mathcal{B}^-$ contains all the negative body literals.

2.3 Well-Founded Semantics

In this section, we present a standard definition of the well-founded semantics in terms of rewriting systems. We start presenting a definition *w.r.t.* 3-valued logic semantics.

Definition 1 (SEM). *[6] For normal logic program P, we define HEAD*$(P) = \{a |\ a \leftarrow \mathcal{B}^+,\ not\ \mathcal{B}^- \in P\}$ *— the set of all head-atoms of P. We also define SEM*$(P) = \langle P^{true}, P^{false} \rangle$*, where* $P^{true} := \{p |\ p \leftarrow \top \in P\}$ *and* $P^{false} := \{p |\ p \in \mathcal{L}_P \backslash HEAD(P)\}$. *SEM*$(P)$ *is also called model of P.*

In order to present a characterization of the well-funded semantics in terms of rewriting systems, we define some basic transformation rules for normal logic programs.

Definition 2 (Basic Transformation Rules). *[6] A transformation rule is a binary relation on $Prog_{\mathcal{L}}$. The following transformation rules are called* basic. *Let a program $P \in Prog_{\mathcal{L}}$ be given.*

RED$^+$: *This transformation can be applied to P, if there is an atom a which does not occur in HEAD(P).* ***RED$^+$*** *transforms P to the program where all occurrences of $not\ a$ are removed.*
RED$^-$: *This transformation can be applied to P, if there is a rule $a \leftarrow \top \in P$.* ***RED$^-$*** *transforms P to the program where all clauses that contain $not\ a$ in their bodies are deleted.*
Success: *Suppose that P includes a fact $a \leftarrow \top$ and a clause $q \leftarrow body$ such that $a \in body$. Then we replace the clause $q \leftarrow body$ by $q \leftarrow body \setminus \{a\}$.*
Failure: *Suppose that P contains a clause $q \leftarrow body$ such that $a \in body$ and $a \notin HEAD(P)$. Then we erase the given clause.*
Loop: *We say that P_2 results from P_1 by* Loop$_A$ *if, by definition, there is a set A of atoms such that 1. for each rule $a \leftarrow body \in P_1$, if $a \in A$, then $body \cap A \neq \emptyset$, 2. $P_2 := \{a \leftarrow body \in P_1 | body \cap A = \emptyset\}$, 3. $P_1 \neq P_2$.*

Let CS_0 be the rewriting system such that contains the transformation rules: RED^+, RED^-, $Success$, $Failure$, and $Loop$. We denote the uniquely determined normal form of a program P with respect to the system $\mathcal{CS}$ by $norm_{\mathcal{CS}}(P)$. Every system $\mathcal{CS}$ induces a semantics SEM$_{\mathcal{CS}}$ as follows: SEM$_{\mathcal{CS}}(P) :=$ SEM$(norm_{\mathcal{CS}}(P))$.

In order to illustrate the basic transformation rules, let us consider the following example.

Example 1. Let P be the following normal program:

$$d(b) \leftarrow \ not\ d(a).\ \ d(c) \leftarrow \ not\ d(b).\ \ d(c) \leftarrow d(a).$$

Now, let us apply CS_0 to P. Since $d(a) \notin HEAD(P)$, then, we can apply **RED$^+$** to P. Thus we get:

$$d(b) \leftarrow \top.\ \ d(c) \leftarrow \ not\ d(b).\ \ d(c) \leftarrow d(a).$$

Notice that now we can apply **RED$^-$** to the new program, thus we get: $d(b) \leftarrow \top.$ $d(c) \leftarrow d(a).$
Finally, we can apply **Failure** to the new program, thus we get: $d(b) \leftarrow \top$. This last program is called the *normal form* of P *w.r.t.* CS_0, because none of the transformation rules from CS_0 can be applied.

WFS was introduced in [8] and was characterized in terms of rewriting systems in [4]. This characterization is defined as follows:

Lemma 1. *[4] CS_0 is a confluent rewriting system. It induces a 3-valued semantics that it is the Well-founded Semantics.*

3 Possibilistic Logic Programs

In this section, we introduce a standard syntax of extended possibilistic logic programs. In whole paper, we will consider finite lattices. This convention was taken based on the assumption that in real applications rarely we will have an infinite set of labels for expressing the incomplete state of a knowledge base.

3.1 Syntax

First of all, we start defining some relevant concepts[4]. A *possibilistic atom* is a pair $p = (a, q) \in \mathcal{A} \times Q$, where $\mathcal{A}$ is a finite set of atoms and $(Q, \leq)$ is a lattice. We apply the projection $*$ over p as follows: $p^* = a$. Given a set of possibilistic atoms S, we define the generalization of $*$ over S as follows: $S^* = \{p^* | p \in S\}$. Given a lattice $(Q, \leq)$ and $S \subseteq Q$, $LUB(S)$ denotes the least upper bound of S and $GLB(S)$ denotes the greatest lower bound of S.

We define the syntax of a valid extended possibilistic normal logic program as follows: Let $(Q, \leq)$ be a lattice. A extended possibilistic normal clause r is of the form:

$$r := (\alpha : a \leftarrow \mathcal{B}^{+},\ not\ \mathcal{B}^{-})$$

where $\alpha \in Q$. The projection $*$ over the possibilistic clause r is: $r^* = a \leftarrow \mathcal{B}^{+},\ not\ \mathcal{B}^{-}$. $n(r) = \alpha$ is a necessity degree representing the certainty level of the information described by r.

An extended possibilistic normal logic program P is a tuple of the form $\langle (Q, \leq), N \rangle$, where $(Q, \leq)$ is a lattice and N is a finite set of extended possibilistic normal clauses. The generalization of the projection $*$ over P is as follows: $P^* = \{r^* | r \in N\}$. Notice that P^* is an extended logic normal program. When P^* is an extended definite program, P is called an extended possibilistic definite logic program.

4 Possibilistic Well-Founded Model Semantics

In this section, we introduce the possibilistic version of the well-founded semantics. This semantics will be presented for two classes of possibilistic logic programs: extended possibilistic definite logic programs and extended possibilistic normal logic programs.

4.1 Extended Possibilistic Definite Logic Programs

In this subsection, we are going to deal with the class of extended possibilistic logic programs. For capturing extended possibilistic definite logic program, we are going to consider the basic idea of *possibilistic least model* which was introduced in [10]. For this purpose, we are going to introduce some basic definitions.

The first basic definition that we present is the definition of three basic operators between sets of possibilistic atoms.

[4] Some concepts presented in this subsection extend some terms presented in [10].

Definition 3. *[10] Let $\mathcal{A}$ be a finite set of atoms and $(Q, \leq)$ be a lattice. Consider $\mathcal{PS} = 2^{\mathcal{A} \times Q}$ the finite set of all the possibilistic atom sets induced by $\mathcal{A}$ and Q. $\forall A, B \in \mathcal{PS}$, we define.*

$$\begin{aligned}
A \sqcap B &= \{(x, GLB\{q_1, q_2\}) | (x, q_1) \in A \wedge (x, q_2) \in B\} \\
A \sqcup B &= \{(x, q) | (x, q) \in A \text{ and } x \notin B^*\} \cup \\
&\quad \{(x, q) | x \notin A^* \text{ and } (x, q) \in B\} \cup \\
&\quad \{(x, LUB\{q_1, q_2\}) | (x, q_1) \in A \text{ and } (x, q_2) \in B\}. \\
A \sqsubseteq B &\iff A^* \subseteq B^*, \text{ and } \forall x, q_1, q_2, \\
&\quad (x, q_1) \in A \wedge (x, q_2) \in B \text{ then } q_1 \leq q_2.
\end{aligned}$$

Observe that essentially this definition suggests an extension of the standard operators between sets in order to deal with uncertain values which belong to a partially ordered set. We want to point out that the original version of Definition 3 consider totally ordered sets instead of partially ordered sets.

Like in [10], we are going to introduce a fix-point operator ΠCn. In order to define ΠCn, let us introduce some basic definitions. Given a possibilistic logic program P and $x \in \mathcal{L}_{P^*}$, $H(P, x) = \{r \in P | head(r^*) = x\}$.

Definition 4. *Let $P = \langle (Q, \leq), N \rangle$ be a possibilistic definite logic program, $r \in N$ such that r is of the form $\alpha : a \leftarrow l_1, \ldots, l_n$ and A be a set of possibilistic atoms,*

- *r is β-applicable in A with $\beta = min\{\alpha, \alpha_1, \ldots, \alpha_n\}$ if $\{(l_1, \alpha_1), \ldots, (l_n, \alpha_n)\} \subseteq A$.*
- *r is $\bot_Q$-applicable otherwise.*

And then, for all atom $x \in \mathcal{L}_{P^*}$ we define:

$$App(P, A, x) = \{r \in H(P, x) | r \text{ is } \beta\text{-applicable in } A \text{ and } \beta > \bot_Q\}$$

$\bot_Q$ denotes the bottom element of Q.

Observe that this definition is based on the inferences rules of possibilistic logic. In order to illustrate this definition, let us consider the following example.

Example 2. Let $P = \langle (Q, \leq), N \rangle$ be a possibilistic definite logic program such that $Q = \{0, 1, \ldots, 0.9, 1\}$, $\leq$ be the standard relation between rational number, and N be the following set of possibilistic definite clauses:

$$\begin{aligned}
r_1 &= 0.4 : a \leftarrow \top. \\
r_2 &= 0.3 : b \leftarrow a. \\
r_3 &= 0.6 : b \leftarrow a. \\
r_4 &= 0.7 : m \leftarrow n.
\end{aligned}$$

we can see that if we consider $A = \emptyset$, then r_1 is 0.4-applicable in A. In fact, we can see that $App(P, A, a) = \{r_1\}$. Also, we can see that if $A = \{(a, 0.4)\}$, then r_2 is 0.4-applicable in A and r_3 is 0.6-applicable in A. Observe that $App(P, A, b) = \{r_2, r_3\}$.

Now, we introduce an operator which is based on Definition 4.

Definition 5. *Let P be a possibilistic definite logic program and A be a set of possibilistic atoms. The immediate possibilistic consequence operator ΠT_P maps a set of possibilistic atoms to another one by this way:*

$$\Pi T_P(A) = \{(x,\delta) | x \in HEAD(P^*), App(P,A,x) \neq \emptyset, \\ \delta = LUB_{r \in App(P,A,x)}\{\beta | r \textit{ is } \beta\textit{-applicable in } A\}\}$$

Then the iterated operator ΠT_P^k is defined by

$$\Pi T_P^k = \emptyset \textit{ and } \Pi T_P^{n+1} = \Pi T_P(\Pi T_P^n), \forall n \geq 0$$

Observe that ΠT_P is a monotonic operator; therefore, we can insure that ΠT_P always reaches a fix-point.

Proposition 1. *Let P be a possibilistic definite logic program, then ΠT_P has a least fix-point $\bigsqcup_{n \geq 0} \Pi T_P^n$ that we call the set of possibilistic consequences of P and we denote it by $\Pi Cn(P)$.*

Example 3. Let P be the extended possibilistic definite logic program introduced in Example 2. It is not difficult to see that $\Pi Cn(P) = \{(a, 0.3), (b, 0.6)\}$.

By considering the operator ΠCn, we define the possibilistic well-founded semantics for extended possibilistic definite logic program as follows: Let $(Q, \leq)$ be a lattice such that $\top_Q$ is the top-element of Q and S be a set of atoms, then $Q_{\top_Q}(S) = \{(a, \top_Q) | a \in S\}$.

Definition 6. *Let P be an extended possibilistic definite logic program. S_1 be a set of possibilistic atoms, S_2 be a set of atoms such that $\langle S_1^*, S_2 \rangle$ is the well-founded model of P^*. $\langle S_1, Q_{\top_Q}(S_2) \rangle$ is the possibilistic well-founded model of P if and only if $S_1 = \Pi Cn(P)$.*

Example 4. Let P be the extended possibilistic definite logic program introduced in Example 2, $S_1 = \{(a, 0.3), (b, 0.6)\}$ and $S_2 = \{m, n\}$. As we can see the well-founded model of P^* is $\langle \{a, b\}, \{m, n\} \rangle$, and $\Pi Cn(P) = \{(a, 0.3), (b, 0.6)\}$. This means that the well-founded model of P is:

$$\langle \{(a, 0.3), (b, 0.6)\}, \{(m, 1), (n, 1)\} \rangle$$

4.2 Extended Possibilistic Normal Programs

In this subsection, we are going to deal with the class of extended possibilistic normal programs. In order to define a possibilistic version of the possibilistic well-founded semantics for capturing extended possibilistic normal program, we define a single reduction of an extended possibilistic normal logic program *w.r.t.* a set of atoms. This reduction is defined as follows:

Definition 7. *Let P be an extended possibilistic logic program and S be a set of atoms. We define $R(P, S)$ as the extended possibilistic logic program obtained from P by deleting*

i *all the formulae of the form* $not\ a$ *in the bodies of the possibilistic clauses such that* $a \in S$*, and*
ii *each possibilistic clause that has a formula of the form* $not\ a$ *in its body.*

Observe that $R(P, S)$ does not have negative literals. This means that $R(P, S)$ is an extended possibilistic definite logic program. In order to illustrate this definition, let us consider the following example.

Example 5. Let $P = \langle (Q, \leq), N \rangle$ be a possibilistic logic program such that $Q = \{0, 0.1, \ldots, 0.9, 1\}$, $\leq$ be the standard relation between rational number, $S = \{b, d, e\}$ and N be the following set of possibilistic clauses:

$$\begin{array}{l} 0.4 : a \leftarrow \ not\ b. \\ 0.6 : b \leftarrow \ not\ c. \\ 0.3 : c \leftarrow \ not\ d. \\ 0.2 : c \leftarrow \ not\ e. \\ 0.5 : f \leftarrow \ not\ f. \end{array}$$

As we can see in Definition 7, for inferring $R(P, S)$, we have to apply two steps. The first step is to remove all the negative literals $not\ a$ from P such that a belongs to S. This means that by the first step of Definition 7 we have the following possibilistic program

$$\begin{array}{l} 0.4 : a \leftarrow \top. \\ 0.6 : b \leftarrow \ not\ c. \\ 0.3 : c \leftarrow \top. \\ 0.2 : c \leftarrow \top. \\ 0.5 : f \leftarrow \ not\ f. \end{array}$$

The next and last step is to remove any possibilistic clause that has a negative literal in its body. This means that $R(P, S)$ is:

$$\begin{array}{l} 0.4 : a \leftarrow \top. \\ 0.3 : c \leftarrow \top. \\ 0.2 : c \leftarrow \top. \end{array}$$

As we can see, $R(P, S)$ is a possibilistic definite logic program.

By considering the fix-point operator $\Pi Cn(P)$ and the reduction $R(P, A)$, we define the possibilistic version of the well-founded semantics for extended possibilistic normal logic programs as follows:

Definition 8 (Possibilistic Well-founded Semantics).
Let $P = \langle (Q, \leq), N \rangle$ *be an extended possibilistic logic program,* S_1 *be a set of possibilistic atoms,* S_2 *be a set of atoms such that* $\langle S_1^*, S_2 \rangle$ *is the well-founded model of* P^**.* $\langle S_1, Q_{\top_Q}(S_2) \rangle$ *is the possibilistic well-founded model of* P *if and only if* $S_1 = \Pi Cn(R(P, S_2))$*.*

In order to illustrate this definition let us consider the following example

Example 6. Let $P = \langle (Q, \leq), N \rangle$ be the possibilistic logic program introduced in Example 5, $S_1 = \{(a\ 0.4), (c\ 0.3)\}$ and $S_2 = \{b, d, e\}$. On order to infer the possibilistic well-founded model of P, we have to infer the well-founded model of the normal program P^*. It is easy to see that $WPF(P^*) = \langle S_1^*, S_2 \rangle$. As we saw in Example 5, $R(P, S_2)$ is

$$\begin{array}{l} 0.4 : a \leftarrow \top. \\ 0.3 : c \leftarrow \top. \\ 0.2 : c \leftarrow \top. \end{array}$$

hence, we can see that $Cn(R(P, S_2) = \{(a, 0.4), (c, 0.3)\}$. This suggests that the possibilistic well-founded model of P is

$$\langle \{(a, 0.4), (c, 0.3)\}, \{(b, 1), (d, 1), (e, 1)\} \rangle$$

A first and basic observation that we can see from the definition of the possibilistic well-founded semantics is that the possibilistic well-founded semantics infers the well-founded semantics.

Proposition 2. *Let $P = \langle (Q, \leq), N \rangle$ be an extended possibilistic logic program. If $\langle S_1, s_2 \rangle$ is the possibilistic well-founded model of P then $\langle S_1^*, S_2^*) \rangle$ is the well-founded model of P^*.*

Usually, one property that always is desired from a logic program semantics is that it could be polynomial time computable *w.r.t.* the size of a given logic program. It is well-known that the well-founded semantics is polynomial time computable. For the case of the possibilistic well-founded semantics we can also insure that it is polynomial time computable. For formalizing this property, let us remember that the size of a logic programs is defined as follows: The size of a possibilistic clause r is the number of atom symbols that occurs in r. The size of a possibilistic logic program P is the sum of sizes of the possibilistic clauses that belong to P.

Proposition 3. *Given an extended possibilistic normal program P, there is an algorithm that computes the possibilistic well founded model of P in polynomial time w.r.t. the size of P.*

In the following proposition, a relationship between the possibilistic answer set semantics and the possibilistic well-founded model is formalized. Also a relationship between the possibilistic pstable semantics and the possibilistic well-founded model. In order to formalize these relationships, we define the following functions: $Poss\_ASP(P)$ denotes a function which returns the set of possibilistic answer semantics [10, 12] of a possibilistic logic program P and $Poss\_Pstable(P)$ denotes a function which returns the set of possibilistic pstable models [13] of a possibilistic logic program P.

Proposition 4. *Let P be a possibilistic logic program and $S_1, S_2 \subseteq \mathcal{L}_P$ such that $\langle S_1, S_2 \rangle$ is the possibilistic well-founded model of P. Hence, the following conditions holds:*

– *If $Poss\_ASP(P) \neq \emptyset$; hence, if $S = \bigcap_{S' \in Poss\_ASP(P)} S'$, then $S_1 \sqsubseteq S$.*

– *If* $S = \bigcap_{S' \in Poss\_Pstable(P)} S'$, *then* $S_1 \sqsubseteq S$.

Observe that this proposition essentially suggests that any possibilistic answer set model is an extension of the possibilistic well-founded model. Also that any possibilistic pstable model is an extension of the possibilistic well-founded model. It is worth mentioning that this relationship between the standard answer set semantics and the well-founded semantics was shown by Dix in [5].

5 Conclusions

In this paper have explored the definition of a possibilistic version of the well-founded semantics in order to capture possibilistic logic programs. For this purpose, we first define a possibilistic version of the well-founded semantics for extended possibilistic define logic programs. This definition considers a possibilistic operator for inferring the possibilistic least model of an extended possibilistic definite logic program and the standard definition of the well-founded semantics. In order to define the possibilistic version of the well-founded semantics for extended possibilistic normal logic programs, we introduce a single reduction of a possibilistic logic program in terms of a given set of atoms. We want to point that our construction of the possibilistic well-founded semantics is flexible enough for considering other variants of the well-founded semantics as the explored in [5, 6]. Hence, we can consider our construction of the possibilistic well-founded semantics for exploring different approaches of skeptical reasoning from a possibilistic knowledge base.

We have showed that the actual version of the possibilistic version of the well-founded semantics is polynomial time computable (Proposition 3). This suggests that one can explore efficient algorithms for performing top-down queries from a possibilistic knowledge. In fact, this issue is part of our future work.

Also we have showed that our possibilistic well-founded semantics can be regarded as an approximation of credulous possibilistic semantics as the possibilistic answer set semantics and the possibilistic pstable semantics (Proposition 4). In fact, we can conclude that any possibilistic answer set model is an extension of the possibilistic well-founded model and that any possibilistic pstable model is an extension of the possibilistic well-founded model.

Acknowledgement

We are grateful to anonymous referees for their useful comments. This research has been partially supported by the EC founded project ALIVE (FP7-IST-215890). The views expressed in this paper are not necessarily those of the ALIVE consortium.

References

1. M. Aulinas. *Management of industrial wastewater discharges through agents' argumentation.* PhD thesis, PhD on Environmental Sciences, University of Girona, to be presented.

2. M. Aulinas, J. C. Nieves, M. Poch, and U. Cortés. Supporting Decision Making in River Basin Systems Using a Declarative Reasoning Approach. In M. Finkel and P. Grathwohl, editors, *Proceedings of the AquaTerra Conference (Scientific Fundamentals for River Basic Management), ISSN 0935-4948*, page 75, March 2009.
3. C. Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, Cambridge, 2003.
4. S. Brass, U. Zukowski, and B. Freitag. Transformation-based bottom-up computation of the well-founded model. In *NMELP*, pages 171–201, 1996.
5. J. Dix. A classification theory of semantics of normal logic programs: II. weak properties. *Fundam. Inform.*, 22(3):257–288, 1995.
6. J. Dix, M. Osorio, and C. Zepeda. A general theory of confluent rewriting systems for logic programming and its applications. *Ann. Pure Appl. Logic*, 108(1-3):153–188, 2001.
7. D. Dubois, J. Lang, and H. Prade. Possibilistic logic. In D. Gabbay, C. J. Hogger, and J. A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming, Volume 3: Nonmonotonic Reasoning and Uncertain Reasoning*, pages 439–513. Oxford University Press, Oxford, 1994.
8. A. V. Gelder, K. A. Ross, and J. S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, 1991.
9. E. Mendelson. *Introduction to Mathematical Logic*. Chapman and Hall/CRC, Fourth edition 1997.
10. P. Nicolas, L. Garcia, I. Stéphan, and C. Lafèvre. Possibilistic Uncertainty Handling for Answer Set Programming. *Annals of Mathematics and Artificial Intelligence*, 47(1-2):139–181, June 2006.
11. J. C. Nieves, M. Osorio, and U. Cortés. Semantics for possibilistic disjunctive programs. In C. Baral, G. Brewka, and J. Schlipf, editors, *Ninth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR-07)*, number 4483 in LNAI, pages 315–320. Springer-Verlag, 2007.
12. J. C. Nieves, M. Osorio, and U. Cortés. Semantics for possibilistic disjunctive programs. In S. Costantini and R. Watson, editors, *Answer Set Programming: Advances in Theory and Implementation*, pages 271–284, 2007.
13. M. Osorio and J. C. Nieves. Pstable semantics for possibilistic logic programs. In *MICAI 2007: Advances in Artificial Intelligence, 6th Mexican International Conference on Artificial Intelligence*, number 4827 in LNAI, pages 294–304. Springer-Verlag, 2007.