Departament de Llenguatges
i Sistemes Informàtics

UNIVERSITAT POLITÈCNICA DE CATALUNYA

Towards the Implementation of a Preference- and Uncertain-Aware Solver Using Answer Set Programming

Roberto Confalonieri, Juan Carlos Nieves, Javier Vázquez-Salceda

Technical Report LSI-10-16-R

June 2010

Dept. Llenguatges i Sistemes Informàtics (LSI)
Universitat Politècnica de Catalunya

Towards the Implementation of a Preference- and Uncertain-Aware Solver Using Answer Set Programming

Roberto Confalonieri, Juan Carlos Nieves, Javier Vázquez-Salceda

Dept. Llenguatges i Sistemes Informàtics (LSI)
Universitat Politècnica de Catalunya
C/ Jordi Girona Salgado 1-3
E - 08034 Barcelona
`{confalonieri,jcnieves,jvazquez}@lsi.upc.edu`

Technical Report LSI-10-16-R

June 2010

Abstract

Logic programs with possibilistic ordered disjunction (or LPPODs) are a recently defined logic-programming framework based on logic programs with ordered disjunction and possibilistic logic. The framework inherits the properties of such formalisms and merging them, it supports a reasoning which is nonmonotonic, preference- and uncertain-aware. The LPPODs syntax allows to specify 1) preferences in a qualitative way, and 2) necessity values about the certainty of program clauses. As a result at semantic level, preferences and necessity values can be used to specify an order among program solutions. This class of program therefore fits well in the representation of decision problems where a best option has to be chosen taking into account both preferences and necessity measures about information. In this technical report we study the computation and the complexity of the LPPODs semantics and we describe the algorithm for its implementation following on Answer Set Programming approach. We describe some decision scenarios where the solver can be used to choose the best solutions by checking whether an outcome is possibilistically preferred over another considering preferences and uncertainty at the same time.

1 Introduction

Logic programs based on answer set semantics [14] are usually considered expressive enough to address many knowledge representation problems in Artificial Intelligence. In fact they have been regarded as the computational incarnation of nonmonotonic reasoning. Answer set programming (ASP) is a form of declarative programming towards complex combinatorial problems which has been applied in multiple areas such as product configuration [18], planning [15] and diagnosis [1]. Its expressivity allows reasoning about incomplete information, but at the same time their syntax is restrictive enough to allow the implementation of several efficient answer set solvers such as *dlv* [12] and *smodels* [17]. However, with the increasing complexity of many type of qualitative decision making contexts, answer set programs lack the capability to express preferences [3]. For this reason several ASP extensions have been proposed to model preferences [9], showing how ASP can constitute an effective way of solving indeterminate solutions, reasoning in terms of preferred answer sets of a logic program. One of these extensions, which has its root in qualitative choice logic [4] is logic programs with ordered disjunction [5].

Logic programs with ordered disjunction (or LPODs) are extended logic programs augmented with an *ordered disjunction* connector $\times$ which permits to explicitly represent qualitative preferences in the head of ordered disjunction rules [5]. Programs of this form can capture user qualitative preferences by means of ordered disjunction rules, represent choices among different alternatives and specify a preference order between the answer sets through predefined comparison criteria. LPODs have been used in applications such as policy languages [2], game theory [13] and user preference representation and reasoning [5]. The semantics of LPODs is implemented by an efficient solver *psmodels* [5].

Although LPODs have shown to be an effective way to model preferences and to specify an order between their outcomes, a priority order between the program clauses can only be specified by defining static meta-preference relations between its preference rules. But in some realistic scenarios, where a knowledge discovery process may retrieve uncertain information, it is desiderable to be able to capture and reason about uncertain knowledge to define a priority order between preference rules in a dynamic way.

Logic programs with possibilistic ordered disjunction (or LPPODs) are a recently defined logic programming framework based on logic programs with ordered disjunction and possibilistic logic [7] which join together in only one formalism common-sense reasoning with qualitative preferences and reasoning under uncertainty. The LPPODs syntax allows to specify qualitative preferences and to associate a priority order to ordered disjunctions rules by means of necessity values according to possibilistic logic [11]. As a result at semantic level, preferences and necessity values can be used to specify an order among program solutions. This class of programs fits well in the representation of decision problems where a best option has to be chosen taking into account both preferences and necessity measures about information. In [7] the semantics for this class of programs has been defined but neither an algorithm nor a complexity study have been presented.

In this technical report we will study the computation and the complexity of LPPODs and we describe the algorithm for its implementation. The algorithm considers the answer set semantics and the possibilistic stable semantics [16] as building block and describes how LPPODs semantics can be computed using a possibilistic program reduction and a possibilistic consequence operator which are a syntactic approach to compute the possibilistic answer set semantics of LPPODs. The LPPODs implementation will be able to compute the candidate possibilistic answer sets and to check whether a given candidate is possibilistically preferred over another considering rules' satisfaction degrees and rules' necessity values at the same time. We present two possible applications where the solver can be used and we compare the LPPODs representation with others such as possibilistic normal logic programs and LPODs.

The technical report is organized as follows. After giving some background information of the basic concepts involved (Section 2), we discuss in Section 3 the computation and the complexity of the LPPODs semantics. Our main results in this section is that the LPPODs semantics is computable and we present the algorithm. In Section 4 we discuss two application scenarios and we show how the LPPOD implementation can be used to decide the best options given the preferences and the necessity values associated to the program clauses. Finally with Section 5 we conclude the technical report.

2 Background

In this section we present the basic definitions we will use throughout the technical report. We assume that the reader has familiarity with basic concepts of *answer set semantics* [14], *possibilistic stable model semantics* [16], and *possibilistic logic* [11].[1]

2.1 Logic Programs with Possibilistic Ordered Disjunction

We consider extended logic programs which have two kinds of negation, strong negation $\neg$ and default negation not. A signature $\mathcal{L}$ is a finite set of elements that we call atoms, where atoms negated by $\neg$ are called *extended atoms*. Intuitively, $not\ a$ is true whenever there is no reason to believe a, whereas $\neg a$ requires a proof of the negated atom. In the following we use the concept of atom without paying attention if it is an extended atom or not. A *literal* is either 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 atoms $\{not\ a_1, ...,\ not\ a_n\}$.

Logic programs with possibilistic ordered disjunction (LPPODs) are a recently defined logic-programming framework based on logic programs with ordered disjunction (LPODs) and possibilistic logic [7]. LPODs are extended logic programs with an ordered disjunction connector $\times$ which allows to express qualitative preferences in the head of rules [5] and necessity values to the program clause themselves can be specified according to possibilistic logic [11]. Given a finite lattice $(\mathcal{Q}, \leq)$, an LPPOD is a tuple of the form $P := \langle(\mathcal{Q}, \leq), N\rangle$ where N is a finite set of *possibilistic ordered disjunction rules* such as:

$$\alpha : c_1 \times \ldots \times c_k \leftarrow b_1, \ldots, b_m,\ not\ b_{m+1}, \ldots,\ not\ b_{m+n}$$

where $\alpha \in \mathcal{Q}$ and $c_i (1 \leq i \leq k)$, $b_j (1 \leq i \leq m+n)$ are atoms. $c_1 \times \ldots \times c_k \leftarrow b_1, \ldots, b_m,\ not\ b_{m+1}, \ldots,\ not\ b_{m+n}$ is an ordered disjunction rule as defined in [5]. Please observe that an ordered disjunction rule uses the operator $\times$ which means that if possible c_1, but if c_1 is not possible then c_2 and so on. When $k = 1$, the given clause is called a *possibilistic normal rule*. A possibilistic normal program is a finite set of possibilistic normal rules. When $k = 1$ and $n = 0$, the given clause is called *possibilistic definite rule*. A possibilistic definite program is a finite set of possibilistic definite rules. Let $Prog_{\mathcal{L}-(\mathcal{Q},\leq)}$ be the set of all LPPODs with atoms from the signature $\mathcal{L}$ and their possibilistic ordered disjunction rules evaluated in terms of the lattice $(\mathcal{Q}, \leq)$.

In a slight abuse of notation we will denote a possibilistic ordered disjunction rule r by $\alpha : c_1 \times \ldots \times c_k \leftarrow \mathcal{B}^+,\ not\ \mathcal{B}^-$ and sometimes by $\mathcal{C}^\times \leftarrow \mathcal{B}^+,\ not\ \mathcal{B}^-$ where $\mathcal{C}^\times = \{c_1, \ldots, c_k\}$, $\mathcal{B}^+ = \{b_1, \ldots, b_m\}$ and $\mathcal{B}^- = \{b_{m+1}, \ldots, b_{m+n}\}$. $n(r) = \alpha$ is a necessity degree representing the certainty level of the information described by r. According to possibilistic logic, $n(r)$ represents the least certainty value of the knowledge represented by r. The projection $*$ for any possibilistic atom p is defined as $p^* = a$.

A *possibilistic atom* is a pair $p = (a, q) \in \mathcal{A} \times \mathcal{Q}$ where $\mathcal{A}$ is a set of atoms and $(\mathcal{Q}, \leq)$ a finite lattice. Given a set of possibilistic atoms M, the projection of $*$ over M is defined as $M^* = \{p^* \mid p \in M\}$. The projection $*$ for a possibilistic ordered disjunction rule r, is $r^* = c_1 \times \ldots \times c_k \leftarrow \mathcal{B}^+,\ not\ \mathcal{B}^-$ and the projection of $*$ over P is defined as $P^* := \{r^* \mid r \in N\}$. Notice that P^* is an LPOD.

Following the construction of the LPODs semantics [5], the semantics of LPPODs is defined by the following syntactic reductions [7]:

Definition 1 [7] *Let $r = \alpha : c_1 \times \ldots \times c_k \leftarrow \mathcal{B}^+,\ not\ \mathcal{B}^-$ be a possibilistic ordered disjunction rule and M be a set of atoms. The $\times$-possibilistic reduct $r_\times^M$ is defined by*

$$r_\times^M := \{\alpha : c_i \leftarrow \mathcal{B}^+ | c_i \in M \textit{ and } M \cap (\{c_1, \ldots, c_{i-1}\} \cup \mathcal{B}^-) = \emptyset\}$$

Definition 2 [7] *Let $P = \langle(\mathcal{Q}, \leq), N\rangle$ be an LPPOD and M be a set of atoms. The $\times$-possibilistic reduct $P_\times^M$ is defined by*

$$P_\times^M = \bigcup_{r \in N} r_\times^M$$

Please notice that the $\times$-possibilistic reduct reduces an LPPOD to a possibilistic definite logic program. This is an important reduction since the semantics of LPPODs can be computed using the possibilistic consequence operator ΠT_P which maps a set of possibilistic atom to another one until the fix-point ΠCn is found (from possibilistic stable model semantics). Due to lack of space, the complete definition of ΠT_P is omitted and we cross-refer its definition to [7, 16].

Keeping in mind the syntactic reduction $\times$-possibilistic reduct and the fix-point ΠCn, the possibilistic semantics which captures LPPODs is defined as follows:

---

[1] A comprehensive description of these concepts can also be found in [7, 8].

Definition 3 [7] *Let* $P = \langle(\mathcal{Q}, \leq), N\rangle$ *be an LPPOD,* M *be a set of possibilistic atoms such that* M^* *is an answer set of* P^*. M *is a possibilistic answer set of* P *if and only if* $M = \Pi Cn(P_{\times}^{M^*})$. $SEM_{LPPOD}(P)$ *is the mapping which assigns to* P *the set of all possibilistic answer sets of* P *and the LPPODs semantics is denoted by* SEM_{LPPOD}.

Once the possibilistic answer sets of an LPPOD have been identified, it is possible to associate a *satisfaction degree* to each possibilistic answer answer set *w.r.t.* to each possibilistic ordered disjunction rule.

Definition 4 *Let* M *be a possibilistic answer set of an LPPOD* P. *Then* M *satisfies the rule* $r = \alpha : c_1 \times \ldots \times c_k \leftarrow b_1, \ldots, b_m,\ not\ b_{m+1} \ldots,\ not\ b_{m+n}$

- *to degree 1 if* $b_j \notin M$ *for some* j $(1 \leq j \leq m)$*, or* $b_i \in M$ *for some* i $(m + 1 \leq i \leq m + n)$*,*
- *to degree* j $(1 \leq j \leq k)$ *if all* $b_l \in M$ $(1 \leq l \leq m)$*,* $b_i \notin M$ $(m + 1 \leq i \leq m + n)$*, and* $j = min\{r \mid c_r \in M, 1 \leq r \leq k\}$.

By considering this satisfaction degree and necessity values of program clauses, the authors in [7] introduce a possibilistic preference relation for comparing possibilistic answer sets. Such preference relation is defined in the next section.

2.2 Transformation Rules for LPPODs and Possibilistic Preference Relation

In order to define a concrete criterion for selecting possibilistic answer set of an LPPOD, the authors in [7] define a set of basic rewriting rules. These rewriting rules are a generalization of the rewriting rules introduced by Dix *et al.* in [10].

Definition 5 [7] *Let* P *and* P' *be LPPODs. The following possibilistic transformation rules are defined:*

Possibilistic Contra: P' *results from* P *by possibilistic elimination of contradictions* ($P \rightarrow_{PC} P'$) *if* P *contains a rule* $r = \alpha : \mathcal{C}^{\times} \leftarrow \mathcal{B}^{+},\ not\ \mathcal{B}^{-}$ *which has an atom* b *such that* $b \in \mathcal{B}^{+}$ *and* $b \in \mathcal{B}^{-}$*, and* $P' = P \backslash \{r\}$

Possibilistic Positive Reduction: P' *results from* P *by possibilistic positive reduction* $PRED^{+}$ ($P \rightarrow_{PRED^{+}} P'$)*, if there is a rule* $r = \alpha : \mathcal{C}^{\times} \leftarrow \mathcal{B}^{+},\ not\ (\mathcal{B}^{-} \cup \{b\})$ *in* P *and such that* $b \notin HEAD(P)$*, and* $P' = (P \backslash \{r\}) \cup \{\alpha : \mathcal{C}^{\times} \leftarrow \mathcal{B}^{+},\ not\ \mathcal{B}^{-}\}$

Possibilistic Negative Reduction: P' *results from* P *by possibilistic negative reduction* $PRED^{-}$ ($P \rightarrow_{PRED^{-}} P'$)*, if* P *contains the rules* $r = \alpha : a \leftarrow \top$*, and* $r' = \beta : \mathcal{C}^{\times} \leftarrow \mathcal{B}^{+},\ not\ (\mathcal{B}^{-} \cup \{a\})$*, and* $P' = (P \backslash \{r'\})$

Possibilistic Success: P' *results from* P *by possibilistic success* ($P \rightarrow_{PS} P'$)*, if* P *contains a fact* $\alpha : a \leftarrow \top$ *and a rule* $r = \beta : \mathcal{C}^{\times} \leftarrow \mathcal{B}^{+},\ not\ \mathcal{B}^{-}$ *such that* $a \in \mathcal{B}^{+}$*, and* $P' = (P \backslash \{r\}) \cup \{GLB\{\alpha, \beta\} : \mathcal{C}^{\times} \leftarrow (\mathcal{B}^{+} \backslash \{a\}),\ not\ \mathcal{B}^{-}\}$

Possibilistic Failure: P' *results from* P *by possibilistic failure* ($P \rightarrow_{PF} P'$)*, if* P *contains a rule* $r = \alpha : \mathcal{C}^{\times} \leftarrow \mathcal{B}^{+},\ not\ \mathcal{B}^{-}$ *such that* $a \in \mathcal{B}^{+}$ *and* $a \notin HEAD(P)$*, and* $P' = (P \backslash \{r\})$.

Let $\mathcal{CS}_{LPPOD}$ be the rewriting system based on the possibilistic rewriting rules introduced in Definition 5. In [7] it has been shown how these transformations are closed under the LPPODs semantics and that the rewriting system $\mathcal{CS}_{LPPOD}$ guarantees that the normal form of an LPPOD P ($norm_{\mathcal{CS}_{LPPOD}}(P)$) can always be reached and it is always unique. In this way the following preference relation between possibilistic answer sets which considers rules' satisfaction degrees and rules' necessity values at the same time is semantically consistent *w.r.t.* the LPPODs semantics.

Definition 6 [7] *Let* P *be an LPPOD,* M_1 *and* M_2 *be possibilistic answer sets of* P*,* $norm_{\mathcal{CS}_{LPPOD}}(P)$ *be the normal form of* P *w.r.t. the rewriting system* $\mathcal{CS}_{LPPOD}$. M_1 *is possibilistic preferred to* M_2 ($M_1 >_{pp} M_2$) *iff* $\exists\ r \in norm_{\mathcal{CS}_{LPPOD}}(P)$ *such that* $deg_{M_1}(r) < deg_{M_2}(r)$*, and* $\nexists r' \in norm_{\mathcal{CS}_{LPPOD}}(P)$ *such that* $deg_{M_1}(r') > deg_{M_2}(r')$*, and* $n(r) < n(r')$.

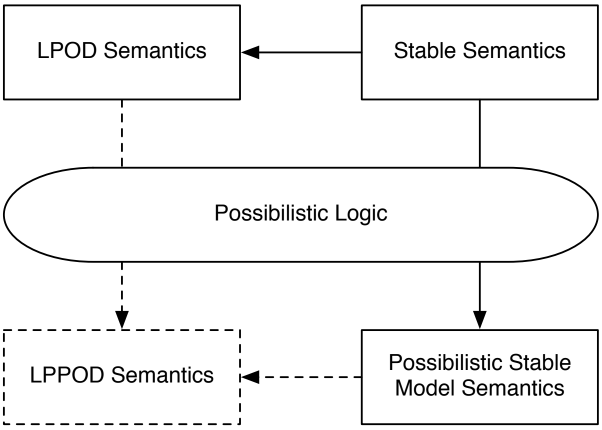

Figure 1: Possibilistic Semantics for LPPODs

3 Computing the LPPODs Semantics

In [7] the semantics for this class of programs has been defined but neither an algorithm nor a complexity study have been discussed and presented and such topics are covered in this technical report. Before doing this, it is necessary to recall an important result obtained when the LPPODs semantics has been defined.

Proposition 1 [7] *Let* $P = \langle(\mathcal{Q}, \leq), N\rangle$ *be an LPPOD and* M *be a possibilistic answer set of* P*, then* M^* *is an answer set of* P^*.

The proposition defines a close relation between the LPPODs semantics and the semantics of its LPODs classical part. The possibilistic answer sets of an LPPOD can in fact be computed by means of the LPODs semantics [5] and the possibilistic stable model semantics [16]. This relationship is shown in Figure 1.

Such result leads to the following theorems which show how the decision problem of existence of a possibilistic answer set of an LPPOD is computable and how it remains in the same complexity class as the decision problem of the existence of an answer set of an LPOD.

Theorem 1 *The LPPOD semantics is computable.*

From computational theory it is well known that a function is computable if it exists an algorithm that can calculate the function's result in a finite number of steps. As the LPPOD semantics maps any LPPOD to the set of its possibilistic answer sets, it can be represented as the function

$$SEM_{LPPOD} : Prog_{\mathcal{L}-(\mathcal{Q},\leq)} \rightarrow 2^{\mathcal{A}\times\mathcal{Q}}$$

Hence, to prove Theorem 1 it is sufficient to find an algorithm that can compute the possibilistic answer sets given an LPPOD P. The algorithm exists and it is described below.

Algorithm 1 SEM_{LPPOD}

```
Input: An LPPOD $P$
Output: Partially Ordered Set of Possibilistic Answer Sets
  $P_{LPOD} \leftarrow P^*$
  $\mathcal{M} \leftarrow SEM_{LPOD}(P_{LPOD})$
  while $\mathcal{M} \neq \emptyset$ do
    $M \leftarrow pop(\mathcal{M})$
    $PM \leftarrow \Pi Cn(P^M_\times)$
    $push(\mathcal{PM}, PM)$
  end while
  $\mathcal{PM} \leftarrow sortPossibilisticModels(\mathcal{PM}, P)$
  $return\ \mathcal{PM}$
```

According to the LPPODs semantics definition (Definition 3) a possibilistic set of atoms cannot be a possibilistic answer set of an LPPOD P, if its projection $*$ is not an answer set of the LPOD P_{LPOD}. Therefore as a first step, the projection $*$ is applied to an LPPOD to obtain the corresponding LPOD. In this way it is possible to use the LPODs semantics (denoted by SEM_{LPOD}) to compute the set of answer sets ($\mathcal{M}$) of P_{LPOD} [5]. As next step, for each answer set $M \in \mathcal{M}$, the possibilistic consequence operator $\Pi T_{P^M_\times}$ is applied to compute the corresponding possibilistic answer set PM of P (given by $Cn(P^M_\times)$). The algorithm always terminates as the consequence operator is monotonic and it always reaches a fix-point (Proposition 8 in [16]).

Another important result that can be obtained is that the complexity of SEM_{LPPOD} remains in the same complexity class of its classical part as stated by the following theorem.

Theorem 2 *The problem of deciding whether an LPPOD has a possibilistic answer set is* $\mathbf{NP}-complete$.

As previously said, the computation of SEM_{LPPOD} is based on the LPODs semantics and on the possibilistic stable models semantics. Thus the SEM_{LPPOD} complexity is related to the complexity of these two semantics. It is known from Theorem 1 in [5] that the complexity of the LPODs semantics is $\mathbf{NP}-complete$. As by the $\times$-possibilistic reduction (Definition 2) we reduce the problem of computing possibilistic answer sets of an LPPOD to the class of possibilistic definite logic programs by means of $Cn(P^{M^*}_\times)$ and by Proposition 9 in [16] we know that this can be done polynomially from the moment an answer set M^* is provided, the complexity of the LPPOD semantics is not significantly harder than the computation of the answer sets for LPODs. Therefore the complexity for SEM_{LPPOD} is $\mathbf{NP}-complete$.

The theorems presented in this section are an important result as they permit to define a straightforward methodology for the LPPODs semantics implementation. We aim to implement the solver in the next future. In the next section we present two possible application areas which show the main features of our approach.

4 Possible Applications

LPPODs are class of logic programs which fit well in the representation of decision problems where a best option has to be chosen taking into account both preference and necessity measures about knowledge. In this section we will present the applicability of LPPODs in two different simple scenarios and how they relate to other ASP-based knowledge representation formalisms such as possibilistic normal logic programs and LPODs.

4.1 Drug Treatment Decision

The first scenario considers the decision problem of a doctor who has to choose between two incompatible drugs when treating one of her patient (taken from [16]).

Example 1 *A patient suffering from two diseases ($di1, di2$) needs a medical treatment. Each disease can be cured by one drug but the two drugs ($dr1, dr2$) are incompatible. If the patient is given drug $dg1$ then she is healed $c1$, while if she receives $dg2$ she will be healed $c2$.*

It would be interesting for the doctor to be able to evaluate what choice to do between these two treatments which are incompatible. In the following we will see how the scenario can be encoded in three different way, possibilistic normal logic programs, LPODs, and LPPODs respectively and we show how LPPODs can effectively help the doctor in making a decision.

In [16] the authors address this problem by means of possibilistic normal logic programs where necessity degrees associated to rules can be be taken into account to determine the level of certainty of each conclusion allowing the doctor to compare them. The representation according to possibilistic normal programs is [16]:

Example 2 *Let us consider a possibilistic normal logic program P modeling the scenario in Example 1.*
$r_1 = \mathbf{1} : dr1 \leftarrow di1,\ not\ dr2.$ $r_4 = \mathbf{0.3} : c2 \leftarrow dr2,\ di2.$
$r_2 = \mathbf{1} : dr2 \leftarrow di2,\ not\ dr1.$ $r_5 = \mathbf{0.9} : di1.$
$r_3 = \mathbf{0.7} : c1 \leftarrow dr1,\ di1.$ $r_6 = \mathbf{0.7} : di2.$

It can be proved that the possibilistic normal logic program P has two possibilistic models $\{(di1, 0.9), (di2, 0.7), (dr1, 0.9), (c1, 0.7)\}$ and $\{(di1, 0.9), (di2, 0.7), (dr2, 0.7), (c2, 0.3)\}$. According to Nicolas *et al.* the doctor should be able to decide which drug to give to the patient based on the necessity values associated to each atoms of the possibilistic models ($dr1$ has a major necessity values than $dr2$, and also its efficiency looks more prominent). But still the doctor is supposed to interpret these results.

An alternative way of representing incompatibility between decisions is by means of qualitative preferences expressed in LPODs' rules. For instance, the doctor decision problem can be captured in LPODs in the following way:

Example 3 *Let us consider an LPOD P' modeling the scenario in Example 1.*
$r_1 = dr1 \times dr2 \leftarrow di1.$ $r_4 = c2 \leftarrow dr2,\ di2.$
$r_2 = dr2 \times dr1 \leftarrow di2$ $r_5 = di1.$
$r_3 = c1 \leftarrow dr1,\ di1.$ $r_6 = di2.$

The LPOD representation leads to two answer sets $\{di1, di2, dr1, c1\}$ and $\{di1, di2, dr2, c2\}$ which happen not to be comparable using any of the preference relations defined in [5]. However, if we consider the LPOD program in Example 3 and associate to each of its program clause the necessity degrees of the possibilistic program in Example 2, we obtain:

Example 4 *Let us consider an LPPOD P'' modeling the scenario in Example 1.*
$r_1 = \mathbf{1} : dr1 \times dr2 \leftarrow di1.$ $r_4 = \mathbf{0.3} : c2 \leftarrow dr2,\ di2.$
$r_2 = \mathbf{1} : dr2 \times dr1 \leftarrow di2.$ $r_5 = \mathbf{0.9} : di1.$
$r_3 = \mathbf{0.7} : c1 \leftarrow dr1,\ di1.$ $r_6 = \mathbf{0.7} : di2.$

As expected the LPPOD program has two possibilistic answer sets, $M_1 = \{(di1, 0.9), (di2, 0.7), (dr1, 0.9), (c1, 0.7)\}$ and $M_2 = \{(di1, 0.9), (di2, 0.7), (dr2, 0.7), (c2, 0.3)\}$, which correspond to the possibilistic answer sets of the possibilistic normal logic program P. Moreover we can observe that M_1^* and M_2^* are answer sets of the LPOD program P' (which is consistent *w.r.t.* Proposition 1). Let us consider the rewriting system $\mathcal{CS}_{\mathcal{LPPOD}}$ defined in Section 2.2. It can be noticed that by applying the *Possibilistic Success* transformation rule we are able to propagates the necessity values of the information about $di1$ and $di2$ to the possibilistic disjunction rules r_1, and r_2. In this way we obtain the normal form of P'' represented by:

Example 5 *Let P''_1 be the LPPOD obtained by $P'' \rightarrow_{PS} P''_1$.*
$r_1 = \mathbf{0.9} : dr1 \times dr2.$ $r_4 = \mathbf{0.3} : c2 \leftarrow dr2,\ di2.$
$r_2 = \mathbf{0.7} : dr2 \times dr1.$ $r_5 = \mathbf{0.9} : di1.$
$r_3 = \mathbf{0.7} : c1 \leftarrow dr1,\ di1.$ $r_6 = \mathbf{0.7} : di2.$

The normal form P''_1 allows to apply the possibilistic preference relation to check whether there is a possibilistic answer set which is the most preferred. Considering rules' satisfaction degrees ($deg_{M_1}(r_1) = 1$, $deg_{M_1}(r_2) = 2$ and $deg_{M_2}(r_1) = 2$, $deg_{M_2}(r_2) = 1$) and rules r_1 and r_2 necessity values ($n(r_1) = 0.9$, $n(r_2) = 0.7$), it is not difficult to see that $M_1 <_{pp} M_2$ as $n(r_1) > n(r_2)$ ($M_2 \not<_{pp} M_1$ follows by Definition 6 as well).

4.2 User Recommendation With Uncertain Information

The second scenario aims to illustrate how the LPPODs solver can be used to implement the reasoning of a simple recommendation agents which takes into account users preferences and certainty about the information discovered in a simple knowledge discovery scenario (Figure 2). The system can be seen as a very simplified version of the system architecture described in [6] whose aim is to provide personalized Semantic web-oriented ubiquitous services to citizens and tourist. The system consists essentially of three types of agents: a *personal assistant agent* collecting and representing user preferences in behalf of the user; a *broker agent* processing user preference to suggest the best recommendation, and a *crawler agent* which retrieves information from an uncertain database. In the following we will see how the scenario can be encoded using LPODs first and then LPPODs.

Let us imagine we want to model the following user recommendation scenario:

Example 6 *A tourist visiting Barcelona is interested in getting some recommendations about restaurants in the city. She normally prefers Mexican to Italian food if she does not have any information about Mexican and Italian restaurants. Otherwise she may prefer to go to either a Mexican or Italian restaurant.*

A natural way to represent user preferences is by means of this LPOD.

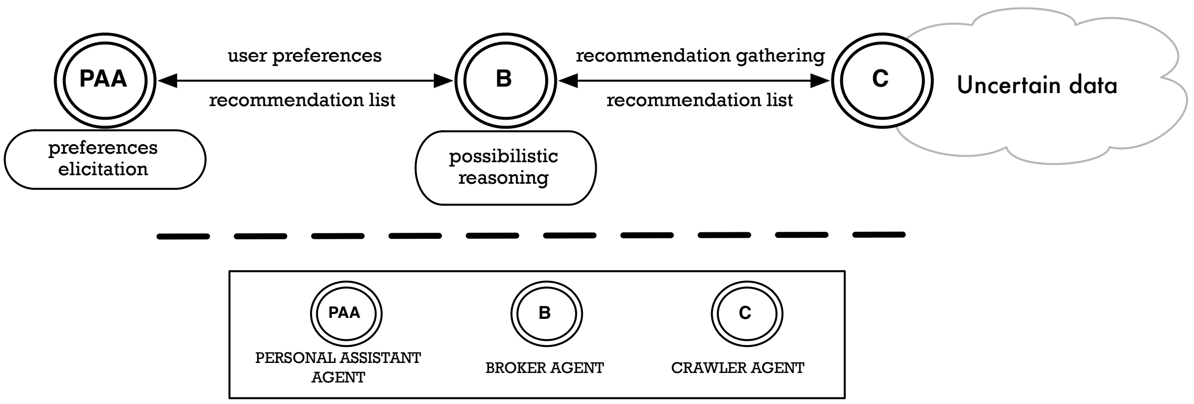

Figure 2: Knowledge Discovery Scenario for User Recommendation

Example 7 *Let us consider an LPOD P modeling the scenario in Example 6.*
$\mathbf{r_1}: mex \times ita \leftarrow \ not\ info(ita),\ not\ info(mex).$
$\mathbf{r_2}: ita \times mex \leftarrow info(ita).$
$\mathbf{r_3}: mex \times ita \leftarrow info(mex).$
$\mathbf{r_4}: \leftarrow mex, ita.$

It can be proved that the LPOD P has two answer sets $M_1 = \{mex\}$ and $M_2 = \{ita\}$, representing the possible choices of the user, and according to the comparison criteria defined in [5] M_1 is preferred to M_2 as expected. However, let us imagine that new information about the restaurants is discovered by the *crawler agent*. Then the LPOD in Example 7 with the added knowledge the *broker agent* has to consider is:

Example 8 *Let us consider the LPOD P' in Example 7 with new knowledge about restaurant added.*
$\mathbf{r_1}: mex \times ita \leftarrow \ not\ info(ita),\ not\ info(mex).$
$\mathbf{r_2}: ita \times mex \leftarrow info(ita).$
$\mathbf{r_3}: mex \times ita \leftarrow info(mex).$
$\mathbf{r_4}: \leftarrow mex, ita.$
$\mathbf{r_5}: info(ita).$
$\mathbf{r_6}: info(mex).$

The LPOD P' still has two answer sets $\{mex, info(mex), info(ita)\}$ and $\{ita, info(mex), info(ita)\}$ but now they are not comparable anymore as the satisfaction degrees prevent from achieving an order. In such a situation, a mechanism that can express the priority between the preference rules of the LPOD is important. The authors in [5] actually propose to introduce meta-preferences by defining a relation $\succ$ on rules to define the preference rules' priority. However, this approach assumes to have a prior knowledge about which rules are more important for the user. A natural question is whether it would be possible to discover the priority of the rules at run-time considering the certainty about the new knowledge.

Hence, if we consider the LPOD program in Example 8 and associate to each of its program clauses necessity degrees as the certainty about the information retrieved by a knowledge discovery process it is possible to induce an order between the rule of the program. Let us imagine that the *crawler agent* retrieves some information about Italian and Mexican restaurants.

Example 9 *Let us consider an LPPOD P'' extending P' with necessity values about the discovered knowledge.*
$r_1 = \mathbf{1}: mex \times ita \leftarrow not\ info(ita),\ not\ info(mex).$
$r_2 = \mathbf{1}: ita \times mex \leftarrow info(ita).$
$r_3 = \mathbf{1}: mex \times ita \leftarrow info(mex).$
$r_4 = \mathbf{1}: \ \leftarrow mex, ita.$
$r_5 = \mathbf{0.8}: info(ita).$
$r_6 = \mathbf{0.5}: info(mex).$

As expected the LPPOD program P'' has two possibilistic answer sets, $M_1 = \{(mex, 0.8), (info(mex), 0.5),$ $(info(ita), 0.8)\}$ and $M_2 = \{(ita, 0.8), (info(mex), 0.5), (info(ita), 0.8)\}$. If we consider the rewriting system $\mathcal{CS}_{\mathcal{LPPOD}}$ and transformation rules $\rightarrow_{PRED^-}$ and $\rightarrow_{PS}$ are applied, the normal form of P'' can be obtained where the necessity values are propagated to the preference rules:

Example 10 *Let P''_2 be the LPPOD obtained by $P'' \rightarrow_{PRED^-} P''_1 \rightarrow_{PS} P''_2$*
$r_2 = \mathbf{0.8} : ita \times mex.$ $\quad r_5 = \mathbf{0.8} : info(ita).$
$r_3 = \mathbf{0.5} : mex \times ita.$ $\quad r_6 = \mathbf{0.5} : info(mex).$
$r_4 = \mathbf{1} : \leftarrow mex, ita.$

The normal form P''_2 allows to use the possibilistic preference relation and considering rules' satisfaction degrees ($deg_{M_1}(r_2) = 2$, $deg_{M_1}(r_3) = 1$ and $deg_{M_2}(r_2) = 1$, $deg_{M_2}(r_3) = 2$) and rules r_2 and r_3 necessity values ($n(r_2) = 0.8$, $n(r_3) = 0.5$), it is not difficult to see that $M_2 <_{pp} M_1$ as $n(r_2) > n(r_3)$ ($M_1 \not<_{pp} M_2$ follows by Definition 6 as well).

5 Conclusions

In this report we have demonstrated that the LPPODs semantics is computable and we the algorithm for its implementation following an ASP approach. We have demonstrated how the complexity of the LPPODs semantics belongs to the same complexity class of its classical part, *i.e.* LPODs. This is an important result as it shows that LPPODs yield a more expressive framework without preventing the semantics to become no computable. We have presented two decision making scenarios in which the solver can be effectively used to choose the best available option taking into account both preferences and priorities about preference rules encoded as necessity values according to possibilistic logic. To the best of our knowledge, no other approaches able to reason about preferences and uncertainty have been proposed yet.

References

[1] M. Balduccini and M. Gelfond. Diagnostic reasoning with A-Prolog. *Theory Practice of Logic Programming*, 3(4):425–461, 2003.

[2] E. Bertino, A. Mileo, and A. Provetti. PDL with Preferences. In *Proc. of the Sixth IEEE Int. Workshop on Policies for Distributed Systems and Networks*, pages 213–222. IEEE Computer Society, 2005.

[3] G. Brewka. Answer Sets and Qualitative Decision Making. In *In Synthese*, volume 146, pages 171–181, 2005.

[4] G. Brewka, S. Benferhat, and D. Le Berre. Qualitative choice logic. *Artificial Intelligence*, 157(1-2):203–237, 2004.

[5] G. Brewka, I. Niemelä, and T. Syrjänen. Logic programs with ordered disjunction. *Computational Intelligence*, 20(2):333–357, 2004.

[6] L. Ceccaroni, V. Codina, M. Palau, and M. Pous. PaTac: Urban, Ubiquitous, Personalized Services for Citizens and Tourists. In *ICDS*, pages 7–12, 2009.

[7] R. Confalonieri, J. C. Nieves, M. Osorio, and J. Vázquez-Salceda. Possibilistic Semantics for Logic Programs with Ordered Disjunction. *Foundations of Information and Knowledge Systems*, 5956:133–152, Feb 2010.

[8] R. Confalonieri, J. C. Nieves, and J. Vázquez-Salceda. Logic Programs with Possibilistic Ordered Disjunction. Research Report LSI-09-19-R, Universitat Politècnica de Catalunya - LSI, 2009.

[9] J. Delgrande, T. Schaub, H. Tompits, and K. Wang. A Classification and Survey of Preference Handling Approaches in Nonmonotonic Reasoning. *Computational Intelligence*, 20(2):308–334, 2004.

[10] J. Dix, M. Osorio, and C. Zepeda. A General Theory of Confluent Rewriting Systems for Logic Programming and its Applications. *Annals of Pure and Applied Logic*, 108(1–3):153–188, 2001.

[11] D. Dubois, J. Lang, and H. Prade. Possibilistic logic. *Handbook of Logic in AI and Logic Programming*, 3:439–513, 1994.

[12] T. Eiter, N. Leone, C. Mateis, G. Pfeifer, T. Wien, and F. Scarcello. The KR System dlv: Progress Report, Comparisons and Benchmarks. In L. S. A.G. Cohn and S. Shapiro, editors, *Proc. of 6th Int. Conf. on Principles of Knowledge Representation and Reasoning*, pages 406–417. Morgan Kaufmann, 1998.

[13] N. Y. Foo, T. Meyer, and G. Brewka. LPOD Answer Sets and Nash Equilibria. In M. J. Maher, editor, *Advances in Computer Science*, volume 3321 of *LNCS*, pages 343–351. Springer, 2004.

[14] M. Gelfond and V. Lifschitz. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing*, 9(3/4):365–386, 1991.

[15] V. Lifschitz. Answer set programming and plan generation. *Artificial Intelligence*, 138(1-2):39 – 54, 2002.

[16] P. Nicolas, L. Garcia, I. Stéphan, and C. Lafèvre. Possibilistic uncertainty handling for answer set programming. *Annals of Mathematics and Artificial Intelligence*, 47:139–181, Nov 2006.

[17] I. Niemelä and P. Simons. Smodels - An Implementation of the Stable Model and Well-Founded Semantics for Normal LP. In *Proc. of the 4th Int. Conf. on LPNMR*, LNCS, pages 421–430, 1997.

[18] T. Soininen and I. Niemelä. Developing a Declarative Rule Language for Applications in Product Configuration. In *Proc. of the 1st Int. Workshop on Practical Aspects of Declarative Languages*, volume 1551 of *LNCS*, pages 305–319, London, UK, 1998. Springer-Verlag.