Annals of Mathematics and Artificial Intelligence manuscript No.
(will be inserted by the editor)

Dealing with Explicit Preferences and Uncertainty in Answer Set Programming

Roberto Confalonieri · Juan Carlos Nieves · Mauricio Osorio · Javier Vázquez-Salceda

Received: date / Accepted: date

Abstract In this paper, we show how the formalism of Logic Programs with Ordered Disjunction (LPODs) and Possibilistic Answer Set Programming (PASP) can be merged into the single framework of Logic Programs with Possibilistic Ordered Disjunction (LPPODs). The LPPODs framework embeds in a unified way several aspects of common-sense reasoning, nonmonotonocity, preferences, and uncertainty, where each part is underpinned by a well established formalism. On one hand, from LPODs it inherits the distinctive feature of expressing context-dependent qualitative preferences among different alternatives (modeled as the atoms of a logic program). On the other hand, PASP allows for qualitative certainty statements about the rules themselves (modeled as necessity values according to possibilistic logic) to be captured. In this way, the LPPODs framework supports a reasoning which is nonmonotonic, preference- and uncertainty-aware. The LPPODs syntax allows for the specification 1) preferences among the exceptions to default rules, and 2) necessity values about the certainty of program rules. As a result, preferences and uncertainty can be used to select the preferred uncertain default rules of an LPPODs and, consequently, to order its possibilistic answer sets. Furthermore, we describe the implementation of an ASP-based solver able to compute the LPPODs semantics.

Keywords Possibilistic Answer Set Programming · Uncertain Nonmonotonic Reasoning · Explicit Preferences

Mathematics Subject Classification (2000) 68T27 · 68T30 · 68T37 · 68N17

R. Confalonieri · J. C. Nieves · J. Vázquez-Salceda
Universitat Politècnica de Catalunya
Dept. Llenguatges i Sistemes Informàtics
C/ Jordi Girona Salgado 1-3
E - 08034 Barcelona
E-mail: {confalonieri,jcnieves,jvazquez}@lsi.upc.edu

M. Osorio
Universidad de las Américas- Puebla
Dept. de Actuaría, Física y Matemáticas
Sta. Catarina Mártir, Cholula
México - 72820 Puebla
E-mail: osoriomauri@googlemail.com

1 Introduction

Answer Set Programming (ASP) is a suitable setting for modeling incomplete information [1]. In ASP, default rules, *i.e.* normative statements such as *normally p's are q's*, are usually modeled by rules of the form $q \leftarrow p, \mathit{not}\ ab$ which contain abnormality atoms (ab) preceded by negation as failure (not). According to this approach, abnormality atoms block the applicability of (default) rules in the reasoning process in a way such that the answer sets of a logic program tend to be those in which everything is as normal as possible [1, 2]. However, when rules are in conflict, that is, they support conflicting conclusions, a criterion for deciding which rules to select is desirable. In nonmonotonic reasoning, the selection of default rules is a matter of preferences [16, 17]. While in some cases preferences can be implicitly obtained by considering the specificity of the defaults [22, 33], most proposals for handling preferences in ASP rely on explicit preferences among rules or atoms of a logic program [26, 47].

Among explicit preference-based approaches, Logic Programs with Ordered Disjunction (LPODs) is a logic programming specification that allows for the specification of conditional preference statements of the form *normally, if c, then a is preferred to b* (encoded as $a \times b \leftarrow c, \mathit{not}\ ab$) [15]. From a nonmonotonic reasoning point of view, these preference statements make it possible to represent a preference order among exceptions *w.r.t.* the default rules in a logic program, and, consequently, to select the preferred default rules to be used in the reasoning [16]. For instance, given two default rules stating that *normally birds can fly* ($f \leftarrow b, \mathit{not}\ \neg f$) and *normally penguins cannot fly* ($\neg f \leftarrow p, \mathit{not}\ f$), then an LPOD (preference) rule stating that *normally, in the case of penguins, it is preferred to assume that birds cannot fly rather than they can* ($\neg f \times f \leftarrow p, \mathit{not}\ b$) can be an effective way to choose the most plausible conclusion (that is, *penguins cannot fly*).

ASP has also been shown to be convenient for modeling uncertain information in terms of possibilistic logic. The pioneer work of Possibilistic Answer Set Programming (PASP) is the first proposal that join nonmonotonic reasoning and possibilistic uncertainty management in ASP [41]. The joint handling of incomplete and uncertain information is an important issue. It has also been addressed in logical-based frameworks [39, 46]. Similar to what happens in nonmonotonic reasoning, uncertain default rules can support conflicting conclusions and a way to select uncertain default rules can be valuable.

The aim of our work is to propose a framework to join together in one single formalism explicit preferences for the selection of default rules and reasoning under uncertainty. We will assume that preference rules about exceptions to default rules that we want to privilege are uncertain. For instance, one may consider that a preference rule stating that *an exception* ab_1 *is preferred over an exception* ab_2 is less certain than another preference rule stating that *an exception* ab_2 *is preferred over an exception* ab_1. In such a case, a mechanism which allows one to consider the most certain preference rule can be useful. To illustrate the main ideas behind our work, let us consider the following example.

Example 1 Let us suppose that we have the following default rules (i) *Antarctic birds are birds*, (ii) *birds normally fly* and (iii) *Antarctic birds normally do not fly.* Furthermore, we have two preference rules which express our preferences among exceptions to the previous rules. In particular, let us suppose that one preference rule states that (iv) *in case of penguins, it is preferred to assume that birds cannot fly rather than Antarctic birds can fly*, and another rule states that (v) *in case of super-penguins, it is preferred to assume that Antarctic birds can fly rather than birds cannot fly.* Finally, we have two further rules expressing that (vi) *Antarctic birds are penguins* and (vii) *Antarctic birds are super-penguins.* The latter rules are uncertain, since we can imagine that in Antarctica there are many penguins that do not fly together with only a few super-penguins that can fly. Let us suppose, for now in an informal way, to associate each rule with a label indicating its certainty. The resulting program is (we also observe an Antarctic bird):

$$P = \left\{ \begin{array}{lllllllll} r_1 = \mathbf{1}: & b & \leftarrow & ant. & r_6 = & \mathbf{1}: & & \leftarrow & ab_1, ab_2. \\ r_2 = \mathbf{1}: & f & \leftarrow & b, not\ ab_1. & r_7 = & \mathbf{0.6}: & p & \leftarrow & ant. \\ r_3 = \mathbf{1}: & \neg f & \leftarrow & ant, not\ ab_2. & r_8 = & \mathbf{0.4}: & sp & \leftarrow & ant. \\ r_4 = \mathbf{1}: & ab_1 \times ab_2 & \leftarrow & p. & r_9 = & \mathbf{1}: & ant & \leftarrow & \top. \\ r_5 = \mathbf{1}: & ab_2 \times ab_1 & \leftarrow & sp. & & & & & \end{array} \right\}$$

The above program without any certainty label is a classical LPOD. In such a case, no choice can be made between the two alternative answer sets, that are, $\{ant, p, sp, b, ab_1, \neg f\}$ and $\{ant, p, sp, b, ab_2, f\}$.[1] In other words, the conflict between the defaults r_2 and r_3 cannot be solved. Intuitively, since the rule r_7 is more certain than the rule r_8, the selection of which exception to privilege, in order to sanction the conflict between the rules r_2 and r_3, should also take into account the certainty about the information encoded in the program. This can be done in principle by considering that the preference rule r_4 is more important than r_5, since it depends on more certain information.

This simple example suggests that in order to select uncertain default rules, and, consequently, to select the most preferred and certain beliefs to be used in the reasoning, not only the preferences over exceptions, but also the certainty encoded in the logic program, can matter. To this end, we extend LPODs in several ways. First, we define a possibilistic semantics which can formally handle uncertainty. Secondly, we study a set of transformation rules able to propagate uncertainty values between the rules and able to preserve the possibilistic semantics.[2] Finally, by taking uncertain preference rules and transformation rules into account, we define a preference relation which can consider preferences and uncertainty in order to select the preferred uncertain default rules and to order possibilistic answer sets. The contributions of this paper are many.

We join the LPODs and PASP frameworks in the formalism of *Logic Programs with Possibilistic Ordered Disjunction* (LPPODs). The LPPODs framework embeds in a unified way several aspects of common-sense reasoning such as nonmonotonicity, explicit preferences, and uncertainty, where each part is underpinned by a well established formalism. We define a syntax which extends LPODs to associate program rules with necessity values for measuring to *what extent atoms and rules are certain.* We define a possibilistic semantics for capturing LPPODs which is a generalization of the LPODs semantics. In this respect, we provide two alternative, but equivalent, characterizations. The first one has a syntactical flavor and it is defined

[1] The LPODs semantics is presented in Section 2.2.

[2] Transformation rules have other nice properties as well as we state later in the paper.

in terms of the possibilistic $\times$-reduction (Definition 7) and the possibilistic consequence operator ΠT for possibilistic definite logic programs [41]. This leads to a straightforward definition of the LPPODs semantics (Definition 10). The second characterization is given in terms of the least possibility distribution of possibilistic definitive logic programs (Definition 11). Similar to what happens in PASP, an LPPOD can be seen as a compact representation of the possibility distribution defined in the interpretations representing the information (Proposition 4).

We propose a set of transformation rules for transforming an LPPOD program. Based on these transformations, we define a rewriting system (Definition 19) and we show how these transformations have noticeable properties. First, they *always* reduce the size of an LPPOD (the rewriting system is noetherian) and they *always* guarantee that a normal form can be reached (the rewriting system is confluent).[3] Most importantly, we show that this normal form is unique and that we can reach it from any LPPOD preserving the LPPODs semantics (Theorem 1). It is worth mentioning that transformation rules have shown to be a solid approach for characterizing different logic programming semantics such as the well-founded semantics [12] and the stable model semantics [10].

The use of transformation rules can be motivated in several ways. First, they make it possible to propagate necessity values between rules. This will be useful in order to define a criterion for comparing possibilistic answer sets which takes preference and uncertainty degrees into account (Definition 20). Secondly, they reduce the size (*i.e.* number or rules and atoms) of an LPPOD. This is important for the computation of the LPPODs semantics. Since the LPPODs complexity is directly proportional to the number of rules in an LPPOD and to the number of certainty levels (Section 6), it is useful to have a mechanism able to reduce the size of an LPPOD. Based on these results, we also present an algorithm for computing the LPPODs semantics and its implementation by means of an ASP-based system called *posPsmodels*.[4]

This paper extends our previous work in [21] by providing an alternative LPPODs semantics definition based on the least possibility distribution, a more detailed description of the transformation rules involved, a semantics implementation and a related work section. The proofs of all the results are also provided (Appendix A). The rest of the paper is structured as follows. After giving some background information on the main concepts involved (Section 2), in Section 3 we define the syntax and semantics of LPPODs. In Section 4, we define a rewriting system for LPPODs. Section 5 describes a strategy for comparing possibilistic answer sets of an LPPOD based on a set of transformation rules and a possibilistic preference relation. In Section 6, we describe the LPPODs algorithm and its implementation. Section 7 is devoted to comparing our framework with other works dealing with uncertainty and preferences. Finally, Section 8 concludes the paper.

2 Background

In this section, we provide all the necessary terminology and relevant definitions in order to make this paper self-contained. In particular, we overview the *answer set semantics* (Section 2.1) and the LPODs formalism (Section 2.2).

[3] A program is said to be in *normal form* when none of the transformations can be applied.

[4] `http://github.com/rconfalonieri/posPsmodels/tarball/master`

2.1 Answer Set Semantics

The language of a propositional logic has an alphabet consisting of (i) proposition symbols: $\top, p_0, p_1, ...$ (ii) connectives : $\vee, \wedge, \leftarrow, \neg$, *not* and (iii) auxiliary symbols: (,) in which $\vee, \wedge, \leftarrow$ are binary-place connectives and $\neg$, *not* are unary-place connectives, The proposition symbols 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 to whether 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 rule, r, is denoted:

$$a_0 \leftarrow a_1, \ldots, a_j, not\ a_{j+1}, \ldots, not\ a_n \tag{1}$$

in which $n \geq 0$ each a_i is an atom[5]. When $n = 0$ the rule is an abbreviation of $a_0 \leftarrow \top$ such that $\top$ is the proposition symbol that always evaluates to true. A constraint is a rule of the form:

$$\leftarrow a_1, \ldots, a_j, not\ a_{j+1}, \ldots, not\ a_n \tag{2}$$

An extended normal program P is a finite set of extended normal rules and constraints. By $\mathcal{L}_P$, we denote the set of atoms in the language of P.

Sometimes, we denote an extended normal rule r by $a_0 \leftarrow \mathcal{B}^+,\ not\ \mathcal{B}^-$; $\mathcal{B}^+$ contains all the positive body literals and $\mathcal{B}^-$ contains all the negative body literals. We denote by $head(r)$ the head a_0 of rule r. When $\mathcal{B}^- = \emptyset$, the rule is called a definite rule. A set of definite rules is called a definite logic program. For managing the constraints in our logic programs, we will replace each rule of the form $\leftarrow \mathcal{B}^+\ not\ \mathcal{B}^-$ by a new rule of the form $f \leftarrow \mathcal{B}^+,\ not\ \mathcal{B}^-,\ not\ f$ such that f is a new atom symbol which does not appear in $\mathcal{L}_P$.

Let A be a set of atoms and P be a (definite or normal) logic program. $r = a_0 \leftarrow \mathcal{B}^+,\ not\ \mathcal{B}^- \in P$ is applicable in A if $\mathcal{B}^+ \subseteq A$. $App(A, P)$ denotes the subset of rules of P which are applicable in A. $r = a_0 \leftarrow \mathcal{B}^+,\ not\ \mathcal{B}^- \in P$ is closed in A if r is applicable in A and $head(r) \in A$.

We will manage the strong negation ($\neg$), in our logic programs, as it is done in ASP [1]. Basically, each extended atom $\neg a$ is replaced by a new atom symbol a' which does not appear in the language of the program and the constraint $\leftarrow a, a'$ is added.

From now on, we assume that the reader is familiar with the notion of an interpretation [25]. It is standard to provide interpretations only in terms of a mapping from $\mathcal{L}_P$ to $\{0, 1\}$. Also, it is standard to use sets of atoms to represent interpretations. The set corresponds exactly to those atoms that evaluate to 1.

An interpretation I is called a (2-valued) model of the logic program P if and only if for each rule $r \in P$, $I(r) = 1$. A theory is consistent if it admits a model, otherwise it is called inconsistent. Given a theory T and a formula φ, we say that φ is a logical consequence of T, denoted by $T \models \varphi$, if every model I of T holds that $I(\varphi) = 1$. We say that a model I of a program P is a minimal model if a model I' of P different from I such that $I' \subset I$ does not exist.

[5] Notice that these atoms can be *extended atoms*.

The answer set semantics was first defined in terms of the so called *Gelfond-Lifschitz reduction* [34] and it is usually studied in the context of syntax dependent transformations on programs. Let P be an extended normal logic program. For any set $S \subseteq \mathcal{L}_P$, let P^S be the definite logic program obtained from P by deleting

(i) each rule that has a formula *not* a in its body with $a \in S$, and then
(ii) all formulæ of the form *not* a in the bodies of the remaining rules.

Clearly P^S does not contain *not*; hence S is called an answer set of P if and only if S is the minimal model of P^S.

2.2 Logic Programs with Ordered Disjunction

The formalism of *Logic Programs with Ordered Disjunction* (LPODs) was created with the idea of expressing explicit context-dependent preference rules in order to select the most plausible atoms to be used in the reasoning and to order answer sets [15].

Technically speaking, LPODs is based on extended logic programs augmented by an ordered disjunction connector $\times$ which allows for the expression of qualitative preferences in the head of rules [15]. An LPOD is a finite collection of rules of the form:

$$r = c_1 \times \ldots \times c_k \leftarrow b_1, \ldots, b_m,\ \mathit{not}\ b_{m+1}, \ldots,\ \mathit{not}\ b_{m+n} \tag{3}$$

where c_i's ($1 \leq i \leq k$) and b_j's ($1 \leq j \leq m+n$) are atoms. The intuitive reading behind a rule like (3) is that if the body of r is satisfied, then some c_i must be true in an answer set, if possible c_1, if c_1 is not possible then c_2, and so on. As previously stated, from a nonmonotonic reasoning point, each of the c_i's can represent alternative ranked options for selecting the most plausible (default) rules of an LPOD.

The LPODs semantics was defined in terms of split programs. Split programs are a way to represent every option of ordered disjunction rules with the property that the set of all answer sets of an LPOD corresponds exactly to the answer sets of the split programs. An alternative and more straightforward characterization of the LPODs semantics was also given in terms of a program reduction defined as follows:

Definition 1 ($\times$-reduction) *[15]* Let $r = c_1 \times \ldots \times c_k \leftarrow b_1, \ldots, b_m,\ \mathit{not}\ b_{m+1}, \ldots,\ \mathit{not}\ b_{m+n}$ be an ordered disjunction rule and M be a set of atoms. The $\times$-reduction of a rule r is defined as:

$$r_\times^M = \{c_i \leftarrow b_1, \ldots, b_m | c_i \in M \wedge M \cap (\{c_1, \ldots, c_{i-1}\} \cup \{b_{m+1}, \ldots, b_{m+n}\}) = \emptyset\}$$

The $\times$-reduction is generalized to an LPOD P in the following way:

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

Based on the $\times$-reduction, the LPODs semantics is defined by the following definition:

Definition 2 (*SEM_{LPOD}*) *[15]* Let P be an LPOD and M be a set of atoms. Then, M is an answer set of P if and only if M is closed under all the rules in P and M is the minimal model of $P^M_{\times}$. We denote by $SEM_{LPOD}(P)$ the set of answer sets of P.

One interesting characteristic of LPODs is that they provide a means to represent preferences among answer sets by considering the satisfaction degree of an answer set *w.r.t.* a rule [15].

Definition 3 (Rule Satisfaction Degree) *[15]* Let M be an answer set of an LPOD P. The satisfaction degree M *w.r.t.* a rule $r = c_1 \times \ldots \times c_k \leftarrow b_1, \ldots, b_m$, $not\ b_{m+1} \ldots$, $not\ b_{m+n}$, denoted by $deg_M(r)$, is

- 1 if $b_j \not\in M$ for some j $(1 \leq j \leq m)$, or $b_i \in M$ for some i $(m+1 \leq i \leq m+n)$,
- j $(1 \leq j \leq k)$ if all $b_l \in M$ $(1 \leq l \leq m)$, $b_i \not\in M$ $(m+1 \leq i \leq m+n)$, and $j = min\{r \mid c_r \in M, 1 \leq r \leq k\}$.

The degrees can be viewed as penalties, as a higher degree expresses a lesser degree of satisfaction. Therefore, if the body of a rule is not satisfied, then there is no reason to be dissatisfied and the best possible degree 1 is obtained [15]. A preference order on the answer sets of an LPOD can be obtained by means of the following preference relation.

Definition 4 *[15]* M_1 is preferred to M_2 (denoted by $M_1 >_p M_2$) if and only if $\exists\ r \in P'$ such that $deg_{M_1}(r) < deg_{M_2}(r)$ and $\nexists r' \in P'$ such that $deg_{M_2}(r') < deg_{M_1}(r')$.

To illustrate the LPODs semantics and the preference relation, let us consider the following example.

Example 2 Let us consider the logic program P presented in the introduction (Example 1) without certainty labels:

$$P = \left\{ \begin{array}{lcl lcl} r_1 = & b & \leftarrow \quad ant. & r_6 = & & \leftarrow ab_1, ab_2. \\ r_2 = & f & \leftarrow \quad b, not\ ab_1. & r_7 = & p & \leftarrow \quad ant. \\ r_3 = & \neg f & \leftarrow ant, not\ ab_2. & r_8 = & sp & \leftarrow \quad ant. \\ r_4 = & ab_1 \times ab_2 & \leftarrow \quad p. & r_9 = & ant & \leftarrow \quad \top. \\ r_5 = & ab_2 \times ab_1 & \leftarrow \quad sp. & & & \end{array} \right\}$$

Given the set of atoms $M_1 = \{ant, p, sp, b, ab_1, \neg f\}$ and $M_2 = \{ant, p, sp, b, ab_2, f\}$, by Definition 2, it can be seen that P is reduced into two programs $P^{M_1}_{\times}$ and $P^{M_2}_{\times}$ which have exactly answer sets M_1 and M_2. Let us consider the satisfaction degrees of M_1 and M_2 *w.r.t.* r_4 and r_5 which are $deg_{M_1}(r_4) = 1$, $deg_{M_1}(r_5) = 2$, $deg_{M_1}(r_4) = 2$, $deg_{M_1}(r_5) = 1$. It is easy to see that, according to Definition 4, neither M_1 is preferred to M_2, nor M_2 is preferred to M_1.

In this paper, we will show that, by combining LPODs and PASP and by means of a set of transformation rules, the preference relation in Definition 4 can be extended to consider the necessity values associated with program rules.

3 Logic Programs with Possibilistic Ordered Disjunction

In order to associate certainty labels with LPODs rules, we join LPODs and PASP. In this section, we propose a syntax and a semantics able to capture Logic Programs with Possibilistic Ordered Disjunction (LPPODs). Similar to what happens in PASP, an LPPOD can be seen as a compact representation of the possibility distribution defined in the interpretations representing the information. Therefore, we are able to provide two equivalent characterizations of the LPPODs semantics: a *syntactical characterization* based on a fix-point operator (Definition 10) and a *semantical characterization* defined in terms of the least possibility distribution (Proposition 4).

3.1 Syntax

Our syntax is based on the syntax of LPODs and PASP. A *possibilistic atom* is a pair $p = (a, \alpha) \in \mathcal{A} \times \mathcal{S}$ in which $\mathcal{A}$ is a set of atoms and $\mathcal{S} \subseteq (0, 1]$ is a finite totally ordered set of necessity values. The projection $*$ for any possibilistic atom p is defined as $p^* = a$. Given a set of possibilistic atoms M, the projection of $*$ over M is defined as $M^* = \{p^* \mid p \in M\}$. A possibilistic ordered disjunction rule r is of the form:

$$\alpha : c_1 \times \ldots \times c_k \leftarrow b_1, \ldots, b_m,\ not\ b_{m+1}, \ldots,\ not\ b_{m+n} \tag{4}$$

where $\alpha \in \mathcal{S}$ and $c_1 \times \ldots \times c_k \leftarrow b_1, \ldots, b_m,\ not\ b_{m+1}, \ldots,\ not\ b_{m+n}$ is an ordered disjunction rule as defined in Section 2.2.

The projection $*$ for a possibilistic ordered disjunction rule r is $r^* = c_1 \times \ldots \times c_k \leftarrow b_1, \ldots, b_m,\ not\ b_{m+1}, \ldots,\ not\ b_{m+n}$. $Nec(r) = \alpha$ is a necessity degree representing the certainty of the information described by r. A possibilistic constraint c is of the form $\mathbf{1} : \leftarrow b_1, \ldots, b_m,\ not\ b_{m+1}, \ldots,\ not\ b_{m+n}$. Observe that the necessity value of a possibilistic constraint is $\mathbf{1}$ since, like constraints in standard ASP, the purpose of the possibilistic constraint is to eliminate possibilistic answer sets. Hence, it is assumed that there is no uncertainty about the information captured by a possibilistic constraint. The projection $*$ for a possibilistic constraint c is $c^* = \leftarrow b_1, \ldots, b_m,\ not\ b_{m+1}, \ldots,\ not\ b_{m+n}$.

A *Logic Program with Possibilistic Ordered Disjunction* (LPPOD) is a finite set of possibilistic ordered disjunction rules and/or possibilistic constraints. If an LPPOD does not contain possibilistic ordered disjunction rules then it is known as a *possibilistic normal logic program*. If a possibilistic normal logic program does not contain negation as failure *not* then it is a *possibilistic definite logic program*. The projection of $*$ over P is defined as $P^* = \{r^* \mid r \in N\}$. Please notice that P^* is an LPOD.

Example 3 We are now able to represent our introductory program as described before. By using the following certainty scale: **1** is *absolutely certain*, **0.9** is *quasi certain*, **0.6** is *almost certain*, and **0.4** is *little certain*, we can associate a certainty degree to each rule:

$$P = \left\{ \begin{array}{lllll} r_1 = \mathbf{1}: & b & \leftarrow & ant. & r_6 = \mathbf{1}: \quad \leftarrow ab_1, ab_2. \\ r_2 = \mathbf{0.9}: & f & \leftarrow & b, not\ ab_1. & r_7 = \mathbf{0.6}: \ p \leftarrow \ ant. \\ r_3 = \mathbf{0.6}: & \neg f & \leftarrow & ant, not\ ab_2. & r_8 = \mathbf{0.4}: \ sp \leftarrow \ ant. \\ r_4 = \mathbf{1}: & ab_1 \times ab_2 & \leftarrow & p. & r_9 = \mathbf{1}: \ ant \leftarrow \ \top. \\ r_5 = \mathbf{1}: & ab_2 \times ab_1 & \leftarrow & sp. & \end{array} \right\}$$

The first rule states that we are *absolutely certain* that an Antarctic bird is a bird (r_1). Then, we are *quasi certain* that birds normally fly (r_2), while we are *almost certain* that Antarctic birds normally do not fly (r_3). We also consider that rule r_4, *i.e.* being a penguin it is more reasonable to assume that Antarctic birds cannot fly rather than that they can fly, and rule r_5, *i.e.* being a super-penguin it is more reasonable to assume that Antarctic birds can fly rather than that they cannot fly, are *absolutely certain.* However, r_7 and r_8 express that we are *almost certain* that an Antarctic bird is a penguin and that we are *little certain* that an Antarctic bird is a super-penguin. The last rule (r_9) indicates that we are *absolutely certain* that we are observing an Antarctic bird.

Certainty values induce a certainty scale which has to be taken into account when inferring the possibilistic answer sets of an LPPOD. In the next sections, we show how it is possible to characterize the LPPODs semantics in two different ways which lead to the same possibilistic answer sets.

3.2 Possibilistic Answer Sets of LPPODs by Fix-Point

The first characterization of the LPPODs semantics is in terms of the *possibilistic least model* for possibilistic definite logic programs [41]. Before presenting this characterization, we introduce some relevant concepts. As we are dealing with possibilistic atoms, the basic relations between ordinary sets have to be extended to be able to consider possibilistic atoms.

Given a finite set of atoms $\mathcal{A}$ and $\mathcal{S} \subseteq (0,1]$, the finite set of all the possibilistic atom sets induced by $\mathcal{A}$ and $\mathcal{S}$ is denoted by $\mathcal{PS}' = 2^{\mathcal{A}\times\mathcal{S}}$ and $\mathcal{PS} = \mathcal{PS}' \setminus \{A | A \in \mathcal{PS}$ such that $x \in \mathcal{A}$ and $Cardinality(\{(x,\alpha)|(x,\alpha) \in A\}) \geq 2\}$[6] Informally speaking, $\mathcal{PS}$ is the subset of $\mathcal{PS}'$ such that each $A \in \mathcal{PS}$ has no atoms with different uncertain value.

Definition 5 *[41]* Given $A, B \in \mathcal{PS}$ the relations $\sqcap$, $\sqcup$ and $\sqsubseteq$ between sets of possibilistic atoms are defined as follows:

$$\begin{array}{ll} A \sqcap B & = \{(a, \min\{\alpha,\beta\}) | (a,\alpha) \in A \ \wedge (a,\beta) \in B\} \\ A \sqcup B & = \{(a,\alpha)|(a,\alpha) \in A \ \wedge \ a \notin B^*\} \cup \{(a,\beta)|a \notin A^* \ \wedge \ (a,\beta) \in B\} \cup \\ & \quad \{(a, \max\{\alpha,\beta\})(a,\alpha) \in A \ \wedge \ (a,\beta) \in B\}. \\ A \sqsubseteq B & \iff A^* \subseteq B^* \wedge \forall a,\alpha,\beta, (a,\alpha) \in A \wedge (a,\beta) \in B \ \text{ then } \alpha \leq \beta. \end{array}$$

The LPPODs semantics is based on a possibilistic fix-point operator which was introduced in [41] by considering possibilistic definite logic programs. To be able to reuse such result, we have first defined a syntactic reduction for LPPODs. It extends the LPODs reduction by taking the necessity values of the rules into account.

[6] *Cardinality* is a function which returns the cardinality of a set.

Definition 6 (Possibilistic Reduction $r_\times^M$) Let $r = \alpha : c_1 \times \ldots \times c_k \leftarrow b_1, \ldots, b_m$, *not* $b_{m+1}, \ldots,$ *not* b_{m+n} be a possibilistic ordered disjunction rule and M be a set of atoms. The $\times$-possibilistic reduction $r_\times^M$ is defined as follows:

$$r_\times^M = \{\alpha : c_i \leftarrow \{b_1, \ldots, b_m\} | c_i \in M \wedge M \cap (\{c_1, \ldots, c_{i-1}\} \cup \{b_{m+1}, \ldots, b_{m+n}\}) = \emptyset\}$$

Definition 7 (Possibilistic Reduction $P_\times^M$) Let P be an LPPOD and M be a set of atoms. The $\times$-possibilistic reduction $P_\times^M$ is defined as follows:

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

Example 4 Let P be the LPPOD in Example 3 and let us consider the following sets of atoms $M_1^* = \{ant, p, sp, ab_1, b, \neg f\}$ and $M_2^* = \{ant, p, sp, ab_2, b, f\}$. We can see that:[7]

$$P_\times^{M_1^*} = \left\{ \begin{array}{l} r_1 = \mathbf{1} : \; b \leftarrow ant. \\ r_2 = \mathbf{0.9} : \; f \leftarrow b. \\ r_4 = \mathbf{1} : \; ab_1 \leftarrow p. \\ r_5 = \mathbf{1} : \; ab_1 \leftarrow sp. \\ r_7 = \mathbf{0.6} : \; p \leftarrow ant. \\ r_8 = \mathbf{0.4} : \; sp \leftarrow ant. \\ r_9 = \mathbf{1} : \; ant \leftarrow \top. \end{array} \right\} \quad P_\times^{M_2^*} = \left\{ \begin{array}{l} r_1 = \mathbf{1} : \; b \leftarrow ant. \\ r_3 = \mathbf{0.6} : \; \neg f \leftarrow ant. \\ r_4 = \mathbf{1} : \; ab_2 \leftarrow p. \\ r_5 = \mathbf{1} : \; ab_2 \leftarrow sp. \\ r_7 = \mathbf{0.6} : \; p \leftarrow ant. \\ r_8 = \mathbf{0.4} : \; sp \leftarrow ant. \\ r_9 = \mathbf{1} : \; ant \leftarrow \top. \end{array} \right\}$$

Please observe that the programs $P_\times^{M_1^*}$ and $P_\times^{M_2^*}$ are possibilistic definite logic programs.

In general, once an LPPOD P has been reduced by a set of atoms M^*, it is possible to test whether M is a possibilistic answer set of the program P by considering the possibilistic least model ΠCn of P. ΠCn is defined in terms of the β-applicability of a rule *w.r.t.* a set of possibilistic atoms and in terms of the possibilistic consequence operator ΠT. In order to define ΠCn, let us introduce some basic definitions. Given a possibilistic definite logic program P and $a \in \mathcal{L}_{P^*}$, $head(P, a) = \{r | r \in P \text{ and } head(r^*) = a\}$, and $head(P^*) = \{head(r^*) | r^* \in P^*\}$.

Definition 8 (β-applicability) *[41]* Let r be a possibilistic definite rule of the form $\alpha : c \leftarrow b_1, \ldots, b_m$ and M be a set of possibilistic atoms,

– r is β-applicable in M if there are $\alpha_1, \ldots, \alpha_n$ such that $\{(b_1, \alpha_1), \ldots, (b_m, \alpha_n)\} \sqsubseteq M$ and $\beta = \min\{\alpha, \alpha_1, \ldots, \alpha_n\}$.
– r is 0-applicable otherwise.

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

$$App(P, M, a) = \{r \in head(P, a) | r \text{ is } \beta\text{-applicable in } M \text{ and } \beta > 0\}$$

Given a set of possibilistic atoms M, the applicability degree β of a rule β-applicable in M captures the certainty of the conclusion that the rule can produce *w.r.t.* M. If the body is empty, then the rule is applicable with its own certainty degree. If the body is not satisfied by M, then the rule is not applicable at all. Otherwise, the applicability level of the rule depends on the certainty level of the propositions.

[7] r_6 is deleted by our possibilistic reduction by the way in which we manage possibilistic constraints in our LPPODs (see Section 3.1).

Example 5 Let P_1 be the possibilistic definite logic program in Example 4 obtained by the possibilistic reduction $P_{\times}^{M_1^*}$. We can see that if we consider $M = \emptyset$ and the atom *ant*, then r_9 is 1-applicable in M. In fact, we can see that $App(P, \{\emptyset\}, ant) = \{r_5\}$. Also, we can see that if $M = \{(ant, 1)\}$ and the atoms p and sp are considered, then r_7 and r_8 are 0.6-applicable and 0.4-applicable in M respectively.

Based on the β-applicability of a rule *w.r.t.* a set of possibilistic atoms (Definition 8), the consequence operator ΠT_P for logic programs has been extended in the following way:

Definition 9 (Possibilistic consequence operator ΠT_P) *[41]* Let P be a possibilistic definite logic program and M be a set of possibilistic atoms. The immediate possibilistic consequence operator ΠT_P maps a set of possibilistic atoms to another one as follows:

$$\begin{aligned}\Pi T_P(M) = \quad & \{(a,\delta) | \varphi \in head(P^*), App(P, M, a) \neq \emptyset, \\ & \quad \delta = \max_{r \in App(P,M,a)}\{\beta | r \text{ is } \beta\text{-applicable in } M\}\}\end{aligned}$$

Then the iterated operator ΠT_P^k is defined by

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

ΠT_P is a monotonic operator and it always reaches a fix-point. The possibilistic answer set of a possibilistic definite logic program is characterized in the following way.

Proposition 1 [41] *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 is called the set of possibilistic consequences of P and it is denoted by $\Pi Cn(P)$.*

By considering the fix-point operator $\Pi Cn(P)$ and the possibilistic reduction $P_{\times}^{M}$, the LPPODs semantics is related to the LPODs semantics and the possibilistic answer set semantics. This directly leads to the following definition.

Definition 10 (Possibilistic Answer Set for LPPODs) Let P 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 $M = \Pi Cn(P_{\times}^{M^*})$. We denote by $SEM_{LPPOD}(P)$ the set of all possibilistic answer sets of P and by SEM_{LPPOD} the LPPOD semantics.

Example 6 Let P be the LPPOD introduced in Example 3 and let us consider the sets of atoms $M_1^* {=} \{ant, p, sp, ab_1, b, \neg f\}$ and $M_2^* = \{ant, p, sp, ab_2, b, f\}$. In order to infer the possibilistic answer sets of P, we have to infer the answer sets of the LPOD P^*. It can be proven that both M_1^* and M_2^* are valid answer sets of P^*. As seen in Example 4, $P_{\times}^{M_1^*}$ and $P_{\times}^{M_2^*}$ are possibilistic definite logic programs. Therefore, the possibilistic answer sets of P are:

$$\Pi Cn(P_{\times}^{M_1^*}) = \{(ant, 1), (p, 0.6), (sp, 0.4), (b, 1), (\neg f, 0.9), (ab_1, 0.6)\}$$

since

$$
\begin{aligned}
\Pi T^0_{P_\times^{M_1^*}} &= \emptyset \\
\Pi T^1_{P_\times^{M_1^*}} &= \Pi T_{P_\times^{M_1^*}}(\emptyset) = \{(ant,1)\} \\
\Pi T^2_{P_\times^{M_1^*}} &= \Pi T_{P_\times^{M_1^*}}(\{(ant,1)\}) = \{(ant,1),(p,0.6),(sp,0.4),(b,1)\} \\
\Pi T^3_{P_\times^{M_1^*}} &= \Pi T_{P_\times^{M_1^*}}(\{(ant,1),(p,0.6),(sp,0.4),(b,1)\}) = \\
&\qquad \{(ant,1),(p,0.6),(sp,0.4),(b,1),(\neg f,0.9),(ab_1,0.6)\} \\
\Pi T^4_{P_\times^{M_1^*}} &= \Pi T_{P_\times^{M_1^*}}(\{(ant,1),(p,0.6),(sp,0.4),(b,1),(\neg f,0.9),(ab_1,0.6)\}) = \\
&\qquad \{(ant,1),(p,0.6),(sp,0.4),(b,1),(\neg f,0.9),(ab_1,0.6)\} \\
\Pi T^{k+1}_{P_\times^{M_1^*}} &= \Pi T^k_{P_\times^{M_1^*}}, \forall k > 3
\end{aligned}
$$

The computation of $Cn(P_\times^{M_2^*})$ can be understood in a similar way and it leads to the possibilistic answer set $M_2 = \{(ant,1),(p,0.6),(sp,0.4),(b,1),(f,0.6),(ab_2,0.6)\}$.

From Definition 10, we can observe that there is an important condition *w.r.t.* the definition of a *possibilistic answer set* of an LPPOD: a possibilistic set M cannot be a possibilistic answer set of an LPPOD P, if M^* is not an answer set of an LPOD P^*. Hence an important relation between the possibilistic semantics of LPPODs and the semantics of LPODs can be formalized by the following proposition.

Proposition 2 (Relation between Semantics) *Let P be an LPPOD and M be a set of possibilistic atoms. If M is a possibilistic answer set of P, then M^* is an answer set of P^*.*

When all the possibilistic rules of an LPPOD P have the necessity value of $\mathbf{1}$, each answer set of P^* can be directly generalized to be a possibilistic answer sets of P.

Proposition 3 (Generalization) *Let P be an LPPOD and M^* is an answer set of P^*. If $\forall r \in P$, $Nec(r) = \mathbf{1}$, then $M = \{(a,1) \mid a \in M^*\}$ is a possibilistic answer set of P.*

An alternative characterization of the LPPODs semantics can be given by the least possibilistic distribution associated with a possibilistic definite logic program as will be discussed in the next section.

3.3 Possibilistic Answer Sets of LPPODs by Possibility Distribution

Definition 10 proposes a syntactical way to compute the possibilistic answer sets of an LPPOD by means of the possibilistic $\times$-reduction (Definition 7) and the fix-point operator ΠCn (Proposition 1). Similar to what has been done in [41] for possibilistic normal programs, we characterize the LPPODs semantics in terms of the least specific distribution for possibilistic definite logic programs.

Definition 11 *[41]* Let P be a possibilistic definite logic program, M be a set of atoms, and $\pi_P : 2^{\mathcal{L}_P} \longmapsto [0,1]$ be a possibility distribution. π_P is compatible with P and it is the least specific distribution for P if:[8]

[8] A definite logic program P^* is said to be grounded if it can be ordered as a sequence $\langle r_1, \ldots, r_n \rangle$ such that $\forall i$, $1 \leq i \leq n$, $r_i \in App(P, head(\{r_1, \ldots, r_{i-1}\}))$.

$$\forall M \in 2^{\mathcal{L}_P} \begin{cases} \pi_P(M) = 0, & M \nsubseteq head(App(P^*, M)) \\ \pi_P(M) = 0, & App(P^*, M) \text{ not grounded} \\ \pi_P(M) = 1, & M \text{ is an answer of } P^* \\ \pi_P(M) = 1 - \max_{r \in N}\{Nec(r) \mid M \nvDash r^*\}, & \text{otherwise} \end{cases}$$

The possibility distribution π_P induces two dual measures which can be used to infer in the ASP framework the necessity degrees of each atom of a possibilistic definite logic program.

Definition 12 *[41]* Let P be a possibilistic definite logic program and π_P its associated possibility distribution. The possibility and necessity measures are defined as:

- $\Pi_P(a) = max_{M \in 2^{\mathcal{L}_P}}\{\pi_P(M) \mid a \in M\}$
- $N_P(a) = 1 - max_{M \in 2^{\mathcal{L}_P}}\{\pi_P(M) \mid a \notin M\}$

Given the above definitions and the $\times$-possibilistic reduction $P_\times^M$, it is possible to give a characterization of a possibilistic answer set of an LPPOD in terms of the possibility distribution for possibilistic definite logic programs.

Proposition 4 *Let P be an LPPOD and M be a set of possibilistic atoms such that M^* is an answer set of P^*, then*

$$\Pi M(P_\times^{M^*}) = \{(a, N_{P_\times^{M^*}}(a)) | a \in \mathcal{L}_{P_\times^{M^*}}, N_{P_\times^{M^*}}(a) > 0\}$$

is a possibilistic answer set of P.

Example 7 Let P be the LPPOD in Example 3, $M_1^*=\{ant, p, sp, ab_1, b, \neg f\}$, $M_2^* = \{ant, p, sp, ab_2, b, f\}$ be the answer sets of P^*, and $P_\times^{M_1^*}$, $P_\times^{M_2^*}$ be the reductions of P *w.r.t.* M_1^* and M_2^* respectively (Example 4). Let us consider the possibility distribution π_P for a possibilistic definite logic program in Definition 11 and let $\mathcal{L}_{P_\times^{M_1^*}} = \{ant, p, sp, ab_1, b, \neg f\} = S$. The least specific possibility distribution induced by $P_\times^{M_1^*}$ on 2^S is:

- $\pi_{P_\times^{M_1^*}}(\{ant, b\}) = 1 - max\{0.9, 0.6, 0.4\} = 0.1$
- $\pi_{P_\times^{M_1^*}}(\{ant, b, \neg f\}) = 1 - max\{0.6, 0.4\} = 0.4$
- $\pi_{P_\times^{M_1^*}}(\{ab_1, ant, b, p\}) = 1 - max\{0.4, 0.9\} = 0.1$
- $\pi_{P_\times^{M_1^*}}(\{ab_1, ant, b, sp\}) = 1 - max\{0.6, 0.9\} = 0.1$
- $\pi_{P_\times^{M_1^*}}(\{ab_1, ant, b, \neg f, p\}) = 1 - max\{0.6\} = 0.4$
- $\pi_{P_\times^{M_1^*}}(\{ab_1, ant, b, \neg f, p\}) = 1 - max\{0.4\} = 0.6$
- $\pi_{P_\times^{M_1^*}}(\{ab_1, ant, b, \neg f, p, b\}) = 1$ (the answer set)
- for all the other sets $S' \in 2^S$, $\pi_{P_\times^{M_1^*}}(S') = 0$

As all the atoms in the signature of $P_\times^{M_1^*}$ belong to the answer set of P^*, their level of consistency with respect to $P_\times^{M_1^*}$ is the most possible value. As expected, the possibility measures associated with each of the atoms are:

$$\begin{array}{cc} \Pi_{P_\times^{M_1^*}}(ant) = 1 & \Pi_{P_\times^{M_1^*}}(ab_1) = 1 \\ \Pi_{P_\times^{M_1^*}}(p) = 1 & \Pi_{P_\times^{M_1^*}}(b) = 1 \\ \Pi_{P_\times^{M_1^*}}(sp) = 1 & \Pi_{P_\times^{M_1^*}}(\neg f) = 1 \end{array}$$

Instead, necessity measures evaluate the certainty level at which each atom is inferred from $P_\times^{M_1^*}$. As expected, necessity values inferred by the possibility distribution $\pi_{P_\times^{M_1^*}}$ are consistent with the necessity values associated to each possibilistic atom in the possibilistic answer set of M_1 of P which was inferred by the fix-point in Example 6.

$$\begin{array}{cc} N_{P_\times^{M_1^*}}(ant) = 1 & N_{P_\times^{M_1^*}}(ab_1) = 0.6 \\ N_{P_\times^{M_1^*}}(p) = 0.6 & N_{P_\times^{M_1^*}}(b) = 1 \\ N_{P_\times^{M_1^*}}(sp) = 0.4 & N_{P_\times^{M_1^*}}(\neg f) = 0.9 \end{array}$$

The inference of the necessity values related to $M_2^* = \{ant, p, sp, ab_2, b, f\}$ in the possibilistic definite logic program $P_\times^{M_2^*}$ can be understood in a similar way.

Therefore, the LPPODs semantics can be defined by two equivalent characterizations: the syntactical characterization based on a fix-point operator and the semantical characterization based on the possibility distribution for possibilistic definite logic programs. For the sake of computation, in Section 6, we will see how the syntactical characterization provides a straightforward methodology for the LPPODs semantics computation.

In the next section, we present a set of transformation rules which allow one to propagate necessity values between rules while preserving the LPPODs semantics. These results are important for the definition of a comparison criterion. Based on these transformations, in Section 5, we will define a comparison criterion for selecting possibilistic answer sets which can consider both *necessity values* and *satisfaction degrees* of possibilistic ordered disjunction rules.

4 Transformation Rules for LPPODs

Generally speaking, a *transformation rule* is a syntactic rule which specifies the conditions under which a logic program P can be transformed into another logic program P'. In the logic programming literature, transformation rules have been studied for several classes of logic programs in order to characterize and infer logic programming semantics [10, 28]. A common requirement is that the transformations can be used to reduce the size of a logic program provided that they do not affect its semantics. In this context, several notions of equivalence between logic programs have been defined [37]. We generalize some basic transformations of normal programs (EC, RED^+, RED^-, *Success*, *Failure*, *Loop*) to LPPODs and we show how these transformations can reduce the structure of an LPPOD to a normal form without affecting the LPPODs semantics. To this end, we first prove that the transformations preserve the LPPODs semantics and we reuse the theory of rewriting system [27] to guarantee that the normal form is always unique.

In this section, we denote a possibilistic ordered disjunction rule r (of the form expressed by 4) by $\alpha : \mathcal{C}^\times \leftarrow \mathcal{B}^+,\ not\ \mathcal{B}^-$, in which $\mathcal{C}^\times = \{c_1, \ldots, c_k\}$,

$\mathcal{B}^+ = \{b_1, \dots, b_m\}$, and $\mathcal{B}^- = \{b_{m+1}, \dots, b_{m+n}\}$. We write $HEAD(P)$ for the set of all atoms occurring in rule heads of an LPPOD P. In the following, we state that two LPPODs P and P' are *equivalent w.r.t.* the LPPODs semantics (denoted $SEM_{LPPOD}(P) \equiv SEM_{LPPOD}(P')$) if they possess the same possibilistic answer sets.

We refer to a rewriting rule as a program transformation as defined in [44]. A program transformation $\rightarrow$ is a binary relation on $Prog_{\langle \mathcal{A}, \mathcal{S} \rangle}$ where $Prog_{\langle \mathcal{A}, \mathcal{S} \rangle}$ is the set of all LPPODs with atoms from the signature $\mathcal{A}$ and necessity values from $\mathcal{S} \subseteq (0, 1]$. A program transformation $\rightarrow$ maps an LPPOD P to another LPPOD P'. We use $P \rightarrow_T P'$ to denote that we get P' from P by applying a transformation rule T to P.

We illustrate the transformations by means of the following running example:

Example 8

$$P_0 = \left\{ \begin{array}{lrcl} r_1 = \mathbf{1}: & a \times b & \leftarrow & not\ c, not\ d. \\ r_2 = \mathbf{1}: & c \times d & \leftarrow & e, not\ e. \\ r_3 = \mathbf{1}: & b \times a & \leftarrow & c. \\ r_4 = \mathbf{1}: & a \times b & \leftarrow & d. \\ r_5 = \mathbf{1}: & & \leftarrow & a, b. \\ r_6 = \mathbf{0.6}: & c & \leftarrow & not\ e. \\ r_7 = \mathbf{0.4}: & d & \leftarrow & not\ e. \\ r_8 = \mathbf{0.8}: & q & \leftarrow & r. \\ r_9 = \mathbf{0.8}: & r & \leftarrow & q. \end{array} \right\}$$

Definition 13 (Possibilistic Elimination of Contradictions) Given two LPPODs P and P', P' results from P by possibilistic elimination of contradictions ($P \rightarrow_{PeC} P'$) 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 $P' = P \backslash \{r\}$.

By applying this first transformation to the program in Example 8, we get rid of rule r_2 and we obtain P_1:

Example 9 ($P_0 \rightarrow_{PeC} P_1$)

$$P_1 = \left\{ \begin{array}{lrcl} r_1 = \mathbf{1}: & a \times b & \leftarrow & not\ c, not\ d. \\ r_3 = \mathbf{1}: & b \times a & \leftarrow & c. \\ r_4 = \mathbf{1}: & a \times b & \leftarrow & d. \\ r_5 = \mathbf{1}: & & \leftarrow & a, b. \\ r_6 = \mathbf{0.6}: & c & \leftarrow & not\ e. \\ r_7 = \mathbf{0.4}: & d & \leftarrow & not\ e. \\ r_8 = \mathbf{0.8}: & q & \leftarrow & r. \\ r_9 = \mathbf{0.8}: & r & \leftarrow & q. \end{array} \right\}$$

We would also like to delete *not* e from all rule bodies whenever e does not appear in the head of an LPPOD. This can be guaranteed by the *Possibilistic Positive Reduction.* On the contrary, if an LPPOD contains $\alpha : a \leftarrow \top$, then the atoms in the head must be true, so rules contained in their bodies *not* a are surely false and should be deleted. This can be guaranteed by the *Possibilistic Negative Reduction.*

Definition 14 (Possibilistic Positive Reduction) Given two LPPODs P and P', 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 $P' = (P \backslash \{r\}) \cup \{\alpha : \mathcal{C}^{\times} \leftarrow \mathcal{B}^{+},\ not\ \mathcal{B}^{-}\}$.

Definition 15 (Possibilistic Negative Reduction) Given two LPPODs P and P', 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 $P' = (P \backslash \{r'\})$.

An application of these reductions reduces the size of an LPPOD. In our example, we can apply $\rightarrow_{PRED^{+}}$ (two times) to obtain P_2 and then $\rightarrow_{PRED^{-}}$ to obtain P_3.

Example 10 $(P_1 \rightarrow_{PRED^{+}} P_2 \rightarrow_{PRED^{+}} P_3,\ P_3 \rightarrow_{PRED^{-}} P_4)$

$$P_3 = \left\{ \begin{array}{lcccc} r_1 = & \mathbf{1}: & a \times b & \leftarrow & not\ c, not\ d. \\ r_3 = & \mathbf{1}: & b \times a & \leftarrow & c. \\ r_4 = & \mathbf{1}: & a \times b & \leftarrow & d. \\ r_5 = & \mathbf{1}: & & \leftarrow & a, b. \\ r_6 = & \mathbf{0.6}: & c & \leftarrow & \top. \\ r_7 = & \mathbf{0.4}: & d & \leftarrow & \top. \\ r_8 = & \mathbf{0.8}: & q & \leftarrow & r. \\ r_9 = & \mathbf{0.8}: & r & \leftarrow & q. \end{array} \right\} \qquad P_4 = \left\{ \begin{array}{lcccc} r_3 = & \mathbf{1}: & b \times a & \leftarrow & c. \\ r_4 = & \mathbf{1}: & a \times b & \leftarrow & d. \\ r_5 = & \mathbf{1}: & & \leftarrow & a, b. \\ r_6 = & \mathbf{0.6}: & c & \leftarrow & \top. \\ r_7 = & \mathbf{0.4}: & d & \leftarrow & \top. \\ r_8 = & \mathbf{0.8}: & q & \leftarrow & r. \\ r_9 = & \mathbf{0.8}: & r & \leftarrow & q. \end{array} \right\}$$

The following two transformations are usually used to replace the Generalize Principle of Partial evaluation (*GPPE*) [10, 11].

Definition 16 (Possibilistic Success) Given two LPPODs P and P', 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 $P' = (P \backslash \{r\}) \cup \{min\{\alpha, \beta\} : \mathcal{C}^{\times} \leftarrow (\mathcal{B}^{+} \backslash \{a\}),\ not\ \mathcal{B}^{-}\}$.

Definition 17 (Possibilistic Failure) Given two LPPODs P and P', 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 $P' = (P \backslash \{r\})$.

The *Possibilistic Success* is particularly important for LPPODs because it allows for the propagation of necessity values between LPPOD rules. Please observe how such transformation reflects the possibilistic modus ponens [32]. By applying $\rightarrow_{PF}$ to P_4 (two times), we obtain:

Example 11 $(P_4 \rightarrow_{PS} P_5 \rightarrow_{PS} P_6)$

$$P_6 = \left\{ \begin{array}{lcccc} r_3 = & \mathbf{0.6}: & b \times a & \leftarrow & \top. \\ r_4 = & \mathbf{0.4}: & a \times b & \leftarrow & \top. \\ r_5 = & \mathbf{1}: & & \leftarrow & a, b. \\ r_6 = & \mathbf{0.6}: & c & \leftarrow & \top. \\ r_7 = & \mathbf{0.4}: & d & \leftarrow & \top. \\ r_8 = & \mathbf{0.8}: & q & \leftarrow & r. \\ r_9 = & \mathbf{0.8}: & r & \leftarrow & q. \end{array} \right\}$$

Sometimes, in a logic program, one can identify a positive dependency between atoms which form a cycle. For instance, by considering r_8 and r_9 from P_6, one can see that there is a positive dependency between the atoms q and r. Observe that these atoms can be considered as false without affecting the semantics of P_6; this means that one can remove r_8 and r_9 from P_6. In order to identify these kinds of positive dependencies which form a cycle, the possibilistic loop transformation is defined.

For its definition it is useful to map an LPPOD to a possibilistic normal program. Given a possibilistic ordered disjunction rule $r = \alpha : \mathcal{C}^\times \leftarrow \mathcal{B}^+,\ not\ \mathcal{B}^-$, we write $ordis - nor(r)$ to denote the set of possibilistic normal rules $\{\alpha : c \leftarrow \mathcal{B}^+,\ not\ (\mathcal{B}^- \cup (\mathcal{C}^\times \backslash \{c\})) | c \in \mathcal{C}^\times\}$ and we extend this definition to LPPODs as follows. Let P be an LPPOD, then $ordis - nor(P)$ denotes the possibilistic normal program $\bigcup_{r \in P} ordis - nor(r)$. Given a possibilistic normal program $ordis - nor(P)$, we write $def(ordis - nor(P))$ to denote the possibilistic definite logic program that is obtained from $ordis - nor(P)$ by removing every negated-by-failure atom in $ordis - nor(P)$. Given a definite logic program $def(ordis - nor(P))^*$, $Cn(def(ordis - nor(P))^*)$ denotes the unique minimal model of $def(ordis - nor(P))^*$ (that always exists for definite logic program, see [38]).

Definition 18 (Possibilistic Loop) Let $P_1 = P_1' \cup C$ and $P_2 = P_2' \cup C$ be two LPPODs in which P_1' and P_2' do not contain constraints and C is a set of possibilistic constraints. P_2 results from P_1 by possibilistic loop ($P_1 \rightarrow_{PLoop} P_2$), if $P_2' = \{\alpha : \mathcal{C}^\times \leftarrow \mathcal{B}^+,\ not\ \mathcal{B}^- | \alpha : \mathcal{C}^\times \leftarrow \mathcal{B}^+,\ not\ \mathcal{B}^- \in P_1'$ and $\mathcal{B}^+ \cap unf(P_1') = \emptyset\}$, where $unf(P_1') = \mathcal{L}_{P^*} \backslash Cn(def(ordis - nor(P_1'))^*)$ and $P_1 \neq P_2$.

Example 12 *($P_6 \rightarrow_{PLoop} P_7$)*

$$P_7 = \left\{ \begin{array}{l} r_3 = \mathbf{0.6} : b \times a \leftarrow \top. \\ r_4 = \mathbf{0.4} : a \times b \leftarrow \top. \\ r_5 = \ \ \mathbf{1} : \qquad \leftarrow a, b. \\ r_6 = \mathbf{0.6} : \ \ c \ \ \leftarrow \top. \\ r_7 = \mathbf{0.4} : \ \ d \ \ \leftarrow \top. \end{array} \right\}$$

Based on these transformations, we define an abstract rewriting system that contains the possibilistic transformation rules introduced in the previous definitions.

Definition 19 (Rewriting System for LPPODs) Let P be an LPPOD and $\mathcal{CS}_{LPPOD}$ be the rewriting system based on the possibilistic transformation rules $\{\rightarrow_{PeC}, \rightarrow_{PRED^+}, \rightarrow_{PRED^-}, \rightarrow_{PS}, \rightarrow_{PF}, \rightarrow_{PLoop}\}$. We denote the normal form of P *w.r.t.* $\mathcal{CS}_{LPPOD}$ by $norm_{\mathcal{CS}_{LPPOD}}(P)$.[9]

As stated before, an essential requirement of program transformations is that they preserve the semantics of the programs to which they are applied. The following lemma is an important result, since it allows for the reduction of an LPPOD without affecting its semantics.

Lemma 1 ($\mathcal{CS}_{LPPOD}$ preserves SEM_{LPPOD}) *Let P and P' be two LPPODs related by any transformation in $\mathcal{CS}_{LPPOD}$. Then $SEM_{LPPOD}(P) \equiv_p SEM_{LPPOD}(P')$.*

[9] Soon, we will show that this normal form is unique.

Returning to the description of the running example, it can be observed that the program P_7 has the property that it cannot be further reduced under the CS_{LPPOD} system (none of our transformations are applicable). Therefore, $P_7 = norm_{\mathcal{CS}_{LPPOD}}(P_0)$. However, the normal form of P could have been obtained by applying a different set of transformations. The following theorem is a strong result as it shows that a different application of our transformations (in a different ordering) leads to the same reduced program.

Theorem 1 (confluence and termination) *Let P be an LPPOD. Then $\mathcal{CS}_{LPPOD}$ is confluent, noetherian and $norm_{\mathcal{CS}_{LPPOD}}(P)$ is unique.*

These results have important implications. In the case of the LPPODs semantics implementation, the confluence guarantees that the order in which the transformations are applied does not matter and that an LPPOD can always be reduced to a unique normal form. Consequently, the semantics of LPPODs can be computed on the normal form of an LPPOD. Moreover, the *Possibilistic Success* is particularly important for LPPODs because it also allows necessity values between LPPODs rules to be propagated. These results are of interest in the next section.

5 Possibilistic Answer Sets Selection

In LPODs, the $\times$ connective is used to express a preference order among atoms. Such preference order induces an order among the answer sets of a logic program, since each answer set is associated with a rule satisfaction degree. Concerning LPPODs, we can consider the projection $*$ to use the rule satisfaction degree and the preference relation defined for LPODs (see Section 2.2) in order to rank the possibilistic answer sets of an LPPOD.

Example 13 Let us return to the LPPOD in Example 3.

$$P = \left\{\begin{array}{rcrcl rcrcl}
r_1 = & \mathbf{1}: & b & \leftarrow & ant. & r_6 = & \mathbf{1}: & & \leftarrow & ab_1, ab_2.\\
r_2 = & \mathbf{0.9}: & f & \leftarrow & b, not\ ab_1. & r_7 = & \mathbf{0.6}: & p & \leftarrow & ant.\\
r_3 = & \mathbf{0.6}: & \neg f & \leftarrow & ant, not\ ab_2. & r_8 = & \mathbf{0.4}: & sp & \leftarrow & ant.\\
r_4 = & \mathbf{1}: & ab_1 \times ab_2 & \leftarrow & p. & r_9 = & \mathbf{1}: & ant & \leftarrow & \top.\\
r_5 = & \mathbf{1}: & ab_2 \times ab_1 & \leftarrow & sp. & & & & &
\end{array}\right\}$$

As seen before (Examples 6-7), P has two possibilistic answer sets: $M_1 = \{(ant, 1), (p, 0.6), (sp, 0.4), (b, 1), (\neg f, 0.9), (ab_1, 0.6)\}$ and $M_2 = \{(ant, 1), (p, 0.6), (sp, 0.4), (b, 1), (f, 0.6), (ab_2, 0.6)\}$. Let us consider the rule satisfaction degrees of M_1 and M_2:

Table 1 Example of Rule Satisfaction Degrees

Rule	$deg_{M_1^*}$	$deg_{M_2^*}$
$r_4^* = ab_1 \times ab_2 \leftarrow p$	1	2
$r_5^* = ab_2 \times ab_1 \leftarrow sp$	2	1

From Definition 4, it is easy to see that it is not possible to compare the possibilistic answer sets.

Therefore, rule satisfaction degrees are not enough to compare the possibilistic answer sets of a program such as P. However, the rules in our LPPODs are associated with certainty values, *i.e.* one ordered disjunction rule can be more certain than another one. This observation is the motivation for the need to explore a more precise comparison criterion able to take necessity values into account when needed.

By exploiting the set of transformation rules we have defined in the previous section, we can define a preference relation that takes the necessity values of LPPOD rules into account in order to compare possibilistic answer sets. The following *possibilistic preference relation* is able to consider rule satisfaction degrees and rule necessity values to specify an order between the possibilistic answer sets of an LPPOD.

Definition 20 (Possibilistic Preferred Relation) 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 ($M_1 \succ_{pp} M_2$) 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 $deg_{M_2^*}(r'^*) < deg_{M_1^*}(r'^*)$ and $Nec(r) < Nec(r')$.[10]

This definition tells us that, once we have rewritten an LPPOD P by applying the transformations seen before, we are able to compare the possibilistic answer sets directly using the normal form of the program To illustrate the above definition, let us consider the following example.

Example 14 Let P be the LPPOD in the example above (Example 13), and $\mathcal{CS}_{LPPOD}$ be the abstract rewriting system we defined for LPPODs. By applying *Possibilistic Success* (three times), we can first rewrite r_1, r_7, and r_8 in P to obtain P_3:

$$P_3 = \left\{ \begin{array}{lrcl lrcl} r_1 = \mathbf{1}: & b & \leftarrow & \top. & r_6 = \mathbf{1}: & & \leftarrow & ab_1, ab_2. \\ r_2 = \mathbf{0.9}: & f & \leftarrow & b, not\, ab_1. & r_7 = \mathbf{0.6}: & p & \leftarrow & \top. \\ r_3 = \mathbf{0.6}: & \neg f & \leftarrow & ant, not\ ab_2. & r_8 = \mathbf{0.4}: & sp & \leftarrow & \top. \\ r_4 = \mathbf{1}: & ab_1 \times ab_2 & \leftarrow & p. & r_9 = \mathbf{1}: & ant & \leftarrow & \top. \\ r_5 = \mathbf{1}: & ab_2 \times ab_1 & \leftarrow & sp. & & & & \end{array} \right\}$$

By applying the same transformation another two times, we can rewrite rule r_4 and r_5 and obtain P_5 which represents the normal form of P:

$$P_5 = \left\{ \begin{array}{lrcl lrcl} r_1 = \mathbf{1}: & b & \leftarrow & \top. & r_6 = \mathbf{1}: & & \leftarrow & ab_1, ab_2. \\ r_2 = \mathbf{0.9}: & f & \leftarrow & b, not\, ab_1. & r_7 = \mathbf{0.6}: & p & \leftarrow & \top. \\ r_3 = \mathbf{0.6}: & \neg f & \leftarrow & ant, not\ ab_2. & r_8 = \mathbf{0.4}: & sp & \leftarrow & \top. \\ r_4 = \mathbf{0.6}: & ab_1 \times ab_2 & \leftarrow & \top. & r_9 = \mathbf{1}: & ant & \leftarrow & \top. \\ r_5 = \mathbf{0.4}: & ab_2 \times ab_1 & \leftarrow & \top. & & & & \end{array} \right\}$$

Indeed, it can be proven that $P_5 = norm_{\mathcal{CS}_{LPPOD}}(P)$ and that it has the same possibilistic answer sets of P, as expected. Once we have obtained the normal form of P, we can apply the possibilistic preference relation to compare M_1 and M_2. By considering rule satisfaction degrees of M_1 and M_2 (Table 1) and the necessity values of rules r_4 and r_5, *i.e.* $Nec(r_4) = 0.6$ and $Nec(r_5) = 0.4$, it is not difficult to

[10] $Nec(r)$ denotes the necessity value of a rule r as explained in Section 3.1.

see that $M_1 \succ_{pp} M_2$, since $Nec(r_4) > Nec(r_5)$ ($M_2 \not\succ_{pp} M_1$ follows by Definition 20 as well).

From a knowledge representation point of view, we can interpret this result in the following way. In the original LPPOD P, we were assuming that rule r_4, *i.e.* being a penguin it is more reasonable to assume that birds cannot fly rather than that Antarctic birds can fly, and rule r_5, *i.e.* being a super-penguin it is more reasonable to assume that Antarctic birds can fly rather than birds cannot fly, were *absolutely certain*. However, these rules are supported by two pieces of knowledge, rules r_7 and r_8, which are *almost certain* and *little certain* respectively. As expected, these certainty values must be considered at the moment of deciding which one of the exceptions ab_1 and ab_2 *w.r.t.* the default rules r_2 and r_3 is the most plausible. This has been achieved by means of a mechanism able to propagate necessity values between the rules of LPPOD P.

The reader may observe that, when necessity values are equal, it is not possible to achieve a total order between possibilistic answer sets of an LPPOD. However, this is in someway what can be expected when both scales prevent such a decision.

Nevertheless, we can observe that there is an important property *w.r.t.* the possibilistic preference relation for possibilistic answer sets. A possibilistic answer set M is comparable if M^* is comparable in the LPOD P^*. In fact, our extension generalizes the preference relation between answer sets of LPODs *w.r.t.* the preference relation for LPODs.

Proposition 5 *Let M_1 and M_2 be possibilistic answer sets of an LPPOD P, $\succ_p$ be the preference relation based on rules satisfaction degrees only, and $\succ_{pp}$ be the possibilistic preference relation. If $M_1^* \succ_p M_2^*$ then $M_1 \succ_{pp} M_2$.*

It is worthy adding that the approach we have defined in this section for comparing possibilistic answer sets based on the rewriting system CS_{LPPOD} can be replaced by other strategies. An alternative strategy can be obtained by associating a unique value, which merges necessity and satisfaction degree together, with each possibilistic answer set and to use such value to rank-order the possibilistic answer sets.[11] Other possible strategies can be proposed by defining an alternative program reduction which focuses on the propagation of necessity values only; or by transforming an LPPOD to an equivalent possibilistic normal logic program where special atoms are introduced for denoting the satisfaction degree of possibilistic ordered disjunction rules and by defining a comparison criterion based on the necessity values associated with these atoms.

6 Computing the LPPODs Semantics

In Section 3, we have seen how the LPPODs semantics can be characterized in two different ways: syntactically, in terms of a syntactic reduction and a fix-point operator, and semantically, in terms of the least specific possibility distribution compatible with an LPPOD.

[11] A possible way to associate a unique value to each possibilistic answer set is by means of the following function: $\sum_{1 \leq i \leq n} \frac{Nec(r_i)}{deg_{M^*}(r_i) n_r}$, in which n_r is the number of rules in an LPPOD P.

Algorithm 1 SEM_{LPPOD}

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

For the sake of computation, the former characterization is of special interest as it defines a straightforward methodology for the LPPODs semantics implementation. According to Definition 10, the LPPODs semantics is defined in terms of the LPODs semantics [15] and the possibilistic answer set semantics [41]. Since both semantics are computable, the decision problem of the existence of a possibilistic answer set of an LPPOD is computable, as stated in the following theorem.

Theorem 2 *Let P be an LPPOD, such that $P \subseteq Prog_{\langle \mathcal{A},\mathcal{S}\rangle}$, where $Prog_{\langle \mathcal{A},\mathcal{S}\rangle}$ is the set of all LPPODs which can be generated by a finite set of atoms $\mathcal{A}$ and a finite set of necessity values $\mathcal{S}$. Then, the LPPODs semantics is computed by the function $SEM_{LPPOD} : P \rightarrow \mathcal{PS}$.*

The algorithm that computes the LPPODs semantics is described as follows. Algorithm 1 accepts an LPPOD P and it returns an ordered set of possibilistic answer sets of P. To be able to infer the possibilistic answer set of P, the algorithm follows the methodology provided by the LPPODs semantics syntactical characterization in a straightforward way. As the first step, the projection $*$ is applied to P to obtain its classical part (P_{LPOD}). Since P_{LPOD} is an LPOD, the LPOD semantics (SEM_{LPOD}) can be used to infer its answer sets ($\mathcal{M}$). If no answer set of P_{LPOD} can be found, then the algorithm terminates. Otherwise, each answer set of P_{LPOD} is used to reduce P ($P_{\times}^{M}$) to a possibilistic definite logic program to which the fix-point operator can be applied in order to infer the necessity values of the atoms corresponding to the possibilistic answer sets of P. Since the consequence operator is monotonic, $\mathcal{PS}$ is finite, and each $ps \subset \mathcal{PS}$ is also finite, the operator always reaches a fix-point in a finite number of steps. Therefore, the algorithm always terminates. In this way, the algorithm provides an effective way to compute the LPPODs semantics.

Furthermore, the relation of the LPPODs semantics with the LPODs and possibilistic answer set semantics suggests other important results.

First, it can be observed the way in which the problem of deciding whether an LPPOD has a possibilistic answer set stays in the same complexity of finding an answer set of an LPOD, *i.e.* Σ_2^P. This is given by the fact that finding a possibilistic answer set of an LPPOD can be done in polynomial time from the moment an answer set of its associated LPOD is provided. This comes in a straightforward way from the fact that (i) the $\times$-possibilistic reduction converts an LPPOD to a possibilistic definite logic program and, (ii) the computation of the possibilistic answer set of a possibilistic definite logic program P can be done in polynomial time *w.r.t.* $k \times |P|$ in which k is the number of different levels of certainty occurring

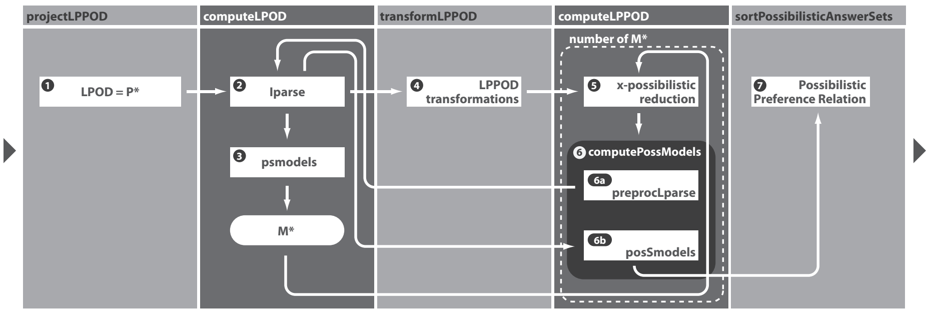

Fig. 1 Overview of the *posPsmodels* system

in P and $|P|$ is the number of rules in P. This is an important result, since it shows how LPPODs can yield a more expressive framework without increasing the complexity of its classical part.

Secondly, as the LPODs and possibilistic answer set semantics are already implemented, we have been able to implement a proof-of-concept prototype of the algorithm using as main components the LPOD solver *psmodels* [15] and the possibilistic normal logic programs solver *posSmodels* [41] as the main components.

We have implemented Algorithm 1 in a C++ program called *posPsmodels* (Figure 1).[12] The system accepts an LPPOD according to the syntax presented in Section 3.1 and returns an ordered set of possibilistic answer sets. In our system we have used *lparse* (2), *psmodels* (3) and *posSmodels* (6) in an interleaved fashion and we have implemented the modules for the LPPOD projection (1), the LPPODs transformations (4), the LPPODs possibilistic $\times$-reduction (5), and the possibilistic preference relation (7).

7 Related Work

Since the LPPODs framework combines preferences, nonmonotonicity and uncertainty in a logic programming framework together, there are several works which relate to ours from different perspectives. Most of the existing approaches either deal with uncertain nonmonotonic reasoning or deal with preference handling.

7.1 Uncertain Nonmonotonic Reasoning

Concerning research on logic programming with uncertainty, existing approaches in the literature interpret the uncertainty in a qualitative or in a quantitative way.

As far as qualitative treating of uncertainty is concerned, the most popular approaches are based on possibility theory. The first approach which combines possibilistic logic and logic programming, is an idea presented in [31]. In this work, logic programming rules are associated with necessity measures and certainty about the atoms in valid interpretations of a logic program are inferred using possibilistic resolution [32]. This approach is able to deal with inconsistent programs by means

[12] A beta version of the *posPsmodels* system can be downloaded from `http://github.com/rconfalonieri/posPsmodels/tarball/master`.

of the α-cut [31, 32], but it is not able to capture incomplete knowledge as it does not consider negation as failure, as is usually done in ASP.

In the context of ASP, one of the first works about qualitative uncertainty handling is the work of PASP [41] which combines possibility theory with normal logic programs. Our approach clearly relates to this work as the LPPODs semantics is based on their possibilistic semantics. Moreover, when all possibilistic rules in an LPPOD are $\times$ free, the LPPODs semantics coincides with PASP (indeed it generalizes PASP).

PASP was later extended to disjunctive programs [42]. The main motivation of this approach was to define a description language and a reasoning process where the user could consider relative likelihoods for modeling different levels of uncertainty. Our approach differs in these aspects as we are more focused on the selection of uncertain default rules. Moreover, uncertainty labels in [42] belong to a partially ordered set (*i.e.* lattice), while in our approach necessity values belong to a totally ordered scale. Despite these slight differences, it is not difficult to see how the two semantics relate to each other. As the $\times$ connector in LPPODs is a special disjunction and as the possibilistic disjunctive semantics [42] is based on minimal models[13], it is possible to represent an LPPOD by means of a possibilistic disjunctive program. The main difference is in the selection of the possibilistic answer set of the program. Nevertheless, it seems also possible to recover the satisfaction degree of LPPODs in a possibilistic disjunctive program as well. Intuitively, as in LPPODs a satisfaction degree is associated with a possibilistic answer set *w.r.t.* a possibilistic ordered disjunction rule. This degree can be recovered in a possibilistic disjunctive rule by defining a satisfaction relation for possibilistic disjunction rules that takes the position of the satisfied atoms into account.

Another possibilistic extension of the semantics of LPODs, based on possibilistic pstable semantics [43], is the approach proposed in [20]. The main motivation behind this work is the handling of possibilistic rules of the form $\alpha : a \leftarrow \mathit{not}\ a$, which under possibilistic answer set semantics are considered inconsistent. The core idea of the approach is first to characterize the LPODs semantics in terms of the pstable semantics and then to generalize such result to possibilistic LPODs [20]. Although the research line of such a framework is in paraconsistent logic, possibilistic pstable semantics for LPODs relates to LPPODs semantics, since each possibilistic answer set is a possibilistic pstable model in which the main difference consists in the necessity values of the inferred atoms (see Proposition 2 in [20]).

A noticeable approach to uncertainty handling in ASP based on possibility theory is reported in [4]. The authors propose a revisited semantics for PASP. The core idea of this approach is to translate a normal program to a set of constraints on possibility distributions and, then, to extend this idea to cover possibilistic ASP. Our approach differs from [4] in the way in which the possibilistic answer sets are computed. Indeed, we first compute the answer sets of the no possibilistic program and, then, we use the possibilistic semantics for inferring the necessity values (according to PASP). This revisited possibilistic semantics has recently also been extended to possibilistic disjunctive logic programs [5].

In the case the uncertainty is treated quantitatively the most used formalism is probability theory as in [3, 45]. Basically these approaches differ from ours in

[13] Given a possibilistic disjunctive rule $\alpha : a \vee b$, $\{(a, \alpha)\}$ and $\{(b, \alpha)\}$ are valid possibilistic answer sets, whereas $\{(a, \alpha), (b, \alpha)\}$ is not.

the underlying notion of uncertainty and how uncertainty values associated to clauses and facts are managed. The work in [3] presents a nonmonotonic probabilistic logic programming language (P-log) which is based on, and generalizes, logic programming under answer set semantics and Causal Bayesian networks. The P-log semantics is defined in terms of probabilistic models representing possible worlds of the domain being represented. Nonmonotonicity is achieved by means of an updating mechanism which changes the collection of possible worlds when new probabilistic information is added to the program. Another example of probabilistic nonmonotonic formalism can be found in [45]. In this work, it is possible to represent rules in which every atom is annotated with a probability interval. Then, the associated semantics is defined in terms of a probabilistic well-founded semantics and a probabilistic stable model semantics.

Finally, among logic-based approaches, the most noticeable works are presented in [39, 46] in the probabilistic and possibilistic logic settings. In [39], a weak nonmonotonic probabilistic logic is proposed to handle statements such as *normally p's are q's with probability of at least* α in a uniform framework. Such framework can handle strict logical knowledge and purely probabilistic knowledge from probabilistic logic, as well as default logical knowledge from conditional knowledge bases. In the possibilistic setting [46], uncertain default rules such as *normally p's are q's with certainty at least* α are handled by a two step process. First, default rules are rewritten into possibilistic logic formulas and preferential entailment is used to draw plausible conclusions. Then, possibility logic resolution [32] is used to handle the certainty associated with the information described in the possibilistic knowledge base.

7.2 Preference Handling

Preference handling has been a hot investigation topic in the last two decades. In logic programming, besides providing an effective way to express an order among rules or among beliefs to be used in the reasoning process [16, 17, 26, 47], qualitative preferences are also very important when user preferences have to be encoded in a compact way [14, 17, 18, 36]. Among quantitative approaches to preference specification, the work of Costantini *et al.* [23], in which an extension of ASP is designed to model quantitative resources and to enable the specification of nonlinear preferences (both in the heads and in the bodies), is worth mentioning. This approach has also been applied to pure ASP in order to capture weight constraints with preferences [24].

Other noticeable approaches to the treatment of the problem of conditional (qualitative) preference specification, are the works in the possibility theory [6, 8] and in the CP-nets [9] settings. For a more general overview about preferences in AI, we refer to [29, 35].

Concerning possibility theory, a closer relationship between possibilistic logic and LPPODs can be drawn when considering the recent work of Bosc *et al.* [8]. In [8], the authors discuss a different perspective on possibilistic logic able to represent preferences and uncertainty at the same time as it happens in LPPODs. This is the case when the possibilistic logic setting is used to encode preference queries to possibilistic databases where the data is pervaded with uncertainty [7, 8]. Since this approach is close to ours, a more detailed discussion is presented in the next section.

7.3 Preference Queries to a Possibilistic Database

In the context of possibilistic databases, the use of possibilistic logic for representing uncertain data can provide a strong representation system for the whole relational algebra [7]. In particular, a possibilistic database can be directly encoded in possibilistic logic by mapping keys into variables, attributes into predicates, and database tuples into instantiated formulas [7]. Then, answering a query such as $\exists x\ C_1(x) \wedge \ldots \wedge C_n(x)$ (resp. $\exists x\ C_1(x) \vee \ldots \vee C_n(x)$) amounts to add the possibilistic logic formula $\{(\neg C_1(x) \vee \ldots \neg C_n(x) \vee answer(x), 1)\}$ (resp. $\{(\neg C_1(x) \vee answer(x), 1),\ldots,(\neg C_n(x) \vee answer(x), 1)\}$) to the possibilistic base and to evaluate the query by the repeated application of possibilistic resolution $(\neg P(x) \vee Q(x, y), \alpha), (P(a) \vee R(z), \beta) \vdash_{PL} (Q(a, y) \vee R(z), min(\alpha, \beta))$ [32].

However, it is possible for these queries to either return an empty set of answers or to return too many results. Such cases have motivated the study of a way to incorporate qualitative preferences inside queries in the context of possibilistic logic. When modeling preferences in possibilistic logic, the necessity measure α associated with a classical formula p in (p, α) is understood as the priority of p rather than its certainty level [6]. The incorporation of preferences inside a query can be achieved by assigning a (total) order over the necessity values associated with the query. In [8], the authors show how it is possible to represent *conjunctive* and *disjunctive* preference queries asking for items satisfying conditions $C_1, C_2, \ldots, C_{n-1}, C_n$ with the information that C_1 *is required and if possible* C_2 *also,. . . , and if possible* C_n *too* and C_n *is required, or better* C_{n-1}*,. . .,or still better* C_1 respectively.

For instance, given three conditions $C_1(x)$, $C_2(x)$, $C_3(x)$ and the necessity measures $1, \alpha, \beta$ (seen as priority levels associated with the possibilistic formula representing the queries) with $1 > \alpha > \beta$, the possibilistic logic encoding of a conjunctive (resp. disjunctive) query is $\{(\neg C_1(x) \vee \neg C_2(x) \vee \neg C_3(x) \vee answer(x),1)$, $(\neg C_1(x) \vee \neg C_2(x) \vee answer(x),\alpha)$, $(\neg C_1(x) \vee answer(x),\beta)\}$ (resp. $\{(\neg C_1(x) \vee answer(x),1)$, $(\neg C_2(x) \vee answer(x),\alpha)$, $(\neg C_3(x) \vee answer(x),\beta)\}$).

Once a preference query is evaluated, the retrieved items are associated with two levels: to *what extent the answer satisfies the query* (*i.e.* $1, \alpha, \beta$), and to *what extent the data used in the query evaluation is certain* (*i.e.* the necessity measures in the database). These scales are used to achieve a totally ordered set of query results [8].

In the following, we show how it is possible to encode uncertain data and preference queries in the LPPODs framework. To this end, we first show how a preference query is represented and processed in possibilistic logic and, then, how it can be represented and processed in the LPPODs framework. We only show the case of a conjunctive query, since an equivalence between conjunctive and disjunctive queries exists [8]. For the comparison, we will borrow an example presented in [8].

Let us consider a database example DB with three relations $Cities$, $Markets$, and $Museums$ which contain uncertain pieces of data (Table 2). We adopt the model proposed in [7]. According to it, each uncertain attribute value is associated

Table 2 Table *Cities* (top-left), *Markets* (top-right), and *Museums* (bottom) (example taken from [8])

id	Name	City
1	John	(Brest,**a**)
2	Mary	(Lannion,**b**)
3	Peter	(Quimper,**c**)

City	Flea Market
Brest	(yes,**d**)
Lannion	(no,**e**)
Quimper	(no,**f**)
Rennes	(yes,**g**)

City	Museum
Rennes	(modern,**h**)
Quimper	(contemporary,**i**)
Brest	(modern,**k**)

with a certainty degree which is modeled as a lower bound of a necessity measure. For instance, the intuitive reading behind a tuple such as $\langle 1, John, (Brest, \mathbf{a})\rangle$ is that, a person named *John* exists for sure, who lives in *Brest* with certainty $\boldsymbol{a}$. The remaining tuples can be understood in a similar way.

Given the possibilistic database DB, let us consider the following preference query asking:

Example 15 [8] *Find people living in a city with a flea market ($fleaMarket(x)$) and preferably a museum of modern art ($modern(x)$) and, if possible, a museum of contemporary art ($contemp(x)$).*
The query is clearly conjunctive and its possibilistic logic representation is:

$$Q_\wedge = \left\{ \begin{array}{c} (\neg fleaMarket(x) \vee \neg modern(x) \vee \neg contemp(x) \vee answer(x), 1) \\ (\neg fleaMarket(x) \vee \neg modern(x) \vee answer(x), \alpha) \\ (\neg fleaMarket(x) \vee answer(x), \beta) \end{array} \right\}$$

where $1 > \alpha > \beta$.
By applying possibilistic logic resolution, it can be proven that valid answers to the query are: $\{answer(''John''),\ \alpha, min(a, d, k)\}$ and $\{answer(''John''),\ \beta, min(a, d)\}$.

In the encoding of the database DB in LPPODs, we adopt the same convention used in [7]. According to it, keys become variables, attributes become predicates, and database tuples are encoded by instantiated predicates. As attribute certainty is modeled as a necessity measure, we can directly map it to the necessity values of program rules in our LPPODs framework. Therefore, the LPPOD representing the DB in Table 2, denoted by P_{DB}, is:

$$P_{DB} = \left\{ \begin{array}{cc} r_1 = \mathbf{a} : city(''John'', ''Brest''). & r_6 = \mathbf{f} : \neg fleaMarket(''Quimper''). \\ r_2 = \mathbf{b} : city(''Mary'', ''Lannion''). & r_7 = \mathbf{g} : fleaMarket(''Rennes''). \\ r_3 = \mathbf{c} : city(''Peter'', ''Quimper''). & r_8 = \mathbf{h} : modern(''Rennes''). \\ r_4 = \mathbf{d} : fleaMarket(''Brest''). & r_9 = \mathbf{i} : modern(''Brest''). \\ r_5 = \mathbf{e} : \neg fleaMarket(''Lannion''). & r_{10} = \mathbf{k} : contemp(''Quimper''). \end{array} \right\}$$

Let us consider now the query in Example 15. Its encoding in the LPPODs framework, denoted by $P_{Q_\wedge}$, is:

$r_{11} = 1 : q_1(X) \leftarrow city(X,Y), fleaMarket(Y), modern(Y), contemp(Y),$
$\qquad\qquad\qquad\qquad\qquad\qquad not\ q_2(X), not\ q_3(X).$
$r_{12} = 1 : q_2(X) \leftarrow city(X,Y), fleaMarket(Y), modern(Y), not\ q_1(X), not\ q_3(X).$
$r_{13} = 1 : q_3(X) \leftarrow city(X,Y), fleaMarket(Y), not\ q_1(X), not\ q_2(X).$
$r_{14} = 1 : q_1(X) \times q_2(X) \times q_3(X).$

The main difference *w.r.t.* the possibilistic logic encoding is in the way in which we represent the query priority. In fact, we encode priority about a query by a possibilistic ordered disjunction rule (r_{14}) rather than by means of necessity values as done in possibilistic logic. Then, we associate queries and preferences about the queries with a necessity of **1** in order to keep the necessity values of the queries results as the certainty values computed by the fix-point operator (which reflects the possibilistic modus ponens).

Concerning the query execution, it can be confirmed that the program $P_{DB} \cup P_{Q_\wedge}$ does not have any result matching q_1 (as in Example 15), while we obtain $M_1 = \{q2(''John''), min(a,\ d, k)\}$ with $deg_{M_1}(r_{14}) = 2$ matching q_2 and $M_2 = \{q3(''John''), min(a, d)\}$ with $deg_{M_2}(r_{14}) = 3$ matching q_3. Please observe that the retrieved items are associated with two levels: (i) the satisfaction degree of the possibilistic answer set *w.r.t.* the ordered disjunction rule, (ii) and the necessity values as certainty measures *w.r.t.* the data used in the query evaluation. This is clearly in accordance with the result obtained in the possibilistic logic setting. Moreover, the total order of the preference query in the possibilistic logic setting is reflected in LPPODs as well. In fact, according to the possibilistic preference relation, M_1 is preferred to M_2 since $deg_{M_1}(r_{14}) < deg_{M_2}(r_{14})$ (see Definition 20).

To conclude, the representation of preference queries in the LPPODs setting seems to be feasible and there is a close relation between the two approaches. However, on a representational level the possibilistic setting seems to be more expressive. In [8], how possibilistic logic can also accommodate some cases of disjunctive information (for instance the case in which the third tuple of relation R is $\langle Peter, (Quimper \vee Rennes, c)\rangle$) is discussed. Instead, the LPPODs syntax does not allow disjunction in the head of program rules. A possible way to overcome this limitation is by extending the LPPODs syntax and semantics with possibilistic disjunction [42]. On the other hand, LPPODs can provide a computational machinery for computing preference queries in a possibilistic database in a practical way, whereas possibility theory addresses the problem of flexible querying in a theoretical way only.

8 Concluding Remarks

In this paper, we have proposed a possibilistic logic programming framework which is able to capture explicit preferences about rules having exceptions and uncertainty values in terms of necessity measures according to possibilistic logic [32]. As reported in Section 7, the use of possibilistic logic in ASP is not new and several proposals under different specifications have been studied [4, 41, 42]. Nevertheless, LPPODs is the first logic programming specification able to consider explicit preferences and uncertainty together based on possibilistic answer set semantics.[14]

[14] In [20], we proposed a possibilistic extension of LPODs based on possibilistic pstable semantics [43].

In defining our framework, we have embedded several aspects of common-sense reasoning, nonmonotonocity, preferences, and uncertainty, where each part is underpinned by a well established formalism: LPODs [15] and PASP [41]. In joining these works together, we have obtained a framework which is able to consider two scales to specify an order between the solutions of an LPPOD: the *rule satisfaction degree* of a possibilistic answer set *w.r.t.* a possibilistic ordered disjunction rule, and the *necessity values* of the possibilistic rules themselves. In working in this direction, we have progressively built all the theoretical knowledge needed to define a possibilistic preference relation which can consider the satisfaction degree and necessity values of program rules for comparing LPPODs solutions.

First, we have extended the LPODs syntax with necessity values and we have defined the LPPODs semantics in terms of the LPODs and possibilistic answer set semantics (Definition 10). We have also shown how the LPPODs semantics can be defined in terms of the least possibility distribution of possibilistic definite logic programs (Proposition 4). Then, we have introduced a set of transformation rules (Definition 19) which represent an efficient way to reduce the size of an LPPOD and to propagate the necessity values between LPPOD rules (preserving the semantics, Lemma 1 and Theorem 1). In this way, necessity values can be taken into account in the possibilistic answer sets selection (Definition 20). Moreover, since the LPPODs semantics is a generalization of both the LPODs semantics and the possibilistic answer set semantics (Definition 10, Proposition 3), the rewriting system we have defined for LPPODs can be applied, in principle, both to LPODs and to possibilistic normal programs.

We have proven that the LPPODs semantics is computable (Theorem 2) and we have discussed 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 since it shows that LPPODs yield a more expressive framework without affecting the complexity. We have presented the LPPODs semantics implementation as an ASP-based solver called *posPsmodels*. The solver implements the transformation rules and the possibilistic preference relation, and thus generally speaking, the LPPODs framework can be tailored to real applications where a best option has to be chosen taking into account both preferences and uncertainty. Some applications of the LPPODs framework have been explored in [17, 19].

In [17], we described how the solver has been integrated in a context-aware system, where LPPODs is used to build user profiles and preferences are considered to enhance the recommendation results. In [19], we proposed an LPPODs-based methodology to model qualitative decision making under uncertainty problems. It is shown how, when knowledge and preferences are kept separate, the LPPODs framework can provide a convenient setting to compute an optimal decision according to optimistic and pessimistic decision criteria formulated according to possibility theory [30].

References

1. Baral C (2003) Knowledge Representation, Reasoning and Declarative Problem Solving. Cambridge University Press
2. Baral C, Gelfond M (1994) Logic Programming and Knowledge Representation. Journal of Logic Programming 19/20:73–148

3. Baral C, Gelfond M, Rushton N (2009) Probabilistic Reasoning with Answer Sets. Theory and Practice of Logic Programming 9(1):57–144
4. Bauters K, Schockaert S, Cock MD, Vermeir D (2010) Possibilistic Answer Set Programming Revisited. In: Proceedings of the 26th Conference on Uncertainty in Artificial Intelligence, (UAI'10), AUAI Press
5. Bauters K, Schockaert S, Cock MD, Vermeir D (2011) Weak and strong disjunction in possibilistic ASP. In: Benferhat S, Grant J (eds) Proceedings of the 11th International Conference on Scalable Uncertainty Management (SUM 2011), Lecture Notes in Computer Science, vol 6929, Springer, pp 475–488
6. Benferhat S, Dubois D, Prade H (2001) Towards a Possibilistic Logic Handling of Preferences. Applied Intelligence 14(3):303–317
7. Bosc P, Pivert O, Prade H (2009) A Model Based on Possibilistic Certainty Levels for Incomplete Databases. In: Godo L, Pugliese A (eds) Proceedings of the 3rd International Conference on Scalable Uncertainty Management (SUM'09), Lecture Notes in Computer Science, vol 5785, Springer Berlin - Heidelberg, pp 80–94
8. Bosc P, Pivert O, Prade H (2010) A Possibilistic Logic View of Preference Queries to an Uncertain Database. In: Proceedings of 19th IEEE International Conference on Fuzzy Systems (FUZZ-IEEE'10), pp 581–595
9. Boutilier C, Brafman RI, Domshlak C, Hoos HH, Poole D (2004) CP-nets: a tool for representing and reasoning with conditional ceteris paribus preference statements. Journal of Artificial Intelligence Research 21(1):135–191
10. Brass S, Dix J (1995) Characterizations of the Stable Semantics by Partial Evaluation. In: Marek V, Nerode A, Truszczyński M (eds) Proceedings of the Third International Conference on Logic Programming and Nonmonotonic Reasoning, (LPNMR'95), Lecture Notes in Computer Science, vol 928, Springer-Verlag, London, UK, pp 85–98
11. Brass S, Dix J (1999) Semantics of (disjunctive) Logic Programs Based on Partial Evaluation. Journal of Logic Programming 40(1):1–46
12. Brass S, Dix J, Freitag B, Zukowski U (2001) Transformation-based bottom-up computation of the well-founded model. Theory and Practice of Logic Programming 1(5):497–538
13. Brewka G, Dix J, Konolige K (1997) Nonmonotonic Reasoning: An Overview. CSLI Lecture Notes 73, CSLI Publications, Stanford, CA
14. Brewka G, Niemelä I, Truszczyński M (2003) Answer Set Optimization. In: Gottlob G, Walsh T (eds) Proceedings of 18th International Joint Conference on Artificial Intelligence, (IJCAI'03), Morgan Kaufmann, pp 867–872
15. Brewka G, Niemelä I, Syrjänen T (2004) Logic Programs with Ordered Disjunction. Computational Intelligence 20(2):333–357
16. Brewka G, Niemelä I, Truszczyński M (2008) Preferences and Nonmonotonic Reasoning. AI Magazine 29(4):69–78
17. Confalonieri R (2011) The Role of Preferences in Logic Programming: Nonmonotonic Reasoning, User Preferences, Decision under Uncertainty. PhD thesis, Universitat Politècnica de Catalunya, Barcelona, Spain
18. Confalonieri R, Nieves JC (2011) Nested Preferences in Answer Set Programming. Fundamenta Informaticae 113, DOI 10.3233
19. Confalonieri R, Prade H (2011) Answer Set Programming for Computing Decisions Under Uncertainty. In: Liu W (ed) Proceedings of the 11th European Conference on Symbolic and Quantitative Approaches to Reasoning with Un-

certainty (ECSQARU 2011), Lecture Notes in Artificial Intelligence, vol 6717, Springer-Verlag, Berlin, Heidelberg, pp 485–496
20. Confalonieri R, Nieves J, Vázquez-Salceda J (2009) Pstable Semantics for Logic Programs with Possibilistic Ordered Disjunction. In: Serra R, Cucchiara R (eds) Proceedings of the 11th International Conference of the Italian Association for Artificial Intelligence on Emergent Perspectives in Artificial Intelligence, (AI*IA '09), Lecture Notes in Artificial Intelligence, vol 5883, Springer Berlin, Berlin, Heidelberg, pp 52–61
21. Confalonieri R, Nieves JC, Osorio M, Vázquez-Salceda J (2010) Possibilistic Semantics for Logic Programs with Ordered Disjunction. In: Link S, Prade H (eds) Proceedings of 6th International Symposium on Foundations of Information and Knowledge Systems, (FoIKS 2010), Lecture Notes in Computer Science, vol 5956, Springer-Verlag, Berlin, Heidelberg, pp 133–152
22. Confalonieri R, Prade H, Nieves JC (2011) Handling Exceptions in Logic Programming without Negation as Failure. In: Liu W (ed) Proceedings of the 11th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU 2011), Lecture Notes in Artificial Intelligence, vol 6717, Springer-Verlag, Berlin, Heidelberg, pp 509–520
23. Costantini S, Formisano A (2009) Modeling preferences and conditional preferences on resource consumption and production in ASP. Journal of Algorithms 64(1):3–15
24. Costantini S, Formisano A (2011) Weight Constraints with Preferences in ASP. In: Delgrande J, Faber W (eds) Proceedings of the 11th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR11), Lecture Notes in Artificial Intelligence, vol 6645, Springer-Verlag, Berlin, Heidelberg, pp 229–235
25. van Dalen D (1994) Logic and Structure, 3rd edn. Springer-Verlag, Berlin
26. Delgrande J, Schaub T, Tompits H, Wang K (2004) A classification and Survey of Preference Handling Approaches in Nonmonotonic Reasoning. Computational Intelligence 20(2):308–334
27. Dershowitz N, Plaisted DA (2001) Rewriting. In: Robinson JA, Voronkov A (eds) Handbook of Automated Reasoning, Elsevier and MIT Press, pp 535–610
28. Dix J, Osorio M, Zepeda C (2001) A General Theory of Confluent Rewriting Systems for Logic Programming and its Applications. Annals of Pure and Applied Logic 108(1–3):153–188
29. Domshlak C, Hüllermeier E, Kaci S, Prade H (2011) Preferences in AI: An overview. Artifical Intelligence 175(7-8):1037–1052
30. Dubois D, Prade H (1995) Possibility theory as a basis for qualitative decision theory. In: Proceedings of the 14th International Joint Conference on Artificial Intelligence, (IJCAI'95), Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp 1924–1930
31. Dubois D, Lang J, Prade H (1991) Towards Possibilistic Logic Programming. In: Furukawa K (ed) Proceedings of International Conference on Logic Programming, (ICLP'91), MIT Press, pp 581–595
32. Dubois D, Lang J, Prade H (1994) Possibilistic logic. In: Gabbay DM, Hogger CJ, Robinson JA, Siekmann JH (eds) Handbook of Logic in Artificial Intelligence and Logic Programming - Volume 3: Nonmonotonic Reasoning and Uncertain Reasoning, Oxford University Press, Inc., New York, NY, USA, pp 439–513

33. Garcia L, Ngoma S, Nicolas P (2009) Dealing Automatically with Exceptions by Introducing Specificity in ASP. In: Sossai C, Chemello G (eds) Proceedings of the 10th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty, (ECSQARU'99), Lecture Notes in Artificial Intelligence, vol 5590, Springer-Verlag, Berlin, Heidelberg, pp 614–625
34. Gelfond M, Lifschitz V (1988) The stable model semantics for logic programming. In: Proceedings of the Fifth International Conference on Logic Programming, (ICLP'88), The MIT Press, pp 1070–1080
35. Kaci S (2011) Working with Preferences: Less Is More. Springer-Verlag
36. Kärger P, Lopes N, Olmedilla D, Polleres A (2008) Towards Logic Programs with Ordered and Unordered Disjunction. In: Proceedings of Workshop on Answer Set Programming and Other Computing Paradigms (ASPOCP2008), 24th International Conference on Logic Programming (ICLP 2008), pp 46–60
37. Lifschitz V, Pearce D, Valverde A (2001) Strongly equivalent logic programs. ACM Transactions on Computational Logic 2(4):526–541
38. Lloyd JW (1987) Foundations of logic programming, 2nd edn. Springer-Verlag New York, Inc., New York, NY, USA
39. Lukasiewicz T (2005) Weak nonmonotonic probabilistic logics. Artificial Intelligence 168(1-2):119–161
40. Newman MHA (1942) On Theories with a Combinatorial Definition of Equivalence. The Annals of Mathematics 43(2):223–243
41. Nicolas P, Garcia L, Stéphan I, Lefèvre C (2006) Possibilistic uncertainty handling for answer set programming. Annals of Mathematics and Artificial Intelligence 47(1-2):139–181
42. Nieves JC, Osorio M, Cortés U (2011) Semantics for Possibilistic Disjunctive Programs. Theory and Practice of Logic Programming (doi: 10.1017/S1471068411000408)
43. Osorio M, Nieves JC (2007) Pstable semantics for possibilistic logic programs. In: Gelbukh A, Morales AFK (eds) Proceedings of the artificial intelligence 6th Mexican international conference on Advances in artificial intelligence, MICAI'07, Springer-Verlag, Berlin, Heidelberg, pp 294–304
44. Osorio M, Navarro JA, Arrazola J (2001) Equivalence in Answer Set Programming. In: Pettorossi A (ed) Selected Papers from the 11th International Workshop on Logic Based Program Synthesis and Transformation, (LOPSTR 2001), Lecture Notes in Computer Science, vol 2372, Springer-Verlag, London, UK, pp 57–75
45. Saad E, Pontelli E (2005) Hybrid Probabilistic Logic Programs with Nonmonotonic Negation. In: Gabbrielli M, Gupta G (eds) Proceedings of 21st International Conference on Logic Programming, (ICLP'05), Lecture Notes in Computer Science, vol 3668, Springer Berlin / Heidelberg, New York, US, pp 204–220
46. Dupin de Saint-Cyr F, Prade H (2008) Handling uncertainty and defeasibility in a possibilistic logic setting. International Journal of Approximate Reasoning 49(1):67–82
47. Schaub T, Wang K (2001) A comparative study of logic programs with preference. In: Nebel B (ed) Proceedings of the 17th International Joint Conference on Artificial Intelligence - Volume 1, (IJCAI'01), Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp 597–602

48. Tarski A (1955) A Lattice-Theoretical Fixpoint Theorem and its Applications. Pacific Journal of Mathematics 5(2):285–309

A Result Proofs

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 is called the set of possibilistic consequences of P and it is denoted by $\Pi Cn(P)$.*

Proof We can observe that the operator T_P is monotonic. Therefore, by Tarsky's theorem [48], T_P reaches a fix-point (as also observed in the proof of Proposition 8 in [41]). □

Proposition 2 *Let P be an LPPOD and M be a set of possibilistic atoms. If M is a possibilistic answer set of P then M^* is an answer set of P^*.*

Remark 1 $(P_\times^{M^*})^* = ((P^*)_\times^{M^*})$

Proof If M is a possibilistic answer set of P, then it is a possibilistic answer set of $\Pi Cn(P_\times^{M^*})$ (by Definition 10). Thus, M^* is answer set of $((P^*)_\times^{M^*})$ (by Definition 2 of answer set of an LPOD), so M^* is an answer set of P^*. □

Proposition 3 *Let P be an LPPOD, and M^* is an answer set of P^*. If $\forall r \in P$, $Nec(r) = \mathbf{1}$, then $M = \{(a, 1) \mid \varphi \in M^*\}$ is a possibilistic answer set of P.*

Proof By Definition 10, we know that if M^* is answer set of P^*, then $M = \Pi Cn(P_\times^{M^*})$ is a possibilistic answer set of P. So we have to prove that the necessity value of the atoms of the possibilistic answer set of the possibilistic definite logic program $P_\times^{M^*}$ corresponds to $\mathbf{1}$. By the possibilistic consequence operator definition (Definition 9), we know that given a set of possibilistic atoms M, the applicability degree β of a rule in M captures the necessity of the conclusion that the rule can produce with respect to M. Given a rule $r = 1 : c \leftarrow \mathcal{B}^+$, we consider the following cases:

Case (i): $\mathcal{B}^+ = \emptyset$, then the rule is 1-applicable in M
Case (ii) $\mathcal{B}^+ \nsubseteq M$, then the rule is 0-applicable in M.
Case (iii): $\mathcal{B}^+ \subseteq M$, then the rule is 1-applicable in M since $min\{\alpha, \alpha_1, \ldots, \alpha_m\} = 1$ where $\forall(b_i, \alpha_i) \in \mathcal{B}^+$, $1 \leq i \leq m$, $(b_i, \alpha_i) \sqsubseteq M$ and $\alpha = \alpha_1 = \ldots = \alpha_m = 1$.

The necessity values of the atoms contained in the fix-point reached by the possibilistic consequence operator (Definition 9) are inferred by

$$\begin{aligned}\Pi T_P(M) = \quad & \{(a, \delta) | a \in head(P^*), App(P, M, a) \neq \emptyset, \\ & \delta = \max_{r \in App(P,M,a)}\{\beta | r \text{ is } \beta\text{-applicable in } M\}\}\end{aligned}$$

but all applicable rules are $\mathbf{1}$-applicable in M. Thus, $\delta = 1$. Hence, $M = \{(a, 1) \mid a \in M^*\}$ is a possibilistic answer set of P. □

Proposition 4 *Let P be an LPPOD and M be a set of possibilistic atoms such that M^* is an answer set of P^*. Then*

$$\Pi M(P_\times^{M^*}) = \{(a, N_{P_\times^{M^*}}(a)) | a \in \mathcal{L}_{P_\times^{M^*}}, N_{P_\times^{M^*}}(a) > 0\}$$

is a possibilistic answer set of P.

Remark 2 $P_{\times}^{M^*}$ is a possibilistic definite logic program.

Proof By Definition 10, given that M^* is an answer set of P^*, M is a possibilistic answer set of P if and only if $M = \Pi Cn(P_{\times}^{M^*})$. Hence, we have to prove that $\Pi(P_{\times}^{M^*}) = \Pi Cn(P_{\times}^{M^*})$. This follows Theorem 1 in [41] in a straightforward way since $P_{\times}^{M^*}$ is a possibilistic definitive logic programs. Thus, M is a possibilistic answer set of P. □

Lemma 1 *Let P and P' be two LPPODs related by any transformation in $\mathcal{CS}_{LPPOD}$. Then $SEM_{LPPOD}(P) \equiv_p SEM_{LPPOD}(P')$.*

Definition 21 (Rewriting System for LPODs) Let P be an LPOD and $\mathcal{CS}_{LPOD}$ be the rewriting system based on transformation rules $\{\rightarrow_C, \rightarrow_{RED^+}, \rightarrow_{RED^-}, \rightarrow_S, \rightarrow_F, \rightarrow_{Loop}\}$, where each transformation is defined by redefining $\mathcal{CS}_{LPPOD}$ omitting the necessity values associated with programs rules in each transformation.

Remark 3 It can be noticed how $SEM_{LPPOD}(P)^* \equiv_p SEM_{LPPOD}(P')^*$ is equivalent to $SEM_{LPOD}(P^*) \equiv SEM_{LPOD}(P'^*)$, where SEM_{LPOD} is the LPODs semantics (Definition 2), and $\equiv$ denotes the equivalence relation *w.r.t.* the LPODs semantics.

Remark 4 Let P be a definite logic program, the least model M of P can be computed as the least fix-point of the consequence operator $T_P : 2^{\mathcal{L}_P} \rightarrow 2^{\mathcal{L}_P}$ such that $T_P(M) = head(App(P, M))$.

Proof To prove that $SEM_{LPPOD}(P) \equiv_p SEM_{LPPOD}(P')$, we have to show that given two LPPODs P and P' related by any transformation $\{\rightarrow_{PC}, \rightarrow_{PRED^+}, \rightarrow_{PRED^-}, \rightarrow_{PS}, \rightarrow_{PF}, \rightarrow_{PLoop}\}$, it holds that $\Pi Cn(P_{\times}^{M^*}) = \Pi Cn(P_{\times}^{'M^*})$. As the LPPODs semantics is a generalization of the LPOD semantics, we prove this lemma by first proving that the transformation rules for LPODs are invariant *w.r.t.* the LPODs semantics.

Let us consider P^*, P'^*, and $\mathcal{CS}_{LPOD}$ (Definition 21). By Remark 3, we have to show that $SEM_{LPOD}(P_{\times}^{*M^*}) = SEM_{LPOD}((P')_{\times}^{*M^*}))$. We can observe that $P_{\times}^{*M^*}$, and $(P')_{\times}^{*M^*}$ are definite logic programs. From [13] it is known that the stable semantics is closed *w.r.t.* the rewriting system $\mathcal{CS}'_{LPOD} = \mathcal{CS}_{LPOD} \backslash (\rightarrow_{Loop})$ and in [44] it has been proven that the transformation rule $\rightarrow_{Loop}$ for disjunctive programs also preserves the stable semantics. In the case of LPODs, the stable semantics is also closed under $\rightarrow_{Loop}$. This is straightforward to see since we can notice that $\forall M \in SEM_{LPOD}(P)$, $M \cap unf(P) = \emptyset$ by construction (Definition 19), *i.e.* only program rules which contain false atoms are removed by $\rightarrow_{Loop}$. Therefore, the LPOD semantics is closed under $\mathcal{CS}_{LPOD}$.

Then, we can see in a straightforward way that the LPPODs semantics is closed under the transformation rules in the rewriting system $\mathcal{CS}'_{LPPOD} = \mathcal{CS}_{LPPOD} \backslash (\rightarrow_{PS})$ since the transformation rules $\{\rightarrow_{PC}, \rightarrow_{PRED^+}, \rightarrow_{PRED^-}, \rightarrow_{PF}, \rightarrow_{PLoop}\}$ do not affect the necessity values associated with program rules.

The only transformation that affects necessity values is the *Possibilistic Success*. Hence, we have to prove that given two LPPODs P and P', $\rightarrow_{PS}$, and M^* answer set of P^*, $\Pi Cn(P_{\times}^{M^*})$ $= \Pi Cn(P_{\times}^{'M^*})$. For a better visibility, let us call $\Pi Cn(P_{\times}^{M^*}) = M_1$ and $\Pi Cn(P_{\times}^{'M^*}) = M_2$.

Now, let us suppose $P = P_1 \cup \{\alpha : \mathcal{C}^{\times} \leftarrow b.\ ;\ \beta : b.\}$ and $P' = P_1 \cup \{min\{\alpha, \beta\} : \mathcal{C}^{\times}.\ ;\ \beta : b.\}$. By applying the $\times$-possibilistic reduction using M^* to both P and P', we obtain $P_{\times}^{M^*} = P_{1_{\times}}^{M^*} \cup \{\alpha : c \leftarrow b.\ ;\ \beta : b.\}$ and $P_{\times}^{'M^*} = P_{1_{\times}}^{M^*} \cup \{min\{\alpha, \beta\} : \mathcal{C}^{\times}.\ ;\ \beta : b.\}$. It can be observed that $P_{1_{\times}}^{M^*}$ is the same subprogram.

Then, we can observe that given the answer set semantics definition in terms of fix-point of the immediate consequence operator (Remark 4) and that $\rightarrow_S$ is closed under answer set semantics, since M^* is an answer set of P^*, then $(\Pi Cn(P_{\times}^{M^*}))^* = (\Pi Cn(P_{\times}^{'M^*}))^*$, *i.e.* $M_1^* = M_2^*$.

To prove $\Pi Cn(P_{\times}^{M^*}) = \Pi Cn(P_{\times}^{'M^*})$, let us suppose by contradiction that $M_1 \neq M_2$. Since $M_1^* = M_2^*$, this basically means that M_1 and M_2 have the same atoms but they can differ in at least in one of the necessity values associated with their atoms. Let us consider P^*, and P'^*. Without loss of generality let us assume that $P_1 = \emptyset$ and thus $P_1^* = \emptyset$ as well. This means that $M_1^* = M_2^* = \{(c,b)\}$. However, by assuming $M_1 \neq M_2$, then $M_1 = \{(c,\delta),(b,\beta)\}$, and $M_2 = \{(c,\delta'),(b,\beta)\}$, where $\delta \neq \delta'$.

But if we apply the fix-point operator to $P_\times^{M^*}$ we obtain:

$\Pi T^0_{P^M_\times} = \emptyset$
$\Pi T^1_{P^M_\times} = \Pi T_{P^M_\times}(\emptyset) = \{(b,\beta)\}$
$\Pi T^2_{P^M_\times} = \Pi T_{P^M_\times}(\{(b,\beta)\}) = \{(b,\beta),(c,\min\{\alpha,\beta\})\}$
$\Pi T^3_{P^M_\times} = \Pi T_{P^M_\times}(\{(b,\beta),(c,\min\{\alpha,\beta\})\}) = \{(b,\beta),(c,\min\{\alpha,\beta\})\}$

And if we apply the fix-point operator to $P_\times^{'M^*}$ we obtain:

$\Pi T^0_{P'^M_\times} = \emptyset$
$\Pi T^1_{P'^M_\times} = \Pi T_{P^M_\times}(\emptyset) = \{(b,\beta)\}$
$\Pi T^2_{P'^M_\times} = \Pi T_{P^M_\times}(\{(b,\beta)\}) = \{(b,\beta),(c,\min\{\alpha,\beta,\min\{\alpha,\beta\}\})\}$
$\Pi T^3_{P'^M_\times} = \Pi T_{P^M_\times}(\{(b,\beta),(c,\min\{\alpha,\beta,\min\{\alpha,\beta\}\})\}) = \{(b,\beta),(c,\min\{\alpha,\beta,\min\{\alpha,\beta\}\})\}$

Thus, $M_1 = \{(b,\beta),(c,\min\{\alpha,\beta\})\}$, and $M_2 = \{(b,\beta),(c,\min\{\alpha,\beta,\min\{\alpha,\beta\}\})\}$. But the necessity values associated with the atoms are the same and no other rules may have influenced the necessity values of the atoms themselves, as we have assumed that $P_1 = \emptyset$. This contradicts the hypothesis that $M_1 \neq M_2$. Thus, the LPPODs semantics is closed under CS_{LPPOD}.

□

Theorem 1 *Let P be an LPPOD. Then $\mathcal{CS}_{LPPOD}$ is confluent, noetherian and norm$_{\mathcal{CS}_{LPPOD}}$ (P) is unique.*

Remark 5 Let $CS = \langle S, \rightarrow \rangle$ be an abstract rewriting system, $\rightarrow$ be a binary relation on a given set S, and $\rightarrow^*$ be the reflexive and transitive closure of $\rightarrow$. Then, a rewriting system is [27]:

noetherian: if $\nexists x_1 \rightarrow x_2 \rightarrow \ldots \rightarrow x_i \rightarrow x_{i+1} \rightarrow \ldots$, where $\forall i\ x_i \neq x_{i+1}$,
confluent: if $\forall u \in S$ such that $u \rightarrow^* x$ and $u \rightarrow^* y$ then $\exists z \in S$ such that $x \rightarrow^* z$ and $y \rightarrow^* z$,
locally confluent: if $\forall u \in S$ such that $u \rightarrow x$ and $u \rightarrow y$ then $\exists z \in S$ such that $x \rightarrow^* z$ and $y \rightarrow^* z$.

If CS is noetherian and confluent, every element x reduces to a unique normal form [27].

Remark 6 Let T_1 and T_2 be two transformation rules, and P, P_1, P_2, and P_3 be logic programs. T_1 and T_2 *commute*, if $P \rightarrow_{T_1} P_1$ and $P \rightarrow_{T_2} P_2$ then $P_1 \rightarrow_{T_2} P_3$ and $P_2 \rightarrow_{T_1} P_3$.

Remark 7 Let T_1 and T_2 be two transformation rules, and P, P_1, P_2 be logic programs. T_1 *absorbs* T_2, if $P \rightarrow_{T_1} P_1$ and $P \rightarrow_{T_2} P_2$ then $P_2 \rightarrow_{T_1} P_1$.

Proof We have to prove that $\mathcal{CS}_{LPPOD}$ is confluent and noetherian, as from rewriting system theory it is known that if a rewriting system is confluent and noetherian then its normal form always exists and it is unique (Remark 5). This means that we have to prove that:

- $\mathcal{CS}_{LPPOD}$ is noetherian
- $\mathcal{CS}_{LPPOD}$ is confluent

We can observe in a straightforward way that $\mathcal{CS}_{LPPOD}$ is noetherian, since all our program transformations decrease the size of an LPPOD.

The confluence of $\mathcal{CS}_{LPPOD}$ can be proven using local confluence as stated by Newman's lemma [40]. According to [40], a noetherian rewriting system is confluent if it is locally confluent. Since commutation and absorption are special cases of local confluence, proving that $\mathcal{CS}_{LPPOD}$ is locally confluent amounts to show that for each pair of transformations in $\mathcal{CS}_{LPPOD}$ either the two transformations commute (Remark 6) or one transformation absorbs the other one (Remark 7). To this end, let us suppose that we have an LPPOD P and we apply a transformation to get P_1. Let us also suppose that from P it is possible to apply another transformation to get P_2. Thus, we need to show that P_3 exists in such a way that P_1 can arrive to P_3 by using some transformations (possibly none) and that P_2 can arrive to P_3 by using of some transformations (possibly none). Then, we have to consider the following pairs of transformations:

Case 1: $\rightarrow_{PC}$ and $\rightarrow_{PS}$ commute. Let P be:

$$\begin{array}{ll} P: & \boldsymbol{\alpha_1} : \mathit{Head}_1 \leftarrow c, \mathit{not}\ c,\ \mathit{Body}_1. \\ & \boldsymbol{\alpha_2} : \mathit{Head}_2 \leftarrow e,\ \mathit{Body}_2. \\ & \boldsymbol{\alpha_3} : e. \\ & P' \end{array}$$

where, $\alpha_1 > \alpha_2 > \alpha_3$ are necessity values in $\mathcal{S} = \{\alpha_3, \alpha_2, \alpha_1\}$, P' denotes the rest of the rules in P, Head_1 and Head_2 denote rules heads, and Body_1, Body_2 denote the rest of the rules body.

Thus, if we apply $\rightarrow_{PS}$ to P, then we obtain P_1 and, if we apply $\rightarrow_{PC}$ to P, then we obtain P_2:

$$\begin{array}{llll} P_1: & \boldsymbol{\alpha_1} : \mathit{Head}_1 \leftarrow c, \mathit{not}\ c,\ \mathit{Body}_1. & P_2: & \\ & \boldsymbol{\alpha_3} : \mathit{Head}_2 \leftarrow \mathit{Body}_2. & & \boldsymbol{\alpha_2} : \mathit{Head}_2 \leftarrow e,\ \mathit{Body}_2. \\ & \boldsymbol{\alpha_3} : e. & & \boldsymbol{\alpha_3} : e. \\ & P' & & P' \end{array}$$

Now, if we apply $\rightarrow_{PC}$ to P_1, then we obtain P_3, and if we apply $\rightarrow_{PS}$ to P_2, then we obtain P_3:

$$\begin{array}{llll} P_3: & & P_3: & \\ & \boldsymbol{\alpha_3} : \mathit{Head}_2 \leftarrow \mathit{Body}_2. & & \boldsymbol{\alpha_3} : \mathit{Head}_2 \leftarrow \mathit{Body}_2. \\ & \boldsymbol{\alpha_3} : e. & & \boldsymbol{\alpha_3} : e. \\ & P' & & P' \end{array}$$

Thus, both transformation rules $\rightarrow_{PC}$ and $\rightarrow_{PS}$ commute.

Case 2: $\rightarrow_{PC}$ and $\rightarrow_{PRED^-}$ commute. Let P be:

$$\begin{array}{ll} P: & \boldsymbol{\alpha_1} : \mathit{Head}_1 \leftarrow c, \mathit{not}\ c,\ \mathit{Body}_1. \\ & \boldsymbol{\alpha_2} : \mathit{Head}_2 \leftarrow \mathit{not}\ e,\ \mathit{Body}_2. \\ & \boldsymbol{\alpha_3} : e. \\ & P' \end{array}$$

Thus, if we apply $\rightarrow_{PC}$ to P, then we obtain P_1, and if we apply $\rightarrow_{PRED^-}$ to P, then we obtain P_2:

$$\begin{array}{llll} P_1: & & P_2: & \boldsymbol{\alpha_1} : \mathit{Head}_1 \leftarrow c, \mathit{not}\ c,\ \mathit{Body}_1. \\ & \boldsymbol{\alpha_2} : \mathit{Head}_2 \leftarrow \mathit{not}\ e,\ \mathit{Body}_2. & & \\ & \boldsymbol{\alpha_3} : e. & & \boldsymbol{\alpha_3} : e. \\ & P' & & P' \end{array}$$

Now, if we apply $\rightarrow_{PRED^-}$ to P_1, then we obtain P_3, and if we apply $\rightarrow_{PC}$ to P_2, then we obtain P_3

$$\begin{array}{llll} P_3: & & P_3: & \\ & & & \\ & \boldsymbol{\alpha_3} : e. & & \boldsymbol{\alpha_3} : e. \\ & P' & & P' \end{array}$$

Thus, both transformation rules $\rightarrow_{PC}$ and $\rightarrow_{PRED^-}$ commute.

Case 3: $\rightarrow_{PLoop}$ absorbs $\rightarrow_{PC}$. Let P be:

$$\begin{array}{ll} P: & \boldsymbol{\alpha_1} : a \leftarrow b, \mathit{not}\ b. \\ & \boldsymbol{\alpha_2} : c \leftarrow c. \\ & \boldsymbol{\alpha_3} : d. \end{array}$$

Thus, if we apply $\rightarrow_{PLoop}$ to P, then we obtain P_1 (since $\mathit{unf}(P) = \{a, b, c\}$). If we apply $\rightarrow_{PC}$ to P, then we obtain P_2:

$$\begin{array}{llll} P_1: & & P_2: & \\ & & & \boldsymbol{\alpha_2} : c \leftarrow c. \\ & \boldsymbol{\alpha_3} : d. & & \boldsymbol{\alpha_3} : d. \end{array}$$

Now, if we apply $\rightarrow_{PLoop}$ to P_2, then we obtain P_1 (since $\mathit{unf}(P_2) = \{c\}$):

$$\begin{array}{llll} P_1: & & P_1: & \\ & & & \\ & \boldsymbol{\alpha_3} : d. & & \boldsymbol{\alpha_3} : d. \end{array}$$

Thus, $\rightarrow_{PLoop}$ absorbs $\rightarrow_{PC}$ (it can be shown how $\rightarrow_{PLoop}$ and $\rightarrow_{PC}$ also commute.)

Table 3 Summary of Transformation Local Confluence

Case	Pair of Transformations	Locally Confluent by
1	$\rightarrow_{PC}, \rightarrow_{PS}$	commute
2	$\rightarrow_{PC}, \rightarrow_{PRED^-}$	commute
3	$\rightarrow_{PLoop}, \rightarrow_{PC}$	absorb/commute
4	$\rightarrow_{PRED^+}, \rightarrow_{PRED^-}$	absorb/commute
5	$\rightarrow_{PC}, \rightarrow_{PRED^+}$	absorb/commute
6	$\rightarrow_{PC}, \rightarrow_{PF}$	commute
7	$\rightarrow_{PRED^+}, \rightarrow_{PF}$	commute
8	$\rightarrow_{PRED^+}, \rightarrow_{PS}$	commute
9	$\rightarrow_{PLoop}, \rightarrow_{PRED^+}$	absorb/commute
10	$\rightarrow_{PRED^-}, \rightarrow_{PF}$	commute
11	$\rightarrow_{PRED^-}, \rightarrow_{PS}$	absorb/commute
12	$\rightarrow_{PRED^-}, \rightarrow_{PLoop}$	absorb/commute
13	$\rightarrow_{PF}, \rightarrow_{PS}$	absorb/commute
14	$\rightarrow_{PLoop}, \rightarrow_{PF}$	absorb
15	$\rightarrow_{PS}, \rightarrow_{PLoop}$	commute

Case 4: $\rightarrow_{PRED^+}$ absorbs $\rightarrow_{PRED^-}$. Let P be:

P: $\alpha_{\mathbf{1}} : Head_1 \leftarrow not\ b, not\ p,\ Body_1.$
$\alpha_{\mathbf{2}} : p.$
P'

Let us suppose that $b \notin HEAD(P)$. If we apply $\rightarrow_{PRED^-}$ to P, then we obtain P_1 and, if we apply $\rightarrow_{PRED^+}$ to P, then we obtain P_2:

P_1:		P_2:	$\alpha_{\mathbf{1}} : Head_1 \leftarrow not\ p,\ Body_1$
	$\alpha_{\mathbf{2}} : p.$		$\alpha_{\mathbf{2}} : p.$
	P'		P'

Now, if we apply $\rightarrow_{PRED^-}$ to P_2, then we obtain P_1:

P_1:		P_1:	
	$\alpha_{\mathbf{2}} : p$		$\alpha_{\mathbf{2}} : p.$
	P'		P'

Thus, $\rightarrow_{PRED^-}$ absorbs $\rightarrow_{PRED^+}$ (it can be shown how $\rightarrow_{PRED^-}$ and $\rightarrow_{PRED^+}$ also commute.)

In a similar way, it can be confirmed that the following pairs of transformations are locally confluent in the remaining cases:
Case 5: $\rightarrow_{PC}$ absorbs $\rightarrow_{PRED^+}$. They also commute.
Case 6: $\rightarrow_{PC}$ and $\rightarrow_{PF}$ commute.
Case 7: $\rightarrow_{PRED^+}$ and $\rightarrow_{PF}$ commute.
Case 8: $\rightarrow_{PRED^+}$ and $\rightarrow_{PS}$ commute.
Case 9: $\rightarrow_{PLoop}$ absorbs $\rightarrow_{PRED^+}$. They also commute.
Case 10: $\rightarrow_{PRED^-}$ and $\rightarrow_{PF}$ commute.
Case 11: $\rightarrow_{PRED^-}$ absorbs $\rightarrow_{PS}$. They also commute.
Case 12: $\rightarrow_{PLoop}$ absorbs $\rightarrow_{PRED^-}$. They also commute.
Case 13: $\rightarrow_{PF}$ absorbs $\rightarrow_{PS}$. They also commute.
Case 14: $\rightarrow_{PLoop}$ absorbs $\rightarrow_{PF}$.
Case 15: $\rightarrow_{PS}$ and $\rightarrow_{PLoop}$ commute.

Table 3 summarizes all pairs of transformations and it shows whether they absorb and/or commute.
Finally, as CS_{LPPOD} is noetherian and as all pair of transformations in CS_{LPPOD} are locally confluent, CS_{LPPOD} is confluent.

□

Proposition 5 *Let M_1 and M_2 be possibilistic answer sets of an LPPOD P, $>_p$ be the preference relation for LPODs (Definition 4), and $>_{pp}$ be the possibilistic preference relation. If $M_1^* >_p M_2^*$ then $M_1 >_{pp} M_2$.*

Remark 8 $M_1^* >_p M_2^*$ iff $\exists\ r \in P^*$ such that $deg_{M_1^*}(r) < deg_{M_2^*}(r)$, and $\nexists r' \in P^*$ such that $deg_{M_2^*}(r') < deg_{M_1^*}(r')$ (Definition 4).

Remark 9 $(norm_{\mathcal{LPPOD}}(P)^*) = norm_{\mathcal{LPOD}}(P^*)$

Proof Let us suppose that $M_1 \not>_{pp} M_2$. By contradiction, we have two cases: either (i) $M_2 >_{pp} M_1$ or (ii) $M_2 \not>_{pp} M_1$ (and $M_1 \not>_{pp} M_2$).

Case 1: $M_2 >_{pp} M_1$. If $M_2 >_{pp} M_1$ it means that $\exists r \in norm_{\mathcal{CS}_{LPPOD}}(P)$ such that $deg_{M_2^*}(r^*) < deg_{M_1}(r^*)$, and $\nexists r' \in norm_{\mathcal{CS}_{LPPOD}}(P)$ such that $deg_{M_1^*}(r'^*) < deg_{M_2^*}(r'^*)$ and $Nec(r) < Nec(r')$. By Remark 9 this would mean that $\exists r^* \in norm_{\mathcal{LPOD}}(P^*)$ such that $deg_{M_2^*}(r^*) < deg_{M_1^*}(r^*)$, and $\nexists r'^* \in norm_{\mathcal{LPOD}}(P^*)$ such that $deg_{M_1^*}(r'^*) < deg_{M_2^*}(r'^*)$, *i.e.* $M_2^* >_p M_1^*$. But this contradicts the hypothesis $M_1^* >_p M_2^*$ (Remark 8).

Case (ii): $M_2 \not>_{pp} M_1$. If $M_2 \not>_{pp} M_1$ (and $M_1 \not>_{pp} M_2$) then it means that neither $\exists r \in norm_{\mathcal{CS}_{LPPOD}}(P)$ such that $deg_{M_2^*}(r^*) < deg_{M_1^*}(r^*)$, and $\nexists r' \in norm_{\mathcal{CS}_{LPPOD}}(P)$ such that $deg_{M_1^*}(r'^*) < deg_{M_2}(r'^*)$ and $Nec(r) < Nec(r')$, nor $\exists r \in norm_{\mathcal{CS}_{LPPOD}}(P)$ such that $deg_{M_1^*}(r^*) < deg_{M_2^*}(r^*)$, and $\nexists r' \in norm_{\mathcal{CS}_{LPPOD}}(P)$ such that $deg_{M_2^*}(r'^*) < deg_{M_1^*}(r'^*)$ and $Nec(r) < Nec(r')$. This is only true if $\forall r \in P$, $deg_{M_1^*}(r^*) = deg_{M_2^*}(r^*)$ and $\exists r' \in P$ such that $deg_{M_1^*}(r^*) = deg_{M_2^*}(r^*)$ and $Nec(r) = Nec(r')$. However by Remark 9 this would mean that $\forall r^* \in norm_{\mathcal{LPOD}}(P^*)$, $deg_{M_1^*}(r^*) = deg_{M_2^*}(r^*)$ and thus $M_2^* \not>_p M_1^*$ and $M_1^* \not>_p M_2^*$. But this contradicts the hypothesis $M_1^* >_p M_2^*$ (Remark 8). □

Theorem 2 *Let P be an LPPOD, such that $P \subseteq Prog_{\langle \mathcal{A},\mathcal{S}\rangle}$, where $Prog_{\langle \mathcal{A},\mathcal{S}\rangle}$ is the set of all LPPODs which can be generated by a finite set of atoms $\mathcal{A}$ and a finite set of necessity values $\mathcal{S}$. Then, the LPPODs semantics is computed by the function $SEM_{LPPOD} : P \to \mathcal{PS}$.*

Proof The proof is straightforward since Algorithm 1 computes the LPPODs semantics. □

B Possibilistic Logic

Possibilistic logic emanates from possibility theory and it was developed as a sound and complete logic system which extends classical logic for representing qualitative uncertainty [32].

At the semantic level, possibilistic logic is defined in terms of a possibility distribution π on the universe of interpretations which represents the compatibility of an interpretation ω with available information (or beliefs) about the real world. By convention $\pi(\omega) = 0$ means that ω is impossible and $\pi(\omega) = 1$ means that nothing prevents ω for being true in the real world. A possibility distribution π induces two dual measures grading respectively the possibility and certainty of a formula φ. The *possibility measure* Π, defined as $\Pi(\varphi) = max\{\pi(\omega)|\omega \vDash \varphi\}$, is the possibility degree which evaluates the extent to which φ is consistent with the available beliefs. The *necessity measure* N, defined as $N(\varphi) = 1 - \Pi(\neg\varphi)$, is the certainty degree which evaluates the extent to which φ is entailed by the available beliefs.

At the syntactic level in the necessity-valued case, possibilistic logic handles *necessity-valued formula* of the form (φ, α) where φ is a classical logic formula and $\alpha \in (0, 1]$ is interpreted as a lower bound of a necessity measure N expressing that the formula φ is *certain* at least to the level α. Given a possibilistic knowledge base Σ as a finite set of necessity-valued formula, *i.e.* $\Sigma = \{(\varphi_i, \alpha_i)|1 \leq i \leq n\}$ there can be many distributions satisfying the constraints in the knowledge base (*i.e.* $N(\varphi_i) \geq \alpha_i$). However, in practice one is interested in the least specific distribution only, since it is the distribution that makes the least possible number of assumptions. Formally a possibility distribution π is said to be the least specific one between all compatible distributions if there is no possibility distribution π' with $\pi \neq \pi'$ compatible with Σ such that $\forall\omega$, $\pi'(\omega) \geq \pi(\omega)$. The least specific distribution always exists and is characterised by $\pi_\Sigma(\omega) = min_{1\leq i\leq n}\pi_{(\varphi_i,\alpha_i)}(\omega)$ with $\pi_{(\varphi_i,\alpha_i)}(\omega) = 1$ if $\omega \vDash \varphi_i$, and $\pi_{(\varphi_i,\alpha_i)}(\omega) = 1 - \alpha_i$ if $\omega \nvDash \varphi_i$.

A possibilistic knowledge base is a compact representation of the possibility distribution defined in the interpretations representing the information. Indeed the treatment of the base in terms of formula and necessity values (syntactical way) leads to the same results of the treatment done in terms of interpretations and possibility distribution (semantical way) [32].