Ann Math Artif Intell (2006) 47: 139–181
DOI 10.1007/s10472-006-9029-y

Possibilistic uncertainty handling for answer set programming

Pascal Nicolas · Laurent Garcia · Igor Stéphan · Claire Lefèvre

Received: 1 February 2006 / Accepted: 23 June 2006 /
Published online: 16 September 2006
© Springer Science + Business Media B.V. 2006

Abstract In this work, we introduce a new framework able to deal with a reasoning that is at the same time non monotonic and uncertain. In order to take into account a certainty level associated to each piece of knowledge, we use possibility theory to extend the non monotonic semantics of stable models for logic programs with default negation. By means of a possibility distribution we define a clear semantics of such programs by introducing what is a possibilistic stable model. We also propose a syntactic process based on a fix-point operator to compute these particular models representing the deductions of the program and their certainty. Then, we show how this introduction of a certainty level on each rule of a program can be used in order to restore its consistency in case of the program has no model at all. Furthermore, we explain how we can compute possibilistic stable models by using available softwares for Answer Set Programming and we describe the main lines of the system that we have developed to achieve this goal.

Keywords non monotonic reasoning · uncertainty · logic programming · possibility theory · answer set programming

PACS 68T30 · 68T27 · 68T37 · 68N17

P. Nicolas (✉) · L. Garcia · I. Stéphan · C. Lefèvre
LERIA, University of Angers, Angers, France
e-mail: pascal.nicolas@univ.angers.fr

L. Garcia
e-mail: laurent.garcia@univ-angers.fr

I. Stéphan
e-mail: igor.stephan@univ.angers.fr

C. Lefèvre
e-mail: claire.lefevre@univ-angers.fr

1 Introduction

Answer set programming (ASP) is an appropriate formalism to represent various problems issued from Artificial Intelligence and arising when available information is incomplete as in non monotonic reasoning, planning, diagnosis... It is also a very convenient framework to encode and solve combinatorial problems since some efficient operational systems are available today to deal with ASP. From a global view, ASP is a general paradigm covering different declarative semantics for different kind of logic programs. Whatever the precise framework, information is encoded by logical rules and solutions are obtained by sets of models. Each model is a minimal set of atoms (or literals) containing sure informations (some facts) and deductions obtained by applying by default some rules. So, conclusions rely on present and absent informations, they form a coherent set of hypotheses and represent a rational view on the world described by the rules. Thus, in whole generality there is not a unique set of conclusions but maybe many ones and each conclusion is no longer absolutely sure but only plausible and more or less certain. This refers to the major features of default reasoning as it can be described using default logic [41], from which stable models semantics for normal logic programs, used in this paper, can be seen as a reduction.

Possibilistic logic is issued from Zadeh's possibility theory [49], which offers a framework for representation of states of partial ignorance owing to the use of a dual pair of possibility and necessity measures. Possibility theory may be quantitative or qualitative [19, 20] according to the range of these measures which may be the real interval [0, 1], or a finite linearly ordered scale as well. Possibilistic logic [18] provides a sound and complete machinery for handling qualitative uncertainty with respect to a semantics expressed by means of possibility distributions which rank-order the possible interpretations. Let us mention that in possibilistic logic we deal with uncertainty by means of classical two-valued (true or false) interpretations that can be more or less certain, more or less possible. We are not concerned by vagueness representation in a multi-valued framework but we stay in the framework of classical logic to which we add a way to graduate the confidence we have in each proposed information. See [21] for further comments about the representation of certainty levels in a logical framework and the difference between multi-valued logics, probability theory and possibility theory.

The aim of our work is to propose a way to join together in only one formalism default reasoning and reasoning under uncertainty. That's why we want to introduce concepts of possibility theory within stable models semantics like it is summarized in figure 1. To illustrate our work, let us take an example of a normal logic program which does not represent any notions of uncertainty. It encodes medical treatment in which a patient is suffering from two diseases. Each disease can be cured by one drug but the two drugs are incompatible. The program

$$P_{med} = \left\{ \begin{array}{ll} dr1 \leftarrow di1, not\ dr2. & dr2 \leftarrow di2, not\ dr1. \\ c1 \leftarrow dr1, di1. & c2 \leftarrow dr2, di2. \\ di1 \leftarrow . & di2 \leftarrow . \end{array} \right\}$$

means that the drug $dr1$ (resp. $dr2$) is given to a patient if he suffers from the disease $di1$ (resp. $di2$) except if he takes the drug $dr2$ (resp. $dr1$); if a patient suffers from

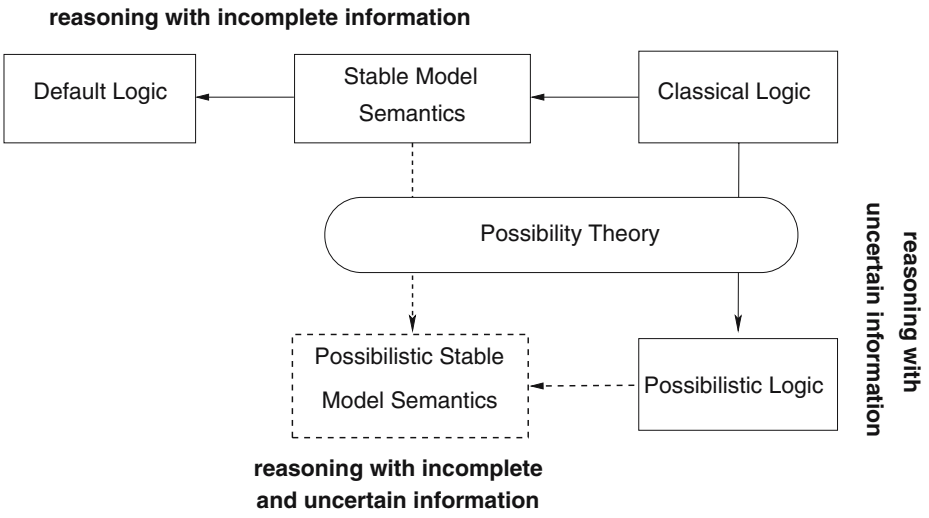


Figure 1 Possibilistic uncertainty handling for Answer Set Programming

disease $di1$ (resp. $di2$) and takes the drug $dr1$ (resp. $dr2$) then he is cured $c1$ (resp. $c2$) of this disease; the patient suffers from the diseases $di1$ and $di2$. From this program, two stable models $\{di1, di2, dr1, c1\}$ and $\{di1, di2, dr2, c2\}$ are obtained. This means that the patient can be cured of one of his two diseases but not of both of them.

Nevertheless, it may seem interesting for a doctor to be able to evaluate what choice to do between these two treatments which are incompatible. One of the criterion for this choice may be the efficiency of each medical treatment : are $c1 \leftarrow dr1, di1$ and $c2 \leftarrow dr2, di2$ absolutely certain rules? Another point may be the confidence the doctor has in his diagnosis : are $di1$ and $di2$ surely established? That is why we propose to address this problem by the help of certainty degrees affected to rules. These degrees should be taken into account during the inferential mechanism in order to determine the level of certainty of each conclusion and allow the doctor to compare them. In this work we show how to achieve our aim by help of possibilistic logic.

This paper aggregates our previous separate works [36–38] and gives the formal proofs of all results in the appendix 9. Section 2 recalls fundamental notions about stable model semantics and possibilistic logic. Sections 3 and 4 are the core of the paper and define our new framework. First, we deal in section 3 with definite logic programs (without default negation). We show how it is possible to reason both in semantical and syntactical ways, these two aspects are shown to be equivalent. Then, in section 4, we follow the same schema when dealing with normal logic programs (with default negations) and we define *possibilistic stable models*. In section 5, we focus on the problem of inconsistency arising when a program has no model at all. In section 6, we present the system that we have developed to compute the possibilistic stable models of a given program. In section 7, we compare our framework with other works dealing with similar ideas.

2 Theoretical backgrounds

2.1 Stable model semantics

ASP is concerned by different kinds of logic programs and different semantics. In our work we deal with *normal logic programs*, interpreted by *stable model semantics* [22]. We consider given a non empty set of atoms $\mathcal{X}$ that determines the language of the programs.[1] A normal logic program is a set of rules of the form:

$$c \leftarrow a_1, \ \ldots, \ a_n, \ not\ b_1, \ \ldots, \ not\ b_m.$$

where $n \geq 0, m \geq 0, \{a_1, \ldots, a_n, b_1, \ldots, b_m, c\} \subseteq \mathcal{X}$. An expression like *not b* is called a *default negation*. The intuitive meaning of such a rule is: "if you have all the a_i's and it may be assumed that you have no b_j's then you can conclude c." For such a rule r we use the following notations (extended to a rule set as usual):

- the positive prerequisites of $r : body^+(r) = \{a_1, \ \ldots, \ a_n\}$,
- the negative prerequisites of $r : body^-(r) = \{b_1, \ \ldots, \ b_m\}$,
- the conclusion of $r : head(r) = c$,
- the positive projection of $r : (r)^+ = head(r) \leftarrow body^+(r)$.

If a program P does not contain any default negation (i.e.: $body^+(P) = \emptyset$), then P is a *definite logic program* and it has one minimal Herbrand model denoted $Cn(P)$ (see [28]). The *reduct* P^X of a program P w.r.t. an atom set X is the definite logic program defined by:

$$P^X = \{r^+ \mid r \in P, \ body^-(r) \cap X = \emptyset\}$$

and it is the core of the definition of a *stable model*.

Definition 1 [22] Let P be a normal logic program and S an atom set. S is a stable model of P if and only if $S = Cn(P^S)$.

Note that a program may have one or many stable models or not at all. In this last case we say that the program is *inconsistent*, otherwise it is *consistent*. When an atom belongs at least to one stable model of P it is called a *credulous* consequence of P and when it belongs to every stable model of P, it is called a *skeptical* consequence of P. This vocabulary comes from the one of default logic [41] and it is natural since as it is illustrated in figure 1 and established in [7, 23], stable model semantics can be seen as a subcase of default logic.

Let A be an atom set, r be a rule and P be a program (definite or normal). For the clarity of the sequel we need to introduce the following additional materials.

- r is *applicable* in A if $\text{body}^+(r) \subseteq A$.
- $App(P, A)$ is the subset of P of its applicable rules in A.
- A satisfies r (or r is satisfied by A), denoted by $A \models r$, if when r is applicable in A, then $head(r) \in A$.
- A is *closed* under P if $\forall r \in P, A \models r$.

[1] In the sequel, if $\mathcal{X}$ is not explicitly given, it is supposed to be the set of all atoms occurring in the considered program.

- P is *grounded*[2] if it can be ordered as a sequence $\langle r_1, \ldots, r_n \rangle$ such that
$$\forall i, 1 \leq i \leq n, r_i \in App(P, \text{head}(\{r_1, \ldots, r_{i-1}\})$$
- r is *blocked* by A if $\text{body}^-(r) \cap A \neq \emptyset$.
- $Cn(P)$, the least (Herbrand) model of a definite logic program P, is the smallest atom set closed under P and it can be computed as the least fix-point of the following consequence operator $T_P : 2^{\mathcal{X}} \rightarrow 2^{\mathcal{X}}$ such that $T_P(A) = head(App(P, A))$.

By the next result we can clarify the links between the least model A of a program P and the rules producing it. We see that A is underpinned by a set of applicable rules, $(App(P,A))$, that satisfies a stability condition and that is grounded. These two features will be used in the sequel to define the core of our work: a possibility distribution over atom sets induced by a definite logic program.

Proposition 1 *Let P be a definite logic program and A be an atom set,*

$$A \text{ is the least Herbrand model of } P \iff \begin{cases} A = head(App(P, A)) \\ App(P, A) \text{ is grounded} \end{cases}$$

Let us end this short presentation of stable model semantics by illustrating some notions on our introductory example.

Example 1 $S_1 = \{di1, di2, dr1, c1\}$ is a stable model of the normal logic program

$$P_{med} = \left\{ \begin{array}{ll} dr1 \leftarrow di1, not\ dr2. & dr2 \leftarrow di2, not\ dr1. \\ c1 \leftarrow dr1, di1. & c2 \leftarrow dr2, di2. \\ di1 \leftarrow . & di2 \leftarrow . \end{array} \right\}$$

because

$$P_{med}^{S_1} = \left\{ \begin{array}{ll} dr1 \leftarrow di1. & \\ c1 \leftarrow dr1, di1. & c2 \leftarrow dr2, di2. \\ di1 \leftarrow . & di2 \leftarrow . \end{array} \right\}$$

and

$$\begin{aligned} T_{P_{med}^{S_1}}(\emptyset) &= \{di1, di2\} \\ T_{P_{med}^{S_1}}(\{di1, di2\}) &= \{di1, di2, dr1\} \\ T_{P_{med}^{S_1}}(\{di1, di2, dr1\}) &= \{di1, di2, dr1, c1\} \\ T_{P_{med}^{S_1}}(\{di1, di2, dr1, c1\}) &= \{di1, di2, dr1, c1\} \text{ is the fix-point.} \end{aligned}$$

So, we have

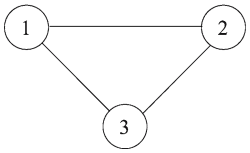
$$Cn(P_{med}^{S_1}) = \{di1, di2, dr1, c1\} = S_1$$

proving that S_1 is a stable model of P_{med}. The same stands for the second stable model $S_2 = \{di1, di2, dr2, c2\}$ and no other set satisfies the fix-point property.

[2]This notion has been firstly introduced in [43] for default logic.

Since one aim of ASP is to solve some combinatorial problems it is necessary to have a compact first order representation language. Thus, even if all our presentation is made in the propositional case, it has to be considered as being the same for a program with variables. Such a program with variables is considered as a shorter representation of the propositional program that is obtained by replacing every variable with every constant in the language. Nevertheless, we are only concerned by a finite non empty set of constants. The following example is given to illustrate this point where a combinatorial problem is encoded by means of three kinds of rules: data rules expressing the particular datas of the given problem, guess rules expressing the different potential solutions (the search space) and constraint rules expressing the constraints that have to be satisfied by a solution.

Example 2



$$P_{color} = \left\{ \begin{array}{rl} \text{data rules:} & v(1) \leftarrow . \quad v(2) \leftarrow . \quad v(3) \leftarrow . \\ & e(1,2) \leftarrow . \quad e(2,3) \leftarrow . \quad e(3,1) \leftarrow . \\ \text{guess rules:} & red(X) \leftarrow v(X), not\ green(X). \\ & green(X) \leftarrow v(X), not\ red(X). \\ \text{check rules:} & bug \leftarrow e(X,Y), red(X), red(Y), not\ bug. \\ & bug \leftarrow e(X,Y), green(X), green(Y), not\ bug. \end{array} \right\}$$

The program P_{color} encodes a 2-coloring problem on the undirected graph with three vertices and three edges given above. The six first rules encodes the data. The two next rules encode the two possible colors (red or green) for every vertex. The two last rules forbid to color two vertices with the same color if they are linked by an edge. Another way to represent this constraint is to used two headless rules like $\leftarrow e(X, Y), green(X), green(Y)$. As it can be checked with the help of the above figure, P_{color} has no stable model at all.

2.2 Possibilistic logic

Possibilistic logic, in the necessity-valued case, handles pairs of the form (p, α) where p is a classical logic formula and α is an element of a totally ordered set. In the sequel we use a finite subset of the interval $[0, 1]$ handled in a qualitative way. That is, numerical values are only required to define an ordered scale of values. The pair (p, α) expresses that the formula p is certain at least to the level α, or more formally by $N(p) \geq \alpha$, where N is the necessity measure associated to the possibility distribution expressing the underlying semantics. Hence, a possibilistic knowledge base is a finite set of weighted formulas $\Sigma = \{(p_i, \alpha_i), i = 1, \ldots, n\}$ where α_i is understood as a lower bound of the necessity degree $N(p_i)$. Formulas with zero degree are not explicitly represented in the knowledge base (only beliefs which are somewhat accepted are explicitly represented). The higher is the weight, the more certain is the formula. This degree α is evaluated by a necessity measure and it is not a probability. Thus, numerical values are not an absolute evaluation (like it is in probability theory) but induce a certainty (or confidence) scale. Moreover, let us note that these values are determined by the expert providing the knowledge base or they are automatically given if, as we can imagine, the rules and their confidence degrees result from a knowledge discovery process.

Given a possibilistic knowledge base Σ we denote Σ^* its classical part obtained by forgetting the weights. The basic element of possibility theory is the possibility

distribution π which is a mapping from Ω, the classical interpretation set of Σ^*, to the interval [0, 1]. $\pi(\omega)$ represents the degree of compatibility of the interpretation ω with the 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 from being the real world, it is a model of Σ^*. When $\pi(\omega) > \pi(\omega')$, ω is a preferred candidate to ω' for being the real state of the world.

A possibility distribution π is said to be normal if $\exists\omega \in \Omega$, such that $\pi(\omega) = 1$, namely there exists at least one interpretation which is consistent with all the available beliefs.

Given a possibility distribution π, we can define two different ways of rank-ordering formulas of the language from this possibility distribution. This is obtained using two mappings grading respectively the possibility and the certainty of a formula p:

– $\Pi(p) = max\{\pi(\omega) \mid \omega \models p\}$ is the possibility (or consistency) degree which evaluates the extent to which p is consistent with the available beliefs expressed by π [49]. It satisfies:

$$\forall p, \forall q, \Pi(p \vee q) = max(\Pi(p), \Pi(q))$$

– $N(p) = 1 - \Pi(\neg p)$ is the necessity (or certainty, entailment) degree which evaluates the extent to which p is entailed by the available beliefs. We have [19]:

$$\forall p, \forall q, N(p \wedge q) = min(N(p), N(q))$$

A possibility distribution is said to be *compatible* with a possibilistic base $\Sigma = \{(p_i, \alpha_i), i = 1, \ldots, n\}$ if, for all $(p_i, \alpha_i) \in \Sigma$, $N(p_i) \geq \alpha_i$ that is $1 - \max_{\omega\in\Omega}\{\pi(\omega) | \omega \not\models p_i\} \geq \alpha_i$. Generally, there are several possibility distributions compatible with Σ. A way to select one particular distribution is to use the minimum specificity principle [48]. It consists in putting to each interpretation the highest possibility degree compatible with the formulas. Intuitively, it corresponds to the least informative possibility distribution.

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 possibility distribution always exists and is characterized in [18] by:

$$\begin{array}{l}\text{if } \omega \text{ is a model of } \Sigma^*, \text{ then } \pi_\Sigma(\omega) = 1\\ \text{else } \pi_\Sigma(\omega) = 1 - max\{\alpha_i \mid \omega \not\models p_i, (p_i, \alpha_i) \in \Sigma\}\end{array}$$

Example 3 Let $\Sigma = \{(a, 0.9), (b, 0.6), (a \wedge b \Rightarrow c, 0.8)$ be a possibilistic base. Its least specific possibility distribution is summarized in the table below.

ω	$\neg a \neg b \neg c$	$\neg a b \neg c$	$\neg a \neg b c$	$\neg a b c$
π_Σ	0.1	0.1	0.1	0.1
ω	$a \neg b \neg c$	$a b \neg c$	$a \neg b c$	$a b c$
π_Σ	0.4	0.2	0.4	1

We can see that all interpretations on the first line have a low possibility degree of 0.1 because they all contain $\neg a$ and then contradict the formula in Σ with highest certainty (here 0.9). On the other side, the model $\{a, b, c\}$ of Σ^* is fully possible.

3 Possibilistic definite logic programs

In this section we deal with definite logic programs. First, we introduce the framework in which uncertainty is taken into account using possibility theory: the *possibilistic definite logic programs*. Like in the possibilistic logic framework, we study the behavior of such programs both in the semantical and in the syntactical ways. The semantical part concerns the treatment of programs by the way of a possibility distribution defined on the sets of atoms. The syntactical part deals with a computation on the rules themselves. The crucial point is that the two ways of treatment are equivalent and lead to the same results.

3.1 Language

We introduce the formal notions useful to merge the possibilistic logic treatment of uncertainty with the knowledge representation framework given by a definite logic program.

Definition 2 Let $\mathcal{X}$ be a finite set of atoms and $\mathcal{N} \subseteq]0, 1]$ a finite, totally ordered set of necessity values. A *possibilistic atom* is a pair $p = (x,\ \alpha) \in \mathcal{X} \times \mathcal{N}$ and we denote

- $p^* = x$ the classical projection of p,
- $n(p) = \alpha$ the necessity degree of p.

A *possibilistic atom set* is a functional relation from $\mathcal{X}$ to $\mathcal{N}$. We denote $\mathcal{A}$ the set of all possibilistic atom sets.

In other words a possibilistic atom set $A \in \mathcal{A}$ is a set of possibilistic atoms where every atom x occurs at most one time in A and always with a strictly positive certainty degree, ie: $\forall x \in \mathcal{X}, |\{(x,\ \alpha) \in A\}| \leq 1$.

Definition 3 A *possibilistic definite logic program* is a set of *possibilistic rules* of the form:

$$r = (c \leftarrow a_1,\ \ldots,\ a_n.\ ,\ \alpha) \text{ with } n \geq 0, \{a_1,\ \ldots,\ a_n,\ c\} \subseteq \mathcal{X}, \alpha \in \mathcal{N}$$

We denote

- $r^* = c \leftarrow a_1,\ \ldots,\ a_n.$ the *classical projection* of the possibilistic rule,
- $n(r) = \alpha$ the necessity degree representing the certainty level of the information described by the rule.

As in possibilistic logic, the degree α is evaluated by a necessity measure and it is not a probability. Thus, numerical values are not an absolute evaluation (like it is done in probability theory) but induce a certainty (or confidence) scale allowing to order the rules. If R is a set of possibilistic rules, then $R^* = \{r^* \mid r \in R\}$ is the definite logic program obtained from R by forgetting all the necessity values. For a given

possibilistic definite logic program P and an atom x, we define $H(P, x) = \{r \in P \mid head(r^*) = x\}$ the set that collects all rules in P having the same head x.

Let us recall that a possibilistic logic base is a compact representation of the possibility distribution defined on interpretations representing the information. Indeed, the treatment of the base in a syntactical way (in terms of formulas and necessity degrees) leads to the same results as the treatment done in a semantical way (in terms of interpretations and possibility distribution). In our framework, the same situation occurs as it will be shown in the next two subsections. Firstly, we define a semantical management of a program that is defined in term of a possibility distribution over all atom sets. Secondly, we provide a syntactical deduction process based on a fix-point operator defined on rules and that leads to the same results as the ones given by the possibility distribution.

3.2 Model theory for possibilistic definite logic programs

From a possibilistic definite logic program P, we can determine, as it is done in possibilistic logic, some possibility distributions defined on all the sets in $2^{\mathcal{X}}$ and that are compatible with P. Like in possibilistic logic, the possibility degree of an atom set is determined by the necessity degrees of the rules of the program that are not satisfied by this set. Knowing the framework of definite logic programs, the reader can see that the satisfiability of a rule r is based on its applicability w.r.t. an atom set A and then $A \not\models r$ iff $\text{body}^+(r) \subseteq A \wedge head(r) \notin A$ (see section 2). But, we have to notice that the contradiction of a rule is not enough to determine the possibility degree of a set since, in ASP, it is important to take into account the notions of groundedness and of stability (see proposition 1). Firstly, the set $A = \{a, b\}$ satisfies every rule in $R = \{a \leftarrow b.,\ b \leftarrow a.\}$, but it is not a model of R because the groundedness is not satisfied. Secondly, the set $A' = \{a, b, d\}$ satisfies every rule in $R' = \{a \leftarrow ., b \leftarrow a., d \leftarrow c.\}$ but it is not a model of R' because d can not be produced by any rule from R' applicable in A'. In these two cases, the possibility of A and A' must be 0 since they cannot be a model at all, even if they satisfy every rule in their respective associated program.

Definition 4 Let P be a possibilistic definite logic program and $\pi : 2^{\mathcal{X}} \rightarrow [0, 1]$ be a possibility distribution. π is compatible with P if

$$\forall A \in 2^{\mathcal{X}} \begin{cases} A \not\subseteq head(App(P^*, A)) \Rightarrow \pi(A) = 0 \\ App(P^*, A) \text{ is not grounded } \Rightarrow \pi(A) = 0 \\ A \text{ is a model of } P^* \Rightarrow \pi(A) = 1 \\ \text{otherwise } \pi(A) \leq 1 - \max_{r \in P}\{n(r) \mid A \not\models r^*\} \end{cases}$$

The necessity degree attached to each rule defines only a lower bound (and not an exact value) of the certainty of the rule. So, as recalled in section 2, many possibility distributions are compatible with these degrees. But, we are only interested by the least informative one, that is the *least specific* one, whose characterization is given below.

Proposition 2 *Let P be a possibilistic definite logic program then $\pi_P : 2^{\mathcal{X}} \rightarrow [0, 1]$ defined by*

$$\forall A \in 2^{\mathcal{X}} \begin{cases} A \not\subseteq head(App(P^*, A)) \Rightarrow \pi_P(A) = 0 \\ App(P^*, A) \text{ is not grounded } \Rightarrow \pi_P(A) = 0 \\ A \text{ is a model of } P^* \Rightarrow \pi_P(A) = 1 \\ \text{otherwise } \pi_P(A) = 1 - \max_{r \in P}\{n(r) \mid A \not\models r^*\} \end{cases}$$

is the least specific possibility distribution.

The definition of π_P ensures that it is compatible and no least specific possibility distribution is compatible. The fourth case ranks the sets which may be solutions with respect to the weights of the rules they falsify.

The following proposition demonstrates the links between the least specific compatible possibility distribution and the least model of the classical projection of the program. In particular, only the model of P^* has a possibility equal to 1.

Proposition 3 *Let P be a possibilistic definite logic program and $A \subseteq \mathcal{X}$ be an atom set, then*

1. $\pi_P(A) = 1 \iff A = Cn(P^*)$
2. $A \supset Cn(P^*) \Rightarrow \pi_P(A) = 0$
3. $Cn(P^*) \neq \emptyset \Rightarrow \pi_P(\emptyset) = 1 - \max_{r \in P}\{n(r) \mid body^+(r^*) = \emptyset\}$

Now, we can give the definition of inference that is, in the framework of ASP, the evaluation of the necessity degree of each atom of the universe.

Definition 5 Let P be a possibilistic definite logic program and π_P the least specific possibility distribution compatible with P, we define the two dual possibility and necessity measures such that:

- $\Pi_P(x) = \max_{A \in 2^{\mathcal{X}}}\{\pi_P(A) \mid x \in A\}$
- $N_P(x) = 1 - \max_{A \in 2^{\mathcal{X}}}\{\pi_P(A) \mid x \notin A\}$

$\Pi_P(x)$ gives the level of consistency of x with respect to the possibilistic definite logic program P and $N_P(x)$ evaluates the level at which x is inferred from P. This is closely related to the definitions of possibilistic logic. For instance, whenever an atom x belongs to the model of the classical program its possibility is total (equal to 1). Some other results are given below.

Proposition 4 *Let P be a possibilistic definite logic program and $Cn(P^*)$ the least model of P^*. $\forall x \in \mathcal{X}$ we have:*

1. $x \in Cn(P^*) \Rightarrow \Pi_P(x) = 1$ *and* $x \notin Cn(P^*) \Rightarrow \Pi_P(x) = 0$
2. $x \notin Cn(P^*) \iff N_P(x) = 0$
3. $x \in Cn(P^*) \Rightarrow N_P(x) = \min_{A \subset Cn(P^*)}\{\max_{r \in P}\{n(r) \mid A \not\models r^*\} \mid x \notin A\}$
4. *Let P' be a possibilistic definite logic program, $P \subseteq P' \Rightarrow N_P(x) \leq N_{P'}(x)$.*

The last point illustrates that certainty of every piece of knowledge monotonically grows when new information is added in the program. Furthermore, the necessity measure allows us to introduce the following definition of a *possibilistic model* of a possibilistic definite logic program.

Definition 6 Let P be a possibilistic definite logic program, then the set

$$\Pi M(P) = \{(x, N_P(x)) \mid x \in \mathcal{X}, N_P(x) > 0\}$$

is its *possibilistic model*.

Proposition 5 *Let P be a possibilistic definite logic program then: $(\Pi M(P))^*$ is the least model of P^*.*

Example 4 Let us take $\mathcal{X} = \{a, b, c\}$ and the following program P.

$$P = \{(a \leftarrow ., 0.9), (b \leftarrow ., 0.6), (c \leftarrow a, b., 0.8)\}$$

The least specific possibility distribution on $2^{\mathcal{X}}$ that is induced by P is the following:

$$\begin{array}{ll} \pi_P(\emptyset) = 1 - \max\{0.9, 0.6\} = 0.1 & \pi_P(\{a\}) = 1 - \max\{0.6\} = 0.4 \\ \pi_P(\{b\}) = 1 - \max\{0.9\} = 0.1 & \pi_P(\{a, b\}) = 1 - \max\{0.8\} = 0.2 \\ \pi_P(\{c\}) = 0 \text{ (no inclusion)} & \pi_P(\{a, c\}) = 0 \text{ (no inclusion)} \\ \pi_P(\{b, c\}) = 0 \text{ (no inclusion)} & \pi_P(\{a, b, c\}) = 1 \text{ (\textit{the model})} \end{array}$$

The reader has to note that program P in example 4 encodes the same kind of information as the base Σ in example 3. The different atom sets just above can be mapped into the interpretations given in example 3 (each column corresponds to a row). But, there is some differences due to the particular semantics of logic programs. For instance, the set $\{c\}$ is absolutely impossible, whereas the corresponding interpretation valuating c with true and a and b with false has a possibility of 0.1 in the propositional case. Again, we can point out some differences between $P = \{(a \leftarrow b., \alpha), (b \leftarrow a., \alpha')\}$ and $\Sigma = \{(b \Rightarrow a, \alpha), (a \Rightarrow b, \alpha')\}$ since we have $\pi_P(\emptyset) = \pi_\Sigma(\{\neg a, \neg b\}) = 1$, but $\pi_P(\{a, b\}) = 0$ and $\pi_\Sigma(\{a, b\}) = 1$. This illustrates the particular semantic of a logic program that is only concerned by minimal models and requires that every conclusion is supported by a chain of applied rules. It justifies our definition of possibility distribution that had to take into account these peculiarities.

3.3 Fix-point theory for possibilistic definite logic programs

Definitions exposed in this subsection are closely related to what can be found in [17]. But here, we adopt an ASP point of view and thus we use atom sets instead of classical interpretations since the underlying possibility distribution is defined on atom sets.

Definition 7 Consider $\mathcal{A}$ the finite set of all possibilistic atom set induced by $\mathcal{X}$ and $\mathcal{N}$ (see definition 2). $\forall A, B \in \mathcal{A}$, we define:

$$\begin{array}{rcl} A \sqcap B & = & \{(x, \min\{\alpha, \beta\}), (x, \alpha) \in A, (x, \beta) \in B\} \\ A \sqcup B & = & \{(x, \alpha) \mid (x, \alpha) \in A, x \notin B^*\} \cup \{(x, \beta) \mid x \notin A^*, (x, \beta) \in B\} \\ & & \cup \{(x, \max\{\alpha, \beta\}) \mid (x, \alpha) \in A, (x, \beta) \in B\} \end{array}$$

$$A \sqsubseteq B \iff \begin{cases} A^* \subseteq B^*, \text{ and} \\ \forall a, \alpha, \beta, (a, \alpha) \in A \wedge (a, \beta) \in B \Rightarrow \alpha \leq \beta \end{cases}$$

Proposition 6 *By definition 7, we have*

- $\forall A, B \in \mathcal{A}$, $A \sqcap B$ *is the greatest lower bound of* $\{A, B\}$,
- $\forall A, B \in \mathcal{A}$, $A \sqcup B$ *is the lowest upper bound of* $\{A, B\}$,
- $\bot = \emptyset$ *is the minimal element of* $\mathcal{A}$
- $\top = \{(x, \max_{\alpha \in \mathcal{N}}\{\alpha\}) \mid x \in \mathcal{X}\}$ *is the maximal element of* $\mathcal{A}$,

and then $\langle \mathcal{A}, \sqsubseteq \rangle$ *is a complete lattice.*

Definition 8 Let $r = (c \leftarrow a_1, \ \ldots, \ a_n., \alpha)$ be a possibilistic rule and A be a possibilistic atom set,

- r is β-applicable in A with $\beta = \min\{\alpha, \alpha_1, \ldots, \alpha_n\}$ if $\{(a_1, \alpha_1), \ldots, (a_n, \alpha_n)\} \subseteq A$, [3]
- r is 0-applicable otherwise.

And then, for all atom x we define:

$$App(P, A, x) = \{r \in H(P, x), r \text{ is } \beta\text{-applicable in } A, \beta > 0\}$$

Definition 9 Let P be a possibilistic definite logic program and A be a possibilistic atom set. The *immediate possibilistic consequence operator* ΠT_P maps a possibilistic atom set A to another one by this way:

$$\Pi T_P(A) = \left\{ (x, \delta) \;\middle|\; \begin{array}{l} x \in head(P^*), App(P, A, x) \neq \emptyset, \\ \delta = \max\limits_{r \in App(P, A, x)} \{\beta \mid r \text{ is } \beta\text{-applicable in } A\} \end{array} \right\}$$

Then the iterated operator ΠT_P^k is defined by

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

Given a possibilistic atom set A, the applicability degree β of a rule β-applicable in A captures the certainty of the conclusion that the rule can produce w.r.t. A. If the body is empty, then the rule is applicable with its own certainty degree. If the body is not verified (not satisfied by A), then the rule is not at all applicable. Otherwise, the applicability level of the rule depends on the certainty level of the propositions inducing the groundedness of the rule and its own certainty degree. As in possibilistic logic, the certainty of a conjunction (the body of the rule) is the minimal value of the necessity values of subformulae (atoms) involved in it. Again, the certainty of a

[3] For two possibilistic atom sets A and B, $A \subseteq B$ means the classical set inclusion and has not to be confused with the order $\sqsubseteq$ introduced in definition 7.

conclusion is the minimal value between the rule certainty and the certainty degree of its body. This is similar to the Generalized Modus Ponens used in possibilistic logic: $\frac{(a, \alpha)\quad (a \Rightarrow b, \beta)}{(b, \min\{\alpha, \beta\})}$. Furthermore, let us remark the following feature of our operator. If one conclusion is obtained by different rules, its certainty is equal to the greatest certainty under which it is obtained in each case (operator max).

Proposition 7 *ΠT_P is monotonic, i.e.: $A \sqsubseteq B \Rightarrow \Pi T_P(A) \sqsubseteq \Pi T_P(B)$.*

Proposition 8 *Let P be a possibilistic definite logic program, then ΠT_P has a least fixpoint $\bigsqcup_{n\geq 0} \Pi T_P^n$ that we call the set of* possibilistic consequences *of P and we denote it by $\Pi Cn(P)$.*

Example 5 Let P be the possibilistic definite logic program given in example 4.

$$\Pi Cn(P) = \{(a, 0.9), (b, 0.6), (c, 0.6)\}$$

since

$$\begin{aligned}
\Pi T_P^0 &= \emptyset \\
\Pi T_P^1 &= \Pi T_P(\emptyset) = \{(a, 0.9), (b, 0.6)\} \\
\Pi T_P^2 &= \Pi T_P(\{(a, 0.9), (b, 0.6)\}) = \{(a, 0.9), (b, 0.6), (c, 0.6)\} \\
\Pi T_P^3 &= \Pi T_P(\{(a, 0.9), (b, 0.6), (c, 0.6)\}) = \{(a, 0.9), (b, 0.6), (c, 0.6)\} \\
\Pi T_P^{k+1} &= \Pi T_P^k,\ \forall k > 2
\end{aligned}$$

Proposition 9 *Let P be a possibilistic definite logic program. The computation of $\Pi Cn(P)$ can be done in polynomial time w.r.t. $k\times \mid P \mid$ where k is the number of different levels of certainty occurring in P.*

As illustrated in the previous example and formalized in the next result, our operator ΠT_P can be used to compute the possibilistic model of a possibilistic definite logic program. This result shows the equivalence between the syntactic and semantic approach of our framework.

Theorem 1 *Let P be a possibilistic definite logic program, then $\Pi Cn(P) = \Pi M(P)$.*

We have shown the equivalence of the semantical and the syntactical ways for dealing with definite logic programs. It is now interesting to focus on logic programs with a larger extent of representation, particularly by allowing the use of default negations. This is the issue of the next section.

4 Possibilistic normal logic programs

If we are not interested in inconsistent models, an *extended logic program* (i.e.: a program built with *literals* and not only atoms but without head *disjunction*) under answer set semantics is reducible to a program (without strong negation) under stable

model semantics (see [23]). That is why we present our work by using the stable model paradigm and deal with *general logic programs* [3] also called *normal logic programs*.

4.1 Language and possibilistic stable models

Here, we want to formalize the notion of *possibilistic stable model* that extends the stable model semantics by taking into account the necessity degree in the rules of a given *possibilistic normal logic program*. As in definition 3, such a program is a finite set of rules of the form:

$$(c \leftarrow a_1,\ \ldots,\ a_n,\ not\ b_1,\ \ldots,\ not\ b_m.\ ,\ \alpha)\ n \geq 0, m \geq 0$$

for which we just have to precise that $\forall i, b_i \in \mathcal{X}$, all the rest being the same as for a possibilistic definite logic program (see subsection 3.1).

As in the classical case without necessity value, we need to define what is the reduction of a program.

Definition 10 Let P be a possibilistic normal logic program and A be an atom set. The *possibilistic reduct* of P w.r.t. A is the possibilistic definite logic program

$$P^A = \{(r^{*^+}, n(r)) \mid r \in P, r \text{ is not blocked by } A\}$$

By this way, the definition of a possibilistic stable model becomes natural.

Definition 11 Let P be a possibilistic normal logic program and S a possibilistic atom set. S is a possibilistic stable model of P if and only if $S = \Pi Cn(P^{(S^*)})$.

By analogy with classical normal logic programs (without necessity values attached to rules) we say that a possibilistic normal logic program P is *consistent* if P has at least one possibilistic stable model. Otherwise P is said to be *inconsistent*. Furthermore when a possibilistic atom belongs[4] at least to one possibilistic stable model of P it is called a *credulous* possibilistic consequence of P and when it belongs to the intersection of all stable models of P, it is called a *skeptical* possibilistic consequence of P.

Example 6 We are now able to represent our introductory program detailed in example 1. By using the following certainty scale : 1 is *absolutely certain*, 0.9 is *quasi certain*, 0.7 is *almost certain* and 0.3 is *little certain*, we can add to each information a certainty level as in the following possibilistic normal logic program.

$$P'_{med} = \left\{ \begin{array}{ll} (dr1 \leftarrow di1, not\ dr2., 1) & (dr2 \leftarrow di2, not\ dr1., 1) \\ (c1 \leftarrow dr1, di1., 0.7) & (c2 \leftarrow dr2, di2., 0.3) \\ (di1 \leftarrow ., 0.9) & (di2 \leftarrow ., 0.7) \end{array} \right\}$$

The two first rules (appropriateness and incompatibility of the drugs) are considered absolutely sure. The third (resp. fourth) rule expresses that we are almost (resp.

[4] $\forall A \in \mathcal{A}, \forall x \in \mathcal{X}, \forall \alpha \in \mathcal{N}, (x, \alpha)$ belongs to $A \iff \{(x, \alpha)\} \sqsubseteq A$.

little) certain of the efficiency of the drug 1 (resp. drug 2). The two last rules indicate that the diagnosis of disease 1 (resp. disease 2) is quasi (resp. almost) certain. P'_{med} has two possibilistic stable models:

$$S_1 = \{(di1, 0.9), (di2, 0.7), (dr1, 0.9), (c1, 0.7)\} \quad \text{and}$$
$$S_2 = \{(di1, 0.9), (di2, 0.7), (dr2, 0.7), (c2, 0.3)\}.$$

So, the doctor can observe that he has an alternative. On one hand, he can quasi certainly give the drug $dr1$ and be almost certain that the patient will be cured of the disease $di1$. On the other hand, he can give almost certainly the drug $dr2$ and the patient will be cured of the disease $di2$ but that is only little certain. However, if the doctor considers that disease $di2$ is very serious, maybe he will choose the drug $dr2$ even if the degree is lower. That is why it is interesting to obtain and keep the two stable models in which every conclusion is weighted with a certainty degree.

Proposition 10 *Let P be a possibilistic normal logic program*

1. *Let A be a possibilistic stable model of P and $\alpha \in \mathcal{N}$, then $(x, \alpha) \in A \iff \alpha = N_{P^{A^*}}(x)$.*
2. *Let A be a stable model of P^*, then $\{(x, N_{P^A}(x)) \mid x \in \mathcal{X}, N_{P^A}(x) > 0\}$ is a possibilistic stable model of P.*
3. *Let A be a possibilistic stable model of P, then A^* is a stable model of P^*.*

Theorem 2 *The problem of deciding whether a possibilistic normal logic program has a possibilistic stable model is NP−complete.*

Proposition 10 shows that there is a one-to-one mapping between the possibilistic stable models of a possibilistic normal logic program P and the stable models of its classical part P^*. It leads to theorem 2 showing that the decision problem of existence of a possibilistic stable model for a possibilistic normal logic program stays in the same complexity class as the decision problem of existence of a stable model for a normal logic program. Furthermore, it indicates an easy way to implement a system able to compute a possibilistic stable model of a possibilistic normal logic program P. First, one can use an available software to compute the stable models of P^*. Second, for every found stable model A, our operator ΠT_{P^A} can be used to compute (in polynomial time) the corresponding possibilistic stable model $\Pi Cn(P^A)$ of P. That is the methodology that we have adopted to develop our system posSmodels described in section 6.

4.2 Possibility distribution

In the previous definition 11 we have proposed a syntactical way to compute the possibilistic stable models of a possibilistic normal logic program by using the fix-point operator ΠCn defined for a possibilistic definite logic program. Now, we examine the semantics that can be given to this framework by defining a possibility distribution induced by the necessity values associated to normal rules. This distribution, denoted $\tilde{\pi}$, over $2^{\mathcal{X}}$ has to reflect the ability of every atom set to be a stable model of P^*.

Definition 12 Let P be a possibilistic normal logic program and A be an atom set, then $\tilde{\pi}_P$ is the possibility distribution defined by $\tilde{\pi}_P : 2^{\mathcal{X}} \longmapsto [0, 1]$ and respecting:

$$\forall A \in 2^{\mathcal{X}}, \tilde{\pi}_P(A) = \pi_{P^A}(A)$$

The reader can observe that definition of $\tilde{\pi}_P$ can be paraphrased by: "the possibility for an atom set A to be a stable model of P is its possibility to be a model (an Herbrand one) of the program P reduced by A." This definition is natural w.r.t. the definition 1 of a stable model and the next result justifies it formally.

Proposition 11 *Let P be a possibilistic normal logic program and $A \in 2^{\mathcal{X}}$ be an atom set, then*

$$\tilde{\pi}_P(A) = 1 \iff A \text{ is a stable model of } P^*$$

Example 7 (example 6 continued) The possibility distribution associated to the program P'_{med} is

$$\begin{array}{rr}
\tilde{\pi}_{P'_{med}}(\emptyset) = 0.1 & \\
\tilde{\pi}_{P'_{med}}(\{di1, dr1\}) = 0.3 & \tilde{\pi}_{P'_{med}}(\{di2, dr2\}) = 0.1 \\
\tilde{\pi}_{P'_{med}}(\{di1, dr1, c1\}) = 0.3 & \tilde{\pi}_{P'_{med}}(\{di2, dr2, c2\}) = 0.1 \\
\tilde{\pi}_{P'_{med}}(\{di1, di2, dr1\}) = 0.3 & \tilde{\pi}_{P'_{med}}(\{di1, di2, dr2\}) = 0.7 \\
\tilde{\pi}_{P'_{med}}(\{di1, di2, dr1, c1\}) = 1 & \tilde{\pi}_{P'_{med}}(\{di1, di2, dr2, c2\}) = 1 \\
\multicolumn{2}{c}{\text{for all other sets } S, \tilde{\pi}_{P'_{med}}(S) = 0}
\end{array}$$

We see that the two atom sets corresponding to the two possibilistic stable models are totally possible.

We define now the two possibility measures of possibility and necessity.

Definition 13 Let P be a possibilistic normal logic program and $\tilde{\pi}_P$ its associated possibility distribution, we define the two dual possibility and necessity measures such that:

- $\tilde{\Pi}_P(x) = \max_{A \in 2^{\mathcal{X}}} \{\tilde{\pi}_P(A) \mid x \in A\}$
- $\tilde{N}_P(x) = 1 - \max_{A \in 2^{\mathcal{X}}} \{\tilde{\pi}_P(A) \mid x \notin A\}$

Proposition 12 *Let P be a consistent possibilistic normal logic program, $\forall x \in \mathcal{X}$ we have:*

1. *x is a credulous consequence of $P^* \iff \tilde{\Pi}_P(x) = 1$*
2. *x is not a skeptical consequence of $P^* \iff \tilde{N}_P(x) = 0$*
   (or in other words : x is a skeptical consequence of $P^* \iff \tilde{N}_P(x) > 0$)

Let us remark that if an atom x is not a credulous consequence of P then it does not necessarily implies that $\tilde{\Pi}_P(x) = 0$ as it is the case for an atom that is not a consequence of a possibilistic definite logic program (see proposition 4 item 1). For instance, $P = \{(a \leftarrow ., 0.6), (b \leftarrow not\ a., 0.7)\}$ has only one possibilistic stable model $\{(a, 0.6)\}$ so b is not a possibilistic credulous consequence of P. But, the

possibility distribution is $\tilde{\pi}_P(\emptyset) = 0.3$, $\tilde{\pi}_P(\{a\}) = 1$, $\tilde{\pi}_P(\{b\}) = 0.4$ and $\tilde{\pi}_P(\{a, b\}) = 0$ so $\tilde{\Pi}_P(b) = 0.4$. This is because, with this program, b is not completely impossible. In fact, b is not a credulous consequence only because a (that is a credulous consequence) blocks the applicability of the rule concluding b. So, in other words if we have a then we cannot have b, but if a is absent then we can have b. Since, the certainty of a is only 0.6, so the possibility of b is naturally 0.4.

As stated at the beginning of this section, strong negations can be encoded in a normal logic program with new atoms. The following example shows how to deal with strong negations in our framework where certainty degrees are given.

Example 8 The possibilistic extended logic program

$$P = \{(a \leftarrow ., 0.8), (\neg b \leftarrow a, not\ c., 0.6), (c \leftarrow a, not\ \neg b., 0.9)\}$$

can be translated into a possibilistic normal logic program

$$P' = \{(a \leftarrow ., 0.8), (b' \leftarrow a, not\ c., 0.6), (c \leftarrow a, not\ b'., 0.9), (bug \leftarrow b, b', not\ bug., 1)\}$$

As it is intuitively expected P' has two possibilistic stable models $\{(a, 0.8), (b', 0.6)\}$ and $\{(a, 0.8), (c, 0.8)\}$, the first one corresponding to $\{(a, 0.8), (\neg b, 0.6)\}$. In P' the reader can see that we have renamed every negative literal $\neg x$ by a new atom x' as it is well known. The last rule $(bug \leftarrow b, b', \text{not } bug., 1)$, where bug is a new atom, forbids b and b' (that is $\neg b$) to belong to the same possibilistic stable model. We have given to this special rule a certainty of 1 in order that the possibility of every atom set containing b and b' is null.

5 Inconsistent possibilistic normal logic programs

One feature of possibilistic logic is its ability to manage inconsistency of a formula set. It proposes a way to restore the consistency of a formula set by deleting some less certain (or preferred) formulas, those with a low certainty degree. We present here an analogous idea in order to deal with inconsistent normal logic programs.

5.1 Formal definitions

A possibilistic base is a set Σ of pairs constituted with a classical formula and a weight that is a necessity degree. Σ is said to be *consistent* (resp. inconsistent) if its classical support Σ^* is classically consistent (resp. inconsistent). It is interesting to note that possibilistic logic addresses the problem of inconsistency by selecting a consistent subbase with respect to the necessity values of the formulas. An α-cut (resp. strict α-cut) of Σ, denoted by $\Sigma_{\geq\alpha}$ (resp. by $\Sigma_{>\alpha}$), is the set of formulas in Σ having a certainty degree greater than (resp. strictly greater than) α. The inconsistency degree of Σ is $Inc(\Sigma) = max\{\alpha \mid \Sigma_{\geq\alpha} \text{is inconsistent}\}$. $Inc(\Sigma) = 0$ means that Σ is consistent. If Σ^* has no model, then, by discarding formulas whose necessity degree is lower than the inconsistency degree, it defines an α-cut $\Sigma_{>Inc(\Sigma)}$ that is consistent. It is clear that this cut may eliminate some formulas that are not involved in the inconsistency. Nevertheless, $Inc(\Sigma)$ defines a plausibility level under which information is no more pertinent. So, it is justified to eliminate all the formulas representing this piece of

knowledge. Let us mention that the inconsistency degree can be computed by means of the least specific possibility distribution of Σ.

Our basic idea is inspired by these general principles coming from possibilistic logic. Considering that every rule in a program has a certainty degree allows us to rank all rules by strata with respect to these degrees. Then, the aim of our consistency restoring process is to keep the greatest number of most preferred strata. The next definition characterizes the goal to reach.

Definition 14 Let P be a possibilistic normal logic program

- the *strict α-cut* of P is the subprogram $P_{>\alpha} = \{r \in P \mid n(r) > \alpha\}$
- the *consistency cut degree* of P is

$$ConsCutDeg(P) = \begin{cases} 0 \text{ if } P \text{ is consistent} \\ \min_{\alpha \in \mathcal{N}}\{\alpha \mid P_{>\alpha} \text{ is consistent}\} \text{ otherwise} \end{cases}$$

The consistency cut degree of a possibilistic normal logic program P defines the minimum level of certainty for which a strict α-cut of P is consistent. When P is inconsistent $P_{>ConsCutDeg(P)}$ is the consistent subprogram of P that we want to compute. Let us note that, because of the non monotonicity of the framework it does not ensure that a higher cut is necessarily consistent. And also, it is not necessarily the greatest (in number of rules) consistent subprogram of P. Here, our approach to restore the consistency of a possibilistic normal logic program is to delete the minimum number of lesser certain rules.

Example 9 Let P be the following possibilistic normal logic program

$$P = \left\{ \begin{array}{l} (c \leftarrow ., 1), (f \leftarrow not\ e, not\ f., 0.9), (e \leftarrow not\ b., 0.8), \\ (a \leftarrow not\ a, not\ b., 0.7), (d \leftarrow c, not\ d., 0.6), (b \leftarrow c., 0.5) \end{array} \right\}$$

Then $ConsCutDeg(P) = 0.7$ since $P_{>0}(= P)$, $P_{>0.5}$ and $P_{>0.6}$ are inconsistent and $P_{>0.7}$ is consistent. Let us remark that $P_{>0.8}$ is inconsistent. This last point illustrates a notable difference between classical logic and stable model semantics. In classical logic, every subset of a consistent set of formulas is itself consistent. But, a subset of a consistent normal logic program is not necessarily consistent and this is due to the non monotonic nature of the formalism.

The previous definition deals with the syntactic aspect of consistency restoring since it is only based on the rules of the program. It is interesting to define an inconsistency degree in a semantic way like it is done in possibilistic logic by using a possibility distribution.

Definition 15 Let P be a possibilistic normal logic program, its inconsistency degree is

$$InconsDeg(P) = 1 - \max_{A \in 2^{\mathcal{X}}}\{\tilde{\pi}_P(A)\}$$

This inconsistency degree can be used to characterize an inconsistent possibilistic normal logic program and to define a cut that is still a superset of the consistent

subprogram that we want to obtain. Unfortunately, conversely to possibilistic logic where the inconsistency degree directly leads to a consistent subbase, it does not necessarily provide us with a consistent program. This is due to the non monotonicity of ASP paradigm.

Proposition 13 *Let P be a possibilistic normal logic program, then*

1. *P is inconsistent $\iff$ $InconsDeg(P) > 0$*
2. *$InconsDeg(P) \leq ConsCutDeg(P)$.*

Our inconsistency degree definition allows us to define a methodology of consistency restoring for a possibilistic normal logic program. This is done by means of the next function *cut* that computes the greatest (w.r.t. the certainty level of rules) consistent subprogram of P.

Definition 16 Let *cut* be the function defined on a possibilistic normal logic program by

$$\begin{cases} cut(P) = P & \text{if } InconsDeg(P) = 0 \\ cut(P) = cut(P_{>InconsDeg(P)}) & \text{otherwise} \end{cases}$$

The next proposition shows that the function *cut* using the notion of inconsistency degree of a program determines its greatest (in terms of scale ordering) consistent subprogram.

Proposition 14 *Let P be a possibilistic normal logic program then $cut(P) = P_{>ConsCutDeg(P)}$.*

Example 10 Let us come back to our program P in example 9. Since $\max_{A \in 2^{\mathcal{X}}} \{\tilde{\pi}_P(A)\} = 0.3$, then we have $InconsDeg(P) = 0.7$. The first call to *cut* is enough to compute the maximal consistent subprogram of P: $cut(P) = \{(c \leftarrow ., 1), (f \leftarrow not\ e, not\ f., 0.9), (e \leftarrow not\ b., 0.8)\}$ such that $cut(P)^*$ has one stable model $\{c, e\}$.

5.2 Relations with possibilistic logic

Here, we focus our attention on possibilistic normal logic program encoding classical possibilistic bases. Let $\mathcal{A}$ be an atom set from which a classical propositional base is built. Recall that every propositional base can be encoded in a clause set S. So, without loss of generality, we consider here only clause sets. On its turn, such a clause set S can be translated in a normal logic program $P(S)$ as follows). First, the translation of a clause $cl = (\neg a_1 \vee \cdots \vee \neg a_n \vee b_1 \vee \cdots \vee b_m)$ in a rule is $P(cl) = false \leftarrow a_1, \ldots, a_n, b'_1, \ldots, b'_m.$ and then, the encoding of a base S is

$$P(S) = \{P(cl) \mid cl \in S\} \cup \{x \leftarrow not\ x'.,\ x' \leftarrow not\ x. \mid x \in \mathcal{A}\} \cup \{bug \leftarrow false, not\ bug.\}$$

The intuition behind this translation stands on the following remarks (a similar process is exposed in [44]).

- x' is a new atom encoding the negative literal $\neg x$.
- Rules $x \leftarrow not\ x'.$ and $x' \leftarrow not\ x.$ allow to generate all possible classical propositional interpretations by doing an exclusive choice between x and $\neg x$ for each atom x in $\mathcal{A}$.
- The goal of each rule $P(cl)$ is to conclude $false$ (a new symbol) if the choice of atoms (x and $\neg x$) corresponds to an interpretation that does not satisfy the clause cl. By this way, if there exists a stable model not containing $false$, then it corresponds to a model of S (since every clause is satisfied).
- The goal of special rule $bug \leftarrow false, not\ bug.$, where bug is a new symbol, is to discard every stable model containing $false$. Since bug appears in the head and in the negative body of this rule and nowhere else, if a stable model exists then it may not contain $false$.

By this way there is a one to one correspondence between the propositional models of S and the stable models of $P(S)$. But, as stated in [39] there is no modular mapping from program to set of clauses, only a modular transformation from set of clauses to program exists. So, in a way, ASP has better knowledge representation capabilities than propositional logic and it is interesting to study how it can be extended to the possibilistic case in particular when there is an inconsistency. To reach our goal, we first extend the transformation P to a new transformation PP for the possibilistic case in a natural way. If $(cl, \alpha) \in \Sigma$, then its encoding keeps the same necessity degree α in $PP(\Sigma)$ and a necessity value equal to 1 is assigned to all the other rules (the "technical" ones).

Definition 17 Let $\Sigma = \{(cl_i, \alpha_i), i = 1, \ldots, n\}$ be a possibilistic base (in CNF), its encoding in a possibilistic normal logic program is:

$$\begin{aligned} PP(\Sigma) = &\{(P(cl_i), \alpha_i) \mid (cl_i, \alpha_i) \in \Sigma\} \\ &\cup \{(x \leftarrow not\ x'., 1), (x' \leftarrow not\ x., 1) \mid x \in \mathcal{A}\} \\ &\cup \{(bug \leftarrow false, not\ bug., 1)\} \end{aligned}$$

In the sequel we use $\mathcal{X} = \cup_{a \in \mathcal{A}} \{a, a'\} \cup \{false, bug\}$ to make the correspondence between the language of the propositional base and the one of its translation.

Definition 18 Let $\mathcal{A}$ be the set of atoms occurring in a possibilistic base Σ. Let $\mathcal{X}$ be the set of atoms occurring in the possibilistic normal logic program $PP(\Sigma)$. A *pseudo interpretation* for $\mathcal{A}$ is an atom set $X \subseteq \mathcal{X}$ such that $\forall a \in \mathcal{A}$, $(a \in X \vee a' \in X) \wedge (a \notin X \vee a' \notin X) \wedge bug \notin X \wedge false \notin X$.

The interesting point for the particular case of a possibilistic normal logic program encoding a possibilistic logic base is that, in this case, we are able to restore its consistency in only one step as it can be summarized in the figure 2.

In the following, we will say that a pseudo interpretation X *corresponds* to a classical interpretation ω if by translating each atom $a' \in X$ in literal $\neg a$, we obtain the interpretation ω. By this way, every stable model of $PP(\Sigma)^*$ is a pseudo interpretation corresponding to a classical model for Σ^* and conversely.

Proposition 15 *Let Σ be a possibilistic base and $\mathcal{A}$ its set of atoms. Let $P = PP(\Sigma)$ be the encoding of Σ, $\forall X \subseteq \mathcal{X}$ we have*

$$\begin{array}{c} X \text{ is not a pseudo interpretation for } \mathcal{A} \text{ and } \tilde{\pi}_P(X) = 0 \\ \text{or} \\ X \text{ is a pseudo interpretation and } \tilde{\pi}_P(X) = \pi_\Sigma(\omega) \\ \text{where } \omega \text{ is the interpretation that corresponds to } X \end{array}$$

Proposition 16 *Let Σ be a possibilistic base, then*

1. $Inc(\Sigma) = Incons Deg(PP(\Sigma))$
2. $\forall \alpha \in \mathcal{N}$, $PP(\Sigma_{>\alpha}) = PP(\Sigma)_{>\alpha}$
3. $Incons Deg(PP(\Sigma)) = 0 \Longrightarrow (PP(\Sigma))^*$ *has at least one stable model and every stable model corresponds to a propositional model of* Σ^*
4. $Incons Deg(PP(\Sigma)) = \alpha > 0 \Longrightarrow (PP(\Sigma)_{>\alpha})^*$ *has at least one stable model and every stable corresponds to a propositional model of* $(\Sigma_{>\alpha})^*$.

These results establish that our methodology exposed in figure 2 is valid. There is a total equivalence between the processing of a classical base with possibilistic logic and the processing of the corresponding possibilistic normal logic program.

Example 11 Let Σ be the following possibilistic base

$$\Sigma = \left\{ \begin{array}{l} (\neg e, 0.9), (b \vee c, 0.8), (\neg b \vee e, 0.7), (\neg a \vee b, 0.7), \\ (\neg d, 0.5), (a, 0.5), (\neg b \vee d, 0.3) \end{array} \right\}$$

Its encoding as a possibilistic normal logic program is

$$\begin{array}{l} PP(\Sigma) = \left\{ \begin{array}{l} (false \leftarrow e., 0.9), (false \leftarrow b', c'., 0.8), (false \leftarrow b, e'., 0.7), \\ \qquad (false \leftarrow a, b'., 0.7), \\ (false \leftarrow d., 0.5), (false \leftarrow a'., 0.5), (false \leftarrow b, d'., 0.3) \end{array} \right\} \\ \cup \{(x \leftarrow not\ x'., 1), (x' \leftarrow not\ x., 1) \mid x \in \{a, b, c, d, e\}\} \\ \cup \{(bug \leftarrow false, not\ bug., 1)\} \end{array}$$

possibilistic logic base		possibilistic normal logic program
inconsistent base Σ	$\Longrightarrow$	inconsistent program $PP(\Sigma)$
$\Downarrow$		$\Downarrow$
consistent subbase $\Sigma_{>\alpha}$	$\Longleftrightarrow$	consistent subprogram $PP(\Sigma)_{>\alpha}$
$\Downarrow$		$\Downarrow$
propositional model	$\Longleftrightarrow$	stable model
α is the inconsistency degree of Σ and $PP(\Sigma)$		

Figure 2 Relation between possibilistic logic and possibilistic stable model semantics.

Then, we have $IncDeg(PP(\Sigma)) = 0.5$ that corresponds to $Inc(\Sigma) = 0.5$ and the computed consistent subprogram of $PP(\Sigma)$ is

$$PP(\Sigma)_{>0.5} = \left\{ \begin{array}{c} (false \leftarrow e., 0.9), (false \leftarrow b', c'., 0.8), (false \leftarrow b, e'., 0.7), \\ (false \leftarrow a, b'., 0.7) \end{array} \right\} \cup\{(x \leftarrow not\ x'., 1), (x' \leftarrow not\ x., 1) \mid x \in \{a, b, c, d, e\}\} \cup\{(bug \leftarrow false, not\ bug., 1)\}$$

We obtain $PP(\Sigma)_{>0.5} = PP(\Sigma_{>0.5})$ and $(PP(\Sigma)_{>0.5})^*$ has two stable models $\{a', b', c, d, e'\}$ and $\{a', b', c, d', e'\}$. They correspond to the two propositional models: $\{\neg a, \neg b, c, d, \neg e\}$ and $\{\neg a, \neg b, c, \neg d, \neg e\}$ of $(\Sigma_{>0.5})^*$ the consistent subbase obtained in possibilistic logic.

5.3 Constraint relaxation

One application domain for ASP is the encoding of combinatorial problems in such a way that, given a problem A, the stable models of a program $P(A)$ are the solutions of A. As illustrated in example 2.1, designing $P(A)$ consists in writing three kinds of rules:

- *data rules* describing the particular data of the given instance,
- *guess rules* able to generate all the search space,
- *check rules*, or *constraints*, eliminating the points in the search space that are not solutions.

By this way, when A has no solution, the corresponding program $P(A)$ is inconsistent. In this case it may be interesting to relax some constraints in order to obtain an approximate solution of A. But which constraint has to be relaxed? In a real case problem (ex: a timetabling problem), it is usual to have different kinds of constraints. Some of them are impossible to circumvent (ex: each teacher cannot give two courses at the same time), but some others are only desirable (ex: do not place a course after 6PM). We see that all constraints can be ranked by level of importance (preference) and so our framework can encode A in a possibilistic normal logic program $PP(A)$. If $PP(A)$ is inconsistent, then by means of inconsistency degree our function *cut* can be used to relax some less important constraints. Then, the resulting subprogram has a stable model that represents an approximate solution of the initial problem A. We illustrate this proposal by pursuing our already introduced two-coloration graph problem.

Example 12 (example 2 continued) The coloring problem, by *red* or *green*, of the undirected graph $G = (\{v1, v2, v3\}, \{(v1, v2), (v2, v3), (v3, v1)\})$ has no solution and then P_{color} is inconsistent. In such a problem, edges are the constraints of the graph. So let us suppose that these constraints can be ranked, by means of an importance degree on every edge as it is illustrated in the left hand graph of figure 3.

Figure 3 Constraint relaxation.

The corresponding possibilistic normal logic program[5] that encodes this additional information is:

$$PP_{color} = \left\{ \begin{array}{l} (v(1) \leftarrow ., 1), \quad (v(2) \leftarrow ., 1), \quad (v(3) \leftarrow ., 1), \\ (e(1,2) \leftarrow ., 1), \ (e(2,3) \leftarrow ., 0.7), \ (e(3,1) \leftarrow ., 0.9), \\ (red(X) \leftarrow v(X), not\ green(X)., 1), \\ (green(X) \leftarrow v(X), not\ red(X)., 1), \\ (bug \leftarrow e(X,Y), red(X), red(Y), not\ bug., 1), \\ (bug \leftarrow e(X,Y), green(X), green(Y), not\ bug., 1) \end{array} \right\}$$

Then, $IconsDeg(PP_{color}) = 0.7$ and $cut(PP_{color}) = PP_{color>0.7}$ is a consistent possibilistic normal logic program. This subprogram $PP_{color>0.7}$ encodes a relaxation of the initial problem in which we eliminated the less important constraint as illustrated in the right hand graph of figure 3. Finally, each stable model of $cut(PP_{color})^*$ (like $\{red(1), green(2), green(3)\}$) encodes some approximate solutions of the initial problem A.

6 Possibilistic stable model computing

In this section we expose how we can compute the possibilistic stable models of a possibilistic normal logic program by using available softwares for ASP and we describe the main lines of the system that we have developed.

6.1 General algorithm

Let us first recall results of section 4.1 : given a possibilistic normal logic program P there is a one to one mapping between the possibilistic stable models of P and the stable models of the corresponding normal logic program P^* (proposition 10). This property ensures that, given S a stable model of P^* then $\Pi Cn(P^S)$ is a possibilistic stable model of P. Moreover, we have established (proposition 9) that the computation of $\Pi Cn(P^S)$ can be done polynomially from the moment we are provided with the stable model S. So, the computation of a possibilistic stable model

[5] As usual in ASP, rules with variables are a shortcut for a set of instantiated rules for which each certainty degree is that of the rule with variables from which it comes.

has not to be significantly harder than the computation of a classical stable model since this last one can be exponential. Despite this high level of complexity, some efficient ASP solvers are available today:

- DLV [26] `http://www.dbai.tuwien.ac.at/proj/dlv`
- Smodels [45] `http://www.tcs.hut.fi/Software/smodels`
- Cmodels [27] `http://www.cs.utexas.edu/users/tag/cmodels.html`
- Nomore++ [2] `http://www.cs.uni-potsdam.de/wv/nomore++`
- ...

So, the extension of one of them to compute possibilistic stable models has to be realizable without losing too much efficiency and the general algorithm for this purpose is sketched in figure 4.

Starting from this general algorithm we have developed a system in C++ by choosing Smodels as underlying ASP solver. Smodels works only for variable-free programs, so it is used with a front-end program Lparse that adds variables (and many others things) to the accepted language and generates a variable-free normal logic program. The choice of this solver has been guided by a compromise taking into account the system performances, the source code availability and our familiarity with the system. Moreover, any ASP system could have been used. But, the ability to clearly separate the grounding of the rules and the computation of the stable models has been also one reason of our choice.

Let us mention that from now and without loss of generality, we use necessity degrees that are natural integers. This integer set allows to represent the finite subset of interval [0, 1] used in the theoretical parts of this work. As recall in section 2.2, these values have not a numerical meaning, but only a qualitative and relative significance. So, for convenience we have chosen integer values to encode them.

Thus, an input file containing a possibilistic normal logic program will have to be presented as a sequence of expressions

$$\alpha \quad c \text{ :- } a_1, \ \ldots, \ a_m, \ not\ b_1, \ \ldots, \ not\ b_n.$$

where $\alpha \in \{1, \ldots, 100\}$ if 100 is enough and encodes the full certainty for instance. For the rest of our presentation we shall use this syntax when giving some examples.

6.2 Rule grounding

If we want to use ASP paradigm to solve some large and realistic problems, then using rules with variables to encode problems is absolutely necessary. As already mentioned, despite the propositional language, it is usual to consider programs with variables. Such a program is viewed as a compact representation of the propositional

computeAllPS M(in Solv : an ASP solver, P : a possibilistic normal logic program)
begin
  while (Solv(P^*) returns a stable model S
    write $\Pi Cn(P^S)$
  endwhile
end

Figure 4 General algorithm.

program obtained by replacing every variable with every constant of the language (its Herbrand instantiation). But, usually, most of the rules of the Herbrand instantiation have unsatisfiable bodies and they may be discarded without affecting the set of stable models. So we call a grounding, a process that transforms a normal logic program into an equivalent ground logic program. Lparse performs such a grounding whenever the program respects some syntactic conditions (see Lparse manual for details).

Example 13 Consider the following possibilistic normal logic program

$$P_1 = \left\{ \begin{array}{ll} 50\ b(X) \text{ :- } a(X), not\ c(X). & 100\ c(X) \text{ :- } a(X), not\ b(X). \\ 100\ a(1). & 20\ a(2). \\ 100\ b(2). & 80\ d(3). \end{array} \right\}$$

its classical part is

$$P_1^* = \left\{ \begin{array}{ll} b(X) \text{ :- } a(X), not\ c(X). & c(X) \text{ :- } a(X), not\ b(X). \\ a(1). & a(2). \\ b(2). & d(3). \end{array} \right\}$$

whose grounded version by Lparse is

$$\overline{P_1^*} = \left\{ \begin{array}{ll} b(1) \text{ :- } not\ c(1). & c(1) \text{ :- } not\ b(1). \\ b(2) \text{ :- } not\ c(2). & \\ a(1). & a(2). \\ b(2). & d(3). \end{array} \right\}$$

Note that no useless instantiation, for example with $X = 3$, is made. Actually, a rule like $b\,(3)$:- $a(3)$, not $c(3)$. is useless since $a(3)$ is impossible to derive with these rules. Moreover, many rule simplifications are made by Lparse as the deletion of $a(X)$ in the positive bodies of rules and the elimination of the rule $c(2)$:- $a(2)$, not $b\,(2)$. since $b\,(2)$ is given.

But, for possibilistic programs, we have first to block some partial evaluations : simplifications like the deletion of $a(X)$ in the three first grounded rules of the above example will not hold for a possibilistic program because they could modify the applicability degree of the rules. And, second, we have to keep the necessity degree affected to each rule with variables to every fully instantiated rule generated from this rule. These tasks are achieved via the following preprocessing that maps every possibilistic rule into a normal one in which a special new atom is inserted to record the certainty degree.

$$preproc(r) = r' \text{ such that } \left\{ \begin{array}{l} head(r') = head(r) \\ body^+(r') = body^+(r) \cup \{nu\_\_(n(r))\} \\ body^-(r') = body^-(r) \end{array} \right.$$

The generalization of this process to a possibilistic normal logic program P is defined as follows

$$\begin{array}{rl} Preproc(P) = & \{preproc(r) \mid r \in P\} \\ & \cup\ \{\sharp external\ nu\_\_(X).\} \\ & \cup\ \{nu\_\_(\alpha). \mid \alpha \in \mathcal{N}\} \end{array}$$

The directive $\sharp external\ nu\_\_(X).$ is a special feature used by Lparse to indicate that expressions $nu\_\_(X)$ are special atoms that could be given in a second step (see Lparse manual for details). Here, the useful point is that Lparse keeps every such atoms in the resulting grounded program. By this way, our initial goal: grounding every rule by keeping the trace of the necessity degree, is achieved as it can be seen in the two programs below.

Example 14 (example 13 continued)
Output of the preprocessing of P_1:

$$Preproc(P_1)=\left\{\begin{array}{ll} b(X)\text{:-}a(X), not\ c(X), nu\_\_(50). & c(X)\text{:-}a(X), not\ b(X), nu\_\_(100). \\ a(1)\text{:-}nu\_\_(100). & a(2)\text{:-}nu\_\_(20). \\ b(2)\text{:-}nu\_\_(100). & d(3)\text{:-}nu\_\_(80). \\ \sharp external\ nu\_\_(X). & \\ nu\_\_(100). \quad nu\_\_(80). & nu\_\_(50). \quad nu\_\_(20). \end{array}\right\}$$

Output of the grounding process done by Lparse :

$$\overline{Preproc(P_1)}=\left\{\begin{array}{ll} b(1) \text{ :- } a(1), nu\_\_(50), not\ c(1). & c(1) \text{ :- } a(1), nu\_\_(100), not\ b(1). \\ b(2) \text{ :- } a(2), nu\_\_(50), not\ c(2). & c(2) \text{ :- } a(2), nu\_\_(100), not\ b(2). \\ a(1) \text{ :- } nu\_\_(100). & a(2) \text{ :- } nu\_\_(20). \\ b(2) \text{ :- } nu\_\_(100). & d(3) \text{ :- } nu\_\_(80). \end{array}\right\}$$

We have implemented these techniques in a program named preprocLparse. It accepts a possibilistic logic program that may contain strong negations or constant declarations. In fact, these particular points have no influence on our preprocessing and they are managed as usual by Lparse. Thus, the following chain has to be used to realize whole preprocessing

```
preprocLparse inputfile | lparse --true-negation
```

where `--true-negation` is given to allow the treatment of strong negation. The output of this process is the input (in the internal Smodels' format) of our program posSmodels described in the next section.

6.3 Possibilistic stable model computation

Here, we describe how our system posSmodels computes the possibilistic stable models of a possibilistic normal logic program by using the grounded normal logic program produced by the preprocessing step described in previous subsection. The whole algorithm, using Smodels as underlying ASP solver, is presented in figure 5. The first thing to do is to read the output of the first step, that is a normal logic program encoded in the internal Smodels' format, and to rebuild its corresponding possibilistic logic program (see part 1 of the algorithm). By this way, and because of the preprocessing treatment, we are provided with a grounded possibilistic normal logic program P' where every rule is given with the right certainty degree. The second step is to build for Smodels the non possibilistic corresponding normal logic program SP' in order to compute its stable models. This is done via the Smodels programming API. Note that we deal with two representations of atoms and rules of the program : one for Smodels, and one for posSmodels. But this is the price to pay to stay independent of Smodels, and it is made so that we can directly access from one

kind of atoms to the other one. Thus, the computation of possibilistic stable models is now possible by following the general algorithm of the figure 4 detailed in the third part of figure 5. The whole process implements, the most efficiently as possible, the immediate possibilistic consequence operator ΠT_P introduced in definition 9.

Let us comment some particular points.

- Since we know that every set S is a stable model of P'^* then we can restrict our attention to the program $PP \subseteq P'^S$. But it is also useless to take into account rules with head or positive body not included in S since we already know that they never will be applicable in S. In fact, the only interesting ones are rules r of P' such that r^* is a *generating rule* of P'^*, ie : r^* is applicable in A and not blocked by A.
- $L(R)$ is a counter initiated with the body length of rule R and decreased each time a new atom belonging to the body of R is added in Res. By this way, at each step the applicable rules are those with $L(R) = 0$.
- An applicable rule with a certainty α and a head x can be dropped from PP when an atom (x, α) is added in Res since this rule (because of the definition of β-applicability) will never produce a new possibilistic atom (x, α') with $\alpha' > \alpha$. But, a rule cannot be discarded before since it can be used many times to produce

```
posSmodels(in P : a normal logic program)
begin
/* read normal logic program P produced by Lparse and construct corresponding possibilistic normal logic program P' */
    P' ← ∅
    while (read a rule R in P)
        body+(R) ← body+(R) \ {nu--(α)}
        P' ← P' ∪ {(R, α)}
    endwhile
/* build program SP' in Smodels (corresponding to P'*) */
    create program SP' in Smodels
    for each rule R ∈ P'
        create rule R* in SP'
    endfor
/* compute all possibilistic stable models */
    while (Smodels(SP') returns a stable model S) do
        PP ← {(r*+, n(r)) | r ∈ P', head(r*) ∈ S and body+(r*) ⊆ S and body−(r*) ∩ S = ∅}
        for each rule R ∈ PP compute L(R), the length of its (positive) body endfor
        Res ← ∅ /* the possibilistic stable model to compute */
        repeat
            FixPoint ← true
            for all rule R ∈ PP
                if (L(R) = 0) then
                    deg ← applicability degree of R in Res /* defi nition 8*/
                    if (head(R) ∉ Res*) then /* a new atom is added */
                        for each rule R's.t. head(R) ∈ body+(R')
                            L(R') ← L(R') − 1
                        endfor
                    endif
                    Res ← Res ⊔ {(head(R), deg)} /* ⊔ of defi nition 7 */
                    if (Res has been modifi ed) then FixPoint ← false endif
                    if (n(R) = deg) then /* this rule is now useless */
                        remove R from PP
                    endif
                endif
            endfor
        until FixPoint
        write Res
    endwhile
end
```

Figure 5 Possibilistic stable model computation.

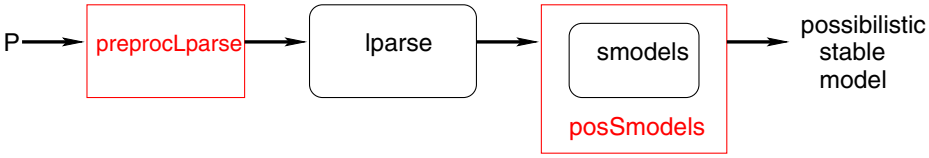


Figure 6 Process chain.

the same atom, but with an increasing certainty degree at each time. For instance, consider the following program P and the evolution of computation of the result Res.

$$P = \left\{ \begin{array}{l} 20\ a. \\ 100\ x. \\ 100\ b\ \text{:-}\ a. \\ 100\ a\ \text{:-}\ x. \end{array} \right\} \qquad \begin{array}{l} \text{Res} = \emptyset \\ \text{Res} = \{(a, 20), (x, 100)\} \\ \text{Res} = \{(x, 100), (b, 20), (a, 100)\} \\ \text{Res} = \{(x, 100), (a, 100), (b, 100)\} \end{array}$$

We can see that rule $100\ b$:- a. is used two times: firstly with $(a, 20)$ to produce $(b, 20)$, and secondly with $(a, 100)$ (when this atom has been produced by rule $100\ a$:- x.) to produce $(b, 100)$.

6.4 Examples and evaluations

Our whole system, sketched in the figure 6, implements in C++ the general algorithm of figure 5. It is available at www.info.univ-angers.fr/pub/pn/Softwares/PosSmodels. Its usage is:

```
preprocLparse inputfile | lparse --true-negation | posSmodels k
```

in order to obtain at most k possibilistic stable models. In the figure 7 we summarize some experimental results. In all cases our goal is to estimate the overhead of "possibilistic computation" so that is why we compare the performance of our whole system (as in the chain of the figure 6) with the performance of Lparse and Smodels on the same programs with or without certainty degrees. Note that, given PP a possibilistic program and $P = PP^*$ its classical part, if PP' is the grounded possibilistic program obtained by our system and P' the grounded program produced by Lparse, we have, in most cases, $PP'^* \neq P'$. The reason is that no partial evaluation is done by Lparse in the possibilistic case. So Lparse does much less work in our case than in classical case. At the opposite, the resulting work for Smodels is more important (rules can contain about twice more atoms due to the absence of simplifications). This explains why our treatment can sometimes be faster than the classical one : partial evaluations done by Lparse are not in these cases profitable.

– On the top graphs, we have reported some results about the computation of all possibilistic and classical stable models of a program encoding a Hamiltonian cycle problem in a so-called simplex graph. The program contains 672 atoms and 30183 rules and has 948 different models. We have successively computed 1, 100, …, 900 and all (classical and possibilistic) stable models.

– On the bottom graphs, we have reported some results about the computation of one, possibilistic or not, stable model of a program. This program encodes a problem of Hamiltonian path in a complete graph. The number of nodes in the graph growths from 5 to 35. For the last instance, the corresponding program has 3,747 atoms and 90,684 rules and Smodels needs roughly 100 s of CPU time to compute the first stable model.

In these graphs, we can see that the times consumed to treat each case (with or without certainty degrees) grow in a same way w.r.t. the number of models to compute and w.r.t. the difficulty to find one solution. Thus, we can say that our system posSmodels is efficient since the time dedicated to the possibilistic part computation is negligible compared to the time consumed to compute the classical stable models. This is in accordance with our theoretical results in proposition 9 and theorem 2.

7 Related works

7.1 Dealing with default reasoning and uncertainty

In the last years, many formalisms have been proposed to deal with non monotonic reasoning or uncertain reasoning. However, it has to be noticed that few formalisms deal conjointly with both features. In this last category, many of these works are underpinned by probability theory and very few by possibility theory.

First of all, some works use a framework of uncertainty to represent default reasoning. For instance [5] follows this line by using possibilistic logic. It uses the ordering defined by Pearl's System Z [40] on default rules to find an ordering on the corresponding formulas in possibilistic logic. This ordering is based on the specificity of default rules. For example, if we take the so-called penguin example defined by the default base *a bird flies*, *a penguin does not fly*, *a penguin is a bird* and the fact *a*

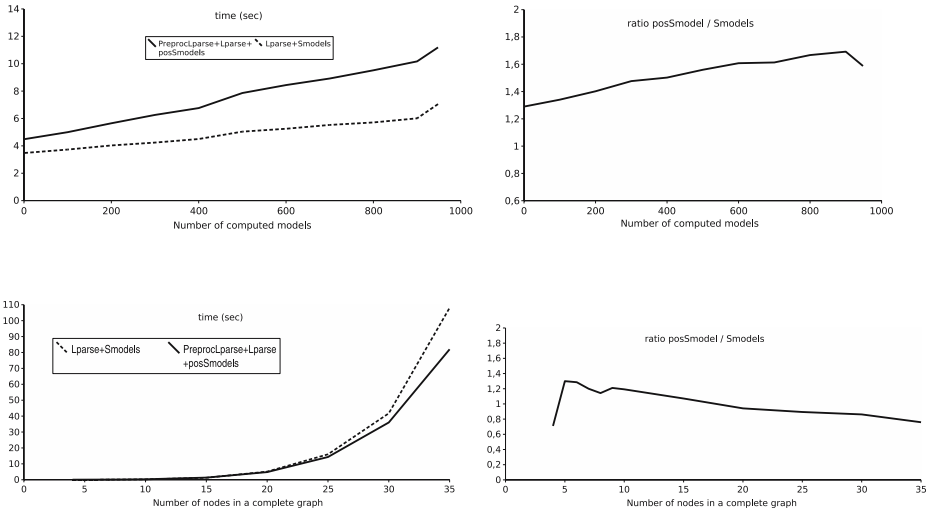


Figure 7 Performance evaluation.

penguin. The possibilistic base defined on these default rules is $\{(\neg b \vee f, 1/3), (\neg p \vee \neg f, 2/3), (\neg p \vee b, 2/3), (p, 1)\}$ since the rules applied to penguins are more specific than the ones applied to bird. The background idea is that they are less affected by exceptions and, then are considered as more certain. Here, the necessity degrees do not describe the certainty degree of each formula but encode the specificity of the rules and then captures the properties of plausible reasoning in the sense defined by system P and rational monotony [25]. Clearly, the uncertainty is no longer able to be represented.

Let us examine some works that merge non monotonicity and uncertainty in a classical logic framework and not a logic programming one. [14] is based on possibility theory and is about evolving systems using uncertain default transition rules. In this case, possibility theory is again used all alone to represent both uncertainty and default reasoning. In the probabilistic setting, [32] proposes several inference processes merging probabilistic logic respectively with system P entailment, system Z entailment and lexicographic entailment. These frameworks are able to merge both strict logical knowledge, default knowledge and probabilistic knowledge. One main difference with our work is that it uses logic and not logic programming (so it belongs to the philosophy of plausible reasoning in the sense of system P). Moreover, the same tool, called conditional constraint, is used to represent default logical knowledge and probabilistic knowledge. So, probability measures sometimes represent default reasoning and sometimes represent real probability. Whereas, in our framework, logic programming is used to represent default reasoning and necessity measures represent uncertainty.

Now, we cite some formalisms where logic programs are extended with various annotations encoding uncertainty. Here, there is a clear distinction between non monotonic reasoning and uncertainty, but the latter is not based on possibility theory. In [35], annotated non-monotonic rule systems are introduced to affect on every piece of knowledge a weight representing a probability, an uncertainty measure, a time, a place,... For instance for a logic program, it leads to rules like $(c, 0.4) \leftarrow (a, 0.5), (not\ b, 0.7)$. This approach is different from ours since we put weights on rules and not on each atom in the rules. Another divergence is the meaning of the weights : in our work we deal with uncertainty while they develop a formalism in which the semantics of weights is unspecified. For us, it seems difficult to develop a formalism in which the semantics of weights is not established a priori as in our work where weights are clearly interpreted in the framework of possibility theory. In [1, 6, 12, 13, 17, 29] the reader can find different propositions about multi-valued or probabilistic logic programs, about possibilistic definite logic programs, about levels of certainty ranking atoms (but not rules) involved in a non-monotonic reasoning,... But, none of these works describes a formalism dealing with uncertainty in a logic program with default negation by means of possibilistic theory, both in a semantic and a syntactic ways. In particular, in [30] many-valued disjunctive logic programs with probabilistic semantics are introduced with minimal, perfect and stable models for such programs. This work is completed in [31] by some fixpoint characterizations for different model semantics. This approach is similar to us but in the probability theory: an uncertainty numerical degree interpreted as a probability is associated to each clause and model theory and fixpoint semantics are given. In [4], the uncertain information which is represented is again strictly probabilistic. In other words, it comes from information given in terms of statistics and the values

have a meaning in themselves. In our framework, it is not the case since uncertainty is interpreted as a scale. Another example of probabilistic non monotonic formalism, clearly situated in the ASP paradigm, can be found in [42] that defines probabilistic well founded semantics and probabilistic stable model semantics. In this work it is possible to represent rules like $c : [0.9, 1] \leftarrow a : [0.6, 0.75], not(b : [0.8, 1])$ where every atom is annotated with a probability interval. Again, this is an important difference with our work, since we consider that the uncertainty concerns the whole rule and not each component of the rule.

To end this quick survey we consider that the following works are related to ours, either because they are concerned by default reasoning and possibility theory, either because of technical considerations they can be considered as a generalization of our framework. In [46], the approach is to reconstruct possibilistic logic by defining a multi-valued interpretation and a corresponding satisfaction relation and forgetting the notion of possibility distribution. Then, it is impossible to provide any result about the possibility of an atom set to be a stable model. Moreover, the necessity (certainty) of a conclusion is not correlated to the possibility of the atom set. So, we think that this approach is far from the spirit of possibilistic logic. In the same way, [34] proposes a general framework extending disjunctive programs and stable models semantics by introducing a reliability interval for each rule. From a syntactic point of view, our approach can be considered as a particular case of this one. But the described formalism do not refer explicitly to possibility theory to deal with uncertainty. It uses multivalued interpretations while we stay in the framework of stable models (which are bivalued). Finally, the description of a graded default logic can be found in [11]. It introduces a general logical framework dealing with uncertainty by using modality: each level of uncertainty is represented by a modality. Since stable models semantics can be seen as a reduction of default logic (see figure 1 and [7, 23]), this graded default logic can be considered as a generalization of our framework. However, [11] does not mention a possibility distribution.

7.2 Dealing with inconsistency

There are many families of methods to handle inconsistency in stratified knowledge bases. Our work is part of the ones that restore consistency by selecting one or several consistent subbases. In this family, our approach is a cautious one that deletes all knowledge under a level of inconsistency. A different way is to keep a maximal number of data in every stratum. For instance, in [8] the knowledge is given by a stratified formula set $T = T_1 \cup \cdots \cup T_n$ where the most important formulas are in T_1. The *preferred subtheory* of T is $S = S_1 \cup \cdots \cup S_n$ iff $\forall k, 1 \leq k \leq n$, $S_1 \cup \cdots \cup S_k$ is consistent and maximal. So, the strategy to extract a consistent subbase from an inconsistent one is, from the most important stratum to the less important one, to compute for each stratum a subset of formulas consistent with the union of the previous ones. The next example illustrates that this strategy may give a different result than our one if we apply it to normal logic programs.

Example 15 Let us consider the inconsistent program $P = P_1 \cup P_2 \cup P_3 \cup P_4$ with $P_1 = \{b \leftarrow not\ a.\}$, $P_2 = \{a \leftarrow not\ a.\}$, $P_3 = \{a \leftarrow not\ b.\}$ and $P_4 = \{b \leftarrow not\ b.\}$. The preferred subtheory approach of [8] leads to the consistent subprogram $S = P_1 \cup \emptyset \cup P_3 \cup P_4 = \{b \leftarrow not\ a., a \leftarrow not\ b., b \leftarrow not\ b.\}$ that has a unique stable

model $\{b\}$. On our side, we can represent the different strata of P by means of the possibilistic normal logic program $PP = \{(b \leftarrow not\ a., 1), (a \leftarrow not\ a., 0.8), (a \leftarrow not\ b., 0.6), (b \leftarrow not\ b., 0.4)\}$. Then, we find $InconsDeg(PP) = 0.4$ and so $cut(PP) = PP_{>0.4} = \{(b \leftarrow not\ a., 1), (a \leftarrow not\ a., 0.8), (a \leftarrow not\ b., 0.6)\}$ that is consistent and such that $cut(PP)^*$ has a unique stable model $\{a\}$.

To summarize, we can say that [8] has a top-down conservative approach, when we use a bottom-up destructive approach. Furthermore, for an inconsistent logic base Σ handled with a possibilistic approach, the consistent subbase $\Sigma_{>Inc(\Sigma)}$ is always a subset of the preferred subtheories of Σ. Whereas the example 15 shows that it is not always the case for the normal logic programs and the difference comes from the non monotonic nature of stable model semantics.

Finally, let us mention that our proposal about constraint relaxation described in 5.3 deals with over-constrained logic programs for which other works exist as Hierarchical Constraint Logic Programming [47]. But, this approach addresses the problem in a different way from ours, by a hierarchy of degrees and some error and comparator functions to choose between different solutions (see [24] for a survey on over-constrained systems).

7.3 Dealing with preferences

In possibilistic logic, the necessity degrees are commonly interpreted as preferences between formulas, the more certain is a formula, the more preferred it is. Since there is a lot of works dealing with preferences between rules in non-monotonic reasoning [15], it is interesting to analyze our work when we consider that the necessity degrees on rules determine a preference order. If we look only on the area of ASP, most of these works use the preferences expressed between the rules to make a choice between the different stable models to keep only the preferred ones [9, 10]. In other words, the priorities between the rules do not evaluate a certainty degree of the rules but are a tool to choose between contradicting rules. This differs from our work because, when several stable models exist, we keep all of them. Additionally, we compute the certainty of the propositions with respect to each stable model. But we do not try to eliminate some stable models since we consider them as alternate solutions.

More precisely, [10] deals with normal logic programs. When a normal logic program has several answer sets, it consists in selecting some of them, the preferred ones, with respect to the priorities expressed between the rules. The difference also occurs in term of expressing priorities between the rules: like in possibilistic logic, we propose a total ordering whereas, Brewka and Eiter propose a partial ordering. This is due to the fact that, in our work, we want an evaluation of the certainty and every rule is evaluated whereas, in [10], the priorities are expressed between the potentially conflicting rules. It is however interesting to compare the two approaches and for this, we focus on the examples given in [10].

Example 16 To express the strong negation we use the encoding described at the end of section 4 and we express the preference relations by means of a corresponding choice of certainty degrees. If the rule r_1 is preferred to the rule r_2, then $n(r_1) > n(r_2)$.

In the following p stands for *penguin*, b for *bird* and f for *fly*.

$$P_{pen} = \left\{ \begin{array}{l} (p \leftarrow ., 1), (b \leftarrow ., 1), (f \leftarrow b, not\ f'., 0.4), (f' \leftarrow p, not\ f., 0.8), \\ (bug \leftarrow f, f', not\ bug., 1) \end{array} \right\}$$

Let us compute the possibility distribution and focus on the sets containing neither *bug*, neither both f and f' (since the others are impossible to obtain and their possibility degree is 0):

$$\begin{array}{llll} \pi_{P_{pen}}(\emptyset) = 0; & \pi_{P_{pen}}(\{f\}) = 0; & \pi_{P_{pen}}(\{p, f\}) = 0; & \pi_{P_{pen}}(\{b, f'\}) = 0; \\ \pi_{P_{pen}}(\{p\}) = 0; & \pi_{P_{pen}}(\{f'\}) = 0; & \pi_{P_{pen}}(\{p, f'\}) = 0; & \pi_{P_{pen}}(\{p, b, f\}) = 1; \\ \pi_{P_{pen}}(\{b\}) = 0; & \pi_{P_{pen}}(\{p, b\}) = 0.2; & \pi_{P_{pen}}(\{b, f\}) = 0; & \pi_{P_{pen}}(\{p, b, f'\}) = 1; \end{array}$$

This possibility distribution indicates two stable models for P^*_{pen} that are $S_1 = \{p, b, f\}$ and $S_2 = \{p, b, f'\}$. Then $A_1 = \{(p, 1), (b, 1), (f, 0.4)\}$ and $A_2 = \{(p, 1), (b, 1), (f', 0.8)\}$ are the two possibilistic stable models of P_{pen}. From these two sets, we can conclude that $\neg f$ is more plausible than f which agrees with result in [10] where the only one preferred answer set is $\{p, b, \neg f\}$.

Example 17 Let us take the following program

$$P = \left\{ \begin{array}{l} (b \leftarrow a, not\ b'., 0.9), (b' \leftarrow not\ b., 0.8), (a \leftarrow not\ a'., 0.7), \\ (bug \leftarrow a, a', not\ bug., 1), (bug \leftarrow b, b', not\ bug., 1) \end{array} \right\}$$

P has two possibilistic stable models of $A_1 = \{(a, 0.7), (b, 0.7)\}$ and $A_2 = \{(a, 0.7), (b', 0.8)\}$. So, $\neg b$ is more preferred than b. This does not agree with principle I in [10] where the preferred answer set is A_1 which contains b. Nevertheless, we can argue that this is due to two different points of view. In [10], the priorities are used to make a choice between the rules but the deduced proposition are considered equally sure in each answer set. So, in the example, when a is deduced, it is natural to prioritize the first rule which leads to b over the second one which leads to $\neg b$. In our point of view, each proposition has a certainty degree in the answer set. So,the applicability of the rules does not only depend on the level of the rule but also on the level of the propositions allowing to apply the rule. In the example, the level of the first rule is dependent on the level of certainty of proposition a.

The next example deals with principle II in [10] for priorities.

Example 18 Let us take the following program:

$$P = \left\{ \begin{array}{l} (b \leftarrow a, not\ b'., 0.9), (a' \leftarrow not\ a., 0.8), (a \leftarrow not\ a'., 0.7), \\ (bug \leftarrow a, a', not\ bug., 1), (bug \leftarrow b, b', not\ bug., 1) \end{array} \right\}$$

$A_1 = \{(a, 0.7), (b, 0.7)\}$ and $A_2 = \{(a', 0.8)\}$ are the two possibilistic stable models of P. Here, we prefer $\neg a$ over a. This is in accordance with [10] where preferred answer set is A_2. It is natural to observe this fact here, since when there is no condition for applying rules, the choice is only made by using the priority defined between these rules.

8 Conclusion

In this paper we have achieved our goal : defining an extension of the Answer Set Programming paradigm for incomplete and uncertain knowledge representation and reasoning. Our framework of possibilistic stable model semantics embeds in a unified way two aspects of common sense reasoning : non monotonicity and uncertainty. Each part is underpinned by a well established formalism : stable models semantics and possibility theory. In a theoretical point of view, we have given some formal results ensuring the clear semantics of our proposal. We have linked this semantics to a syntactic process able to derive uncertain conclusions from a knowledge base expressed by means of a possibilistic normal logic program. We have shown that the decision problem induced by our formalism are in the same complexity class as those of the classical stable model semantics without uncertainty. We have also proposed a method to restore the consistency of a program by interpreting certainty degrees on rules as a way to choose which rules to eliminate. In a practical point of view, we have developed an extension of system Smodels that allows to efficiently use our framework for real world applications.

To pursue this work, we can envisage several perspectives. In the area of ASP it should be interesting to study how we can extend our framework to possibilistic disjunctive logic programs. With regard to inconsistency, a new methodology, always based on certainty degrees, should be studied to try to keep a great number of rules especially those that are not concerned by inconsistency. About the interpretation of certainty degrees as preference relations between rules, a deeper analysis is needed to understand how the two approaches are related. Also, we should have to see how to apply our framework and use our system to solve weighted CSP. To end, and about software, we could try to incorporate the certainty degrees handling directly in the searching process of a stable model instead to do that in a second separate step.

9 Result proofs

Proof (**Proposition 1**)
$\Rightarrow$: Firstly, since A is the model of P then $A = Cn(P)$ and it is the least fixpoint of T_P, so $A = head(App(P, A))$ is obvious. Secondly, the iterative computation of the least fixpoint of T_P (described in the first column below) produces a sequence of rule sets (described in the second column below).

$A_0 = \emptyset$	$R_0 = \emptyset$
$A_1 = T_P(A_0)$	$R_1 = App(P, A_0)$
$A_2 = T_P(A_1)$	$R_2 = App(P, A_1)$
...	...
$A_k = T_P(A_{k-1}) = A_k$	$R_k = App(P, A_{k-1}) = R_k$
$A_k = A$	$R_k = App(P, A)$
(*least fixpoint*)	

So, we can order the final set R_k as $R_k = \langle r_1^1, \ldots, r_{n_1}^1, r_1^2, \ldots, r_{n_2}^2, \ldots, r_1^k, \ldots, r_{n_k}^k \rangle$ by respecting $\forall i = 1, \ldots, k, \forall j = 1, \ldots, n_i, r_j^i \in R_i \setminus \cup_{l<i} R_l$. By this way, the groundedness of $App(P, A)$ is proven.

$\Longleftarrow$: $A = head(App(P, A)) \Rightarrow A = T_P(A)$ by definition of T_P. Thus A is a fixpoint of T_P. By groundedness of $App(P, A)$ the least fixpoint of T_P is a set $B \supseteq A$ (see the two iterations described above) and then A is the least fixpoint of T_P and so the least Herbrand model of P.

Proof (**Proposition 2**)
The definition of π_P ensures obviously its compatibility and the minimality of specificity is ensured by the fourth point.

Proof (**Proposition 3**)

1. $\Rightarrow$: If $\pi_P(A) = 1$, then $A \subseteq head(App(P^*, A))$ and $head(App(P^*, A)) \subseteq A$ since $\forall r \in P, A \models r^*$ (by proposition 2). So $A = head(App(P^*, A))$. In addition, since $\pi_P(A) > 0$ we have $App(P^*, A)$ is grounded (by proposition 2). Thus, by proposition 1, $A = Cn(P^*)$.
   $\Leftarrow$: If $A = Cn(P^*)$, then (by proposition 1) we have $A \subseteq head(App(P^*, A))$ and $App(P^*, A)$ is grounded, so $\pi_P(A) > 0$. In addition, we have $A = head(App(P^*, A))$ (by proposition 1) and so $\forall r \in P, A \models r^*$, and thus $\pi_P(A) = 1$.
2. $A \supset Cn(P^*) \Rightarrow A \supset head(App(P^*, A)) \vee App(P^*, A)$ is not grounded by proposition 1. So, by proposition 2, $\pi_P(A) = 0$.
3. $\emptyset \subseteq head(App(P^*, \emptyset))$ and $App(P^*, \emptyset)$ is grounded are two obvious properties. So $\pi_P(\emptyset)$ is only defined by the fourth case in proposition 2 and since $Cn(P^*) \neq \emptyset$ we have

$$\begin{aligned}\pi_P(\emptyset) &= 1 - \max_{r \in P}\{n(r) \mid \emptyset \not\models r^*\} \\ &= 1 - \max_{r \in P}\{n(r) \mid body^+(r^*) = \emptyset\}\end{aligned}$$

Proof (**Proposition 4**)

1. By definition 4 and propositions 1 and 2, $\pi_P(Cn(P^*)) = 1$ and thus $x \in Cn(P^*) \Rightarrow \Pi_P(x) = 1$ by definition 5.
   Otherwise, if $x \notin Cn(P^*)$ let us suppose that $\Pi_P(x) > 0$. Then, by definition 5 $\exists A \subseteq \mathcal{X}, x \in A, \pi_P(A) > 0$ and $A \neq Cn(P^*)$ since $x \notin Cn(P^*)$. So, by proposition 1, we have 2 cases :
   - $A \neq head(App(P^*, A))$ and then $\pi_P(A) = 0$ by definition 4, that is a contradiction.
   - $App(P^*, A)$ is not grounded and then $\pi_P(A) = 0$ by definition 4, that is a contradiction.

   Thus, $\Pi_P(x) = 0$.
2. Since $\pi(Cn(P^*)) = 1$, then $N_P(x) = 0$ is obvious when $x \notin Cn(P^*)$. Conversely, if $N_P(x) = 0$ then $\exists A \subseteq \mathcal{X}, x \notin A$ and $\pi(A) = 1$. So, such a set A must be $Cn(P^*)$, since, by proposition 3, it is the unique set with a possibility of 1. Thus, $x \notin Cn(P^*)$.
3. $$\begin{aligned}N_P(x) &= 1 - \max_{A \in 2^{\mathcal{X}}}\{\pi_P(A) \mid x \notin A\} \\ &= 1 - \max_{A \subseteq Cn(P^*)}\{\pi_P(A) \mid x \notin A\} \text{ by proposition 3} \\ &= 1 - \max_{A \subset Cn(P^*)}\{\pi_P(A) \mid x \notin A\} \text{ since } x \in Cn(P^*)\end{aligned}$$

Since we consider $x \in Cn(P^*)$, $Cn(P^*)$ is not empty and so by proposition 3 we have at least one set A (for instance $A = \emptyset$) such that $\pi_P(A) = 1 - \max_{r \in P}\{n(r) \mid A \not\models r^*\}$. Thus

$$N_P(x) = 1 - \max_{A \subset Cn(P^*)} \{1 - \max_{r \in P}\{n(r) \mid A \not\models r^*\} \mid x \notin A\}$$
$$= \min_{A \subset Cn(P^*)} \{\max_{r \in P}\{n(r) \mid A \not\models r^*\} \mid x \notin A\}.$$

4. Firstly, $x \notin Cn(P^*) \Rightarrow N_P(x) = 0$ (by item 2 in this proposition) and so $N_P(x) \leq N_{P'}(x)$.
Secondly, let us consider $x \in Cn(P^*)$. Then, we have also $x \in Cn(P'^*)$.

$$P \subseteq P' \Rightarrow \forall A \in 2^{\mathscr{X}}, \max_{r \in P}\{n(r) \mid A \not\models r^*\} \leq \max_{r \in P'}\{n(r) \mid A \not\models r^*\}$$
$$\Rightarrow \min_{A \in 2^{\mathscr{X}}} \{\max_{r \in P}\{n(r) \mid A \not\models r^*\}, x \notin A\} \leq \min_{A \in 2^{\mathscr{X}}} \{\max_{r \in P'}\{n(r) \mid A \not\models r^*\}, x \notin A\}$$
$$\Rightarrow N_P(x) \leq N_{P'}(x) \text{ by item 3 in this proposition.}$$

Proof (**Proposition 5**)
Obvious by proposition 4 item 2.

Proof (**Proposition 6**).
There is no particular difficulty to obtain this result.

Proof (**Proposition 7**)
The proof of the monotonicity of ΠT_P relies on the use of max operator and on the obvious fact:

$$A \sqsubseteq B \Rightarrow \forall x \in head(P^*), App(P, A, x) \subseteq App(P, B, x)$$

Proof (**Proposition 8**)
This result is similar to this in [17] and the existence and the characterization of the least fixpoint of ΠT_P is ensured by the Knaster–Tarski theorem.

Proof (**Proposition 9**)
It is known that Dowling–Gallier algorithm [16] computes the least Herbrand model of a definite logic program in polynomial time with respect to the number of rules in the program. In our case the polynomial time is now defined with respect to number of rules in the program $\times$ number of different uncertainty degrees occurring in the program (see algorithm in figure 5 in subsection 6.3).
So $\Pi Cn(P)$ can be computed in polynomial time.

Proof (**Theorem 1**)
$\subseteq$: Given a possibilistic definite logic program P and its associated necessity measure N_P, we start by proving $\Pi Cn(P) \subseteq \Pi M(P)$.

By definition of $\Pi Cn(P)$ that is the least fixpoint of ΠT_P, we can firstly remark that $\forall x \in \mathscr{X}, \alpha \in \mathscr{N}, (x,\alpha) \in \Pi Cn(P) \Rightarrow \alpha > 0$. Let us suppose that

$$N_P(x) = \alpha' > \alpha \tag{1}$$

Since $x \in Cn(P^*)$, by proposition 4 item 3, we have:

$$N_P(x) = \min_{A \in 2^{\mathscr{X}}} \{\max_{r \in P}\{n(r) \mid A \not\models r^*\}, x \notin A\} = \alpha'$$

$$\Rightarrow \forall A \in 2^{\mathscr{X}}, x \notin A, \max_{r \in P}\{n(r) \mid A \not\models r^*\} \geq \alpha'$$

$$\Rightarrow \forall A \in 2^{\mathscr{X}}, x \notin A, \exists r \in P \mid A \not\models r^*, n(r) \geq \alpha' \tag{2}$$

So, by using (2) at each step i, we can build the two maximal following sequences $\langle A_i \rangle$ and $\langle r_i \rangle$ such that $\forall i, 0 \geq i \geq k, x \notin A_i^*$ and $head(r_i^*) \neq x$

$A_0 = \emptyset \Rightarrow \exists r_0 \in P \mid A_0^* \not\models r_0^*, n(r_0) \geq \alpha'$

$A_1 = \{(head(r_0^*), n(r_0))\} \Rightarrow \exists r_1 \in P \mid A_1^* \not\models r_1^*, n(r_1) \geq \alpha'$

...

$A_k = \{(head(r_0^*), n(r_0), \ldots, (head(r_{k-1}^*), n(r_{k-1}))\} \Rightarrow \exists r_k \in P \mid A_k^* \not\models r_k^*, n(r_k) \geq \alpha'$

$A_{k+1} = \{(head(r_0^*), n(r_0), \ldots, (head(r_k^*), n(r_k))\}$

$\langle r_0, \ldots, r_k \rangle$ is maximal, ie : $\neg\exists r \in P \mid head(r^*) \neq x$ and $A_{k+1}^* \not\models r$. But, since

$$x \notin A_{k+1}^* \text{ and } A_{k+1} \sqsubseteq \Pi Cn(P) \text{ and } x \in \Pi Cn(P)^*$$

we have

$$\forall r \in P, A_{k+1}^* \not\models r \Rightarrow head(r^*) = x$$

and in particular, by using (2) again, we have

$$\exists \rho \in P \mid A_{k+1}^* \not\models \rho^*, n(\rho^*) \geq \alpha' \text{ and } head(\rho^*) = x.$$

In addition, $\forall (y,\beta) \in A_{k+1}, \beta \geq \alpha'$ and since $n(\rho^*) \geq \alpha'$, we have $(head(\rho^*), \gamma) = (x,\gamma) \in \Pi Cn(P)$ with $\gamma \geq \alpha' > \alpha$. This is in contradiction with the fact $(x,\alpha) \in \Pi Cn(P)$ since $\Pi Cn(P) = \sqcup_{n \geq 0} \Pi T_P^n(\emptyset)$. So, our initial supposition (1) is false and then $N_P(x) \leq \alpha$.

Furthermore, we have

$$\begin{aligned}(x,\alpha) \in \Pi Cn(P) &\Rightarrow \exists P' \subseteq P, (x,\alpha) \in \Pi Cn(P') \text{ and } P' \text{ is minimal} \\ &\quad (\text{ie: } \forall P'' \subseteq P', (x,\alpha) \notin \Pi Cn(P'')) \\ &\Rightarrow \forall r \in P', n(r) \geq \alpha \text{ (because of properties of } \Pi T_P) \\ &\Rightarrow N_{P'}(x) \geq \alpha \text{ (by definition 5 and proposition 4)} \\ &\Rightarrow N_P(x) \geq \alpha \text{ (by proposition 4)}\end{aligned}$$

Thus, $N_P(x) = \alpha$ and then $(x,\alpha) \in \Pi M(P)$, so $\Pi Cn(P) \subseteq \Pi M(P)$.

$\supseteq$: Now in order to prove $\Pi M(P) \subseteq \Pi Cn(P)$, let us first remark the following equivalences $\forall x \in \mathscr{X}$

$$\begin{array}{ccc} x \notin Cn(P^*) & \Longleftrightarrow & N_P(x) = 0 \\ \Updownarrow & & \Updownarrow \\ \neg\exists \beta, \beta > 0, (x, \beta) \in \Pi Cn(P) & & \neg\exists \beta, \beta > 0, (x, \beta) \in \Pi M(P) \end{array}$$

Then, we have,

$$\forall x \in \mathscr{X}, \exists \beta, \beta > 0, (x, \beta) \in \Pi Cn(P) \iff \exists \beta, \beta > 0, (x, \beta) \in \Pi M(P)$$

and so,

$$\begin{array}{c} \forall x \in \mathscr{X}, \alpha > 0, (x, \alpha) \in \Pi M(P) \Rightarrow \exists \beta, \beta > 0, (x, \beta) \in \Pi Cn(P) \\ \Rightarrow \exists \beta, \beta > 0, (x, \beta) \in \Pi M(P) \text{ (because of the first part of this proof)} \\ \Rightarrow \beta = \alpha \text{ since } \Pi M(P) = \{(x, N_P(x))\} \Rightarrow (x, \alpha) \in \Pi Cn(P) \end{array}$$

This ends the proof of $\Pi Cn(P) = \Pi M(P)$.

Proof (**Proposition 10**)

1. By definition 11, A is the set of possibilistic consequences of P^{A^*}. So, by theorem 1 the result is immediate.
2. Firstly let us remark that $(P^A)^* = (P^*)^A$ and that P^A is a possibilistic definite logic program, so we have.
   - $\forall x \in \mathscr{X}, N_{P^A}(x) > 0 \Rightarrow x \in Cn((P^A)^*)$ by proposition 4 and so $x \in Cn((P^*)^A)$. Thus, $x \in A$ since $A = Cn((P^*)^A)$ (A is a stable model of P^*).
   - $\forall x \in \mathscr{X}, x \in A \Rightarrow x \in Cn((P^*)^A)$ since A is a stable model of P^* and so $x \in Cn((P^A)^*)$. Thus, $N_{P^A}(x) > 0$ by proposition 4.

   So, we have $\{x \in \mathscr{X}, N_{P^A}(x) > 0\} = A$ and then

   $$\{(x, N_{P^A}(x)) \mid x \in \mathscr{X}, N_{P^A}(x) > 0\}^* = A \tag{3}$$

   For a better lisibility, we denote $S = \{(x, N_{P^A}(x)) \mid x \in \mathscr{X}, N_{P^A}(x) > 0\}$ and since P^A is a possibilistic definite logic program, we have

   $$\begin{array}{rl} & S = \Pi M(P^A) \text{ by definition} \\ \Longleftrightarrow & S = \Pi Cn(P^A) \text{ by theorem 1} \\ \Longleftrightarrow & S = \Pi Cn(P^{S^*}) \text{ by previous equality (3)} \end{array}$$

   Thus, $S = \{(x, N_{P^A}(x)) \mid x \in \mathscr{X}, N_{P^A}(x) > 0\}$ is a possibilistic stable model of P.
3. If A is a possibilistic stable model of P so it is the possibilistic model of $P^{(A^*)}$ (by definition 11). Thus A^* is the model of $\left(P^{(A^*)}\right)^*$ (by proposition 5), so A^* is a stable model of P^*.

Proof (**Theorem 2**)
For a given possibilistic normal logic program P, the second and third items of proposition 10 lead to:

P has at least one possibilistic stable model $\iff$ P^* has at least one stable model.

Knowing that the problem of deciding whether a normal logic program has a stable model is *NP−complete* [33] the result holds.

Proof (**Proposition 11**)
$\Rightarrow$: $\tilde{\pi}_P(A) = 1 \Rightarrow \pi_{P^A}(A) = 1$ (by definition 12) $\Rightarrow A = Cn(P^{A^*})$ (by proposition 3), so A is a stable model of P^* (by definition 1).
$\Longleftarrow$: A is a stable model of $P^* \Rightarrow A = Cn(P^{A^*})$ (by definition 1) $\Rightarrow \pi_{P^A}(A) = 1$ (by definition 3) $\Rightarrow \tilde{\pi}_P(A) = 1$ (by definition 12).

Proof (**Proposition 12**)

1. x is a credulous consequence of P^*
   $\Longleftrightarrow \exists A \in 2^{\mathscr{X}}$ s.t. $x \in A$ and A is a stable model of P^*
   $\Longleftrightarrow \exists A \in 2^{\mathscr{X}}$ s.t. $x \in A$ and $\tilde{\pi}_P(A) = 1$ by proposition 11
   $\Longleftrightarrow \tilde{\Pi}_P(x) = 1$ by definition 13
2. $\Rightarrow$: x is not a skeptical consequence of the consistent possibilistic normal logic program $P^* \Rightarrow \exists A \in 2^{\mathscr{X}}$ a stable model of P^* s.t. $x \notin A$. In addition we have $\tilde{\pi}_P(A) = 1$ by proposition 11 and thus $\tilde{N}_P(x) = 0$ by definition.
   $\Longleftarrow$: $\tilde{N}_P(x) = 0 \Rightarrow \max_{A \in 2^{\mathscr{X}}} \{\tilde{\pi}_P(A) \mid x \notin A\} = 1 \Rightarrow \exists A \in 2^{\mathscr{X}}, x \notin A, \tilde{\pi}_P(A) = 1 \Rightarrow A$ is a stable model of P^* that does not contain x, thus x is not a skeptical consequence of P^*.

Proof (**Proposition 13**)

1. It is a direct consequence of proposition 11.
2. If P is consistent the result is obvious since $InconsDeg(P) = 0$ by first item of this proposition. So, in the sequel of this proof, we consider an inconsistent possibilistic normal logic program P.
   Since $\mathscr{N}$ is a finite set we can consider $\mathscr{N} = \{\alpha_1, \ldots, \alpha_n\}$ such that $\forall i, 1 \leq i < n, \alpha_i > \alpha_{i+1}$ and we denote $\forall i, 1 \leq i \leq n, P_i = P_{>\alpha_i}$.
   Let $\alpha_k = ConsCutDeg(P)$, so P_k is consistent and so $\exists A \in 2^{\mathscr{X}}$, A is a stable model of $(P_k)^*$. We have $P^A = (P_k)^A \cup P'^A$ and we show $P'^A \neq \emptyset$. If this were not the case, $P'^A = \emptyset \Rightarrow Cn(P^A) = Cn((P_k)^A) = A$, since A is a stable model of $P_k(A)$. Thus A is a stable model of P, that is a contradiction since P is inconsistent and so $P'^A \neq \emptyset$.

$$\left.\begin{array}{r} A \text{ is a stable model of } (P_k)^A \Rightarrow \forall r \in (P_k)^A, A \models r^* \\ \forall r \in P'^A, n(r) \leq \alpha_k \end{array}\right\} \Rightarrow \max_{r \in (P_k)^A \cup P'^A} \{n(r) \mid A \not\models r^*\} \leq \alpha_k$$

$$\Rightarrow \pi_{(P_k)^A \cup P'^A}(A) = 1 - \max_{r \in (P_k)^A \cup P'^A} \{n(r) \mid A \not\models r^*\} \geq 1 - \alpha_k$$

$$\Rightarrow \tilde{\pi}_P(A) = \pi_{P^A}(A) = \pi_{(P_k)^A \cup P'^A}(A) \geq 1 - \alpha_k$$

$$\Rightarrow \max_{A \in 2^{\mathscr{X}}} \tilde{\pi}_P(A) \geq 1 - \alpha_k$$

$$\Rightarrow InconsDeg(P) = 1 - \max_{A \in 2^{\mathscr{X}}} \{\tilde{\pi}_P(A)\} \leq \alpha_k = ConsCutDeg(P)$$

Proof (**Proposition 14**)
Let P be a possibilistic normal logic program. If P is consistent then $P_{>ConsCutDeg(P)} = P$ and $InconsDeg(P) = 0$, so the result is obvious.

Let P be an inconsistent possibilistic normal logic program.

$$\Rightarrow 0 < InconsDeg(P) \leq ConsCutDeg(P) \text{ by proposition 13}$$
$$\Rightarrow P_{>ConsCutDeg(P)} \subseteq P_{>InconsDeg(P)} \subset P$$

So, at each time it is called, *cut* reduces the number of rules in the given program by eliminating some less certain strata. Additionally, the result is ever a superset of $P_{>ConsCutDeg(P)}$. Since the number of strata in P is finite and all strict supersets of $P_{>ConsCutDeg(P)}$ are inconsistent, then *cut* stops and returns $P_{>ConsCutDeg(P)}$ after a finite number of calls .

Proof (**Proposition 15**)
We note $r_{bug} = bug \leftarrow false, not\ bug.$ and recall $\tilde{\pi}_P(X) = \pi_{P^X}(X)$.

- if X is not a pseudo interpretation, we have the following cases
  - $bug \in X \Rightarrow r_{bug} \notin P^X \Rightarrow r_{bug} \notin App((P^X)^*, X) \Rightarrow bug \notin head(App((P^X)^*, X)) \Rightarrow \pi_{P^X}(X) = 0 \Rightarrow \tilde{\pi}_P(X) = 0.$
  - $bug \notin X \wedge false \in X \Rightarrow X \not\models r_{bug}$ and since $n(r_{bug}) = 1$ then $\tilde{\pi}_P(X) = 0$.
  - $bug \notin X \wedge false \notin X \wedge \exists x \in \mathscr{A}, x \in X \wedge x' \in X \Rightarrow x \leftarrow not\ x' \notin App((P^X)^*, X) \wedge x' \leftarrow not\ x \notin App((P^X)^*, X) \Rightarrow X \not\subseteq head(\mathrm{App}((P^X)^*, X))$ and then $\tilde{\pi}_P(X) = 0$.
  - $bug \notin X \wedge false \notin X \wedge \exists x \in \mathscr{A}, x \notin X \wedge x' \notin X \Rightarrow X \not\models x \leftarrow not\ x'$ and since $n(x \leftarrow not\ x') = 1$ then $\tilde{\pi}_P(X) = 0$.

  thus, in every case where X is not a pseudo-interpretation, $\tilde{\pi}_P(X) = 0$.
- if X is a pseudo interpretation, then $\forall a \in \mathscr{A}$, one and only one of the two rules $a \leftarrow not\ a'$ and $a' \leftarrow not\ a$ is in $App((P^X)^*, X)$ and then $App((P^X)^*, X)$ is grounded and $X \subseteq head(App((P^X)^*, X))$.
  Let ω be the interpretation for Σ that corresponds to X.
  $\forall cl = (\neg a_1 \vee \cdots \vee \neg a_n \vee b_1 \vee \cdots \vee b_m), \in \Sigma,$
  - $\omega \not\models cl \Rightarrow \omega \models a_1 \wedge \cdots \wedge a_n \wedge \neg b_1 \wedge \cdots \wedge \neg b_m \Rightarrow \{a_1, \ldots, a_n, b'_1, \ldots, b'_m\} \subseteq X$. But, $false \notin X$, so $X \not\models P(cl)$.
  - $X \not\models P(cl) \Rightarrow \{a_1, \ldots, a_n, b'_1, \ldots, b'_m\} \subseteq X \wedge false \notin X \Rightarrow \omega \models a_1 \wedge \cdots \wedge a_n \wedge \neg b_1 \wedge \cdots \wedge \neg b_m \Rightarrow \omega \not\models cl.$

  Thus, we have $\omega \not\models cl \iff X \not\models P(cl)$ and since $\forall (cl, \alpha) \in \Sigma, n(P(cl)) = \alpha$, we have $\pi_\Sigma(\omega) = \tilde{\pi}_P(X), \forall X \in \mathscr{X}$ and its corresponding interpretation ω.

Proof (**Proposition 16**)

1. $InconsDeg(PP(\Sigma)) = 1 - \max_{X \in 2^{\mathscr{X}}} \{\tilde{\pi}_{PP(\Sigma)}(X)\}$. If X is not a pseudo-interpretation, $\tilde{\pi}_{PP(\Sigma)}(X) = 0$ by proposition 15. So, we can restrict the application of max to the set of pseudo-interpretations, and then (again by proposition 15) we have:
$$InconsDeg(PP(\Sigma)) = 1 - \max_{\omega \in \Omega} \{\pi_\Sigma(\omega)\} = Inc(\Sigma)$$
2. Since the translation is based on a given atom set $\mathcal{A}$, $PP(\Sigma_{>\alpha}) = (PP(\Sigma))_{>\alpha}$ is obvious.
3. If $InconsDeg(PP(\Sigma)) = 0$, then the proposition 13 ensures that $(PP(\Sigma))^*$ has at least one stable model. Every stable model corresponds to a model of Σ^* because

the translation exposed in the beginning of this subsection establishes a one to one correspondence between stable models of $PP(\Sigma)$ and propositional models of Σ^*.

4. $InconsDeg(PP(\Sigma)) = \alpha \Rightarrow Inc(\Sigma) = \alpha$ by the first item of this proposition and $\Sigma_{>\alpha}$ is consistent by a possibilistic logic result. So, $Inc(\Sigma_{>\alpha}) = 0$ and by the first item of this proposition we have $InconsDeg(PP(\Sigma_{>\alpha})) = 0$. Thus, by the previous item, $(PP(\Sigma)_{>\alpha})^*$ has at least one stable model and every stable model corresponds to a propositional model of $(\Sigma_{>\alpha})^*$.

References

1. Alsinet, T., Godo, L.: A complete calculus for possibilistic logic programming with fuzzy propositional variables.In: Boutilier, C., Goldszmidt, M. (eds.), Conference in Uncertainty in Artificial Intelligence, pp. 1–10. Stanford University, Stanford, California, USA. Morgan Kaufmann, San Francisco, California (2000)
2. Anger, C., Gebser, M., Linke, T., Neumann, A., Schaub, T.: The `nomore++` system. In: Baral, C., Greco, G., Leone, N., Terracina, G. (eds.), Proceedings of the 8th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'05), volume 3662 of Lecture Notes in Artificial Intelligence, pp. 422–426. Springer, Berlin Heidelberg New York (2005)
3. Baral, C., Gelfond, M.: Logic programming and knowledge representation. J. Log. Program. **19/20**, 73–148 (1994)
4. Baral, C., Gelfond, M., Rushton, N.J.: Probabilistic reasoning with answer sets. In: Lifschitz, V., Niemelä, I., (eds.), Proceedings of the 7th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'04), volume 2923 of *LNCS*, pp. 21–33. Fort Lauderdale, FL, USA. Springer, Berlin Heidelberg New York (2004)
5. Benferhat, S., Dubois, D., Prade, H.: Representing default rules in possibilistic logic. In: International Conference on the Principles of Knowledge Representation and Reasoning, pp. 673–684 (1992)
6. Benferhat, S., Dubois, D., Prade, H.: Possibilistic logic: From nonmonotonicity to logic programming. In: Clarke, M., Kruse, R., Moral, S. (eds.), European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty volume 747 of *LNCS*, pp. 17–24. Springer, Berlin Heidelberg New York (1993)
7. Bidoit, N., Froidevaux, C.: General logical databases and programs: Default logic semantics and stratification. Inf. Comput. **91**(1), 15–54 (1991)
8. Brewka, G.: Preferred subtheories: An extended logical framework for default reasoning. In: International Joint Conference on Artificial Intelligence, pp. 1043–1048 (1989)
9. Brewka, G.: Complex preferences for answer set optimization. In: Dubois, D., Welty, C.A., Williams, M.A. (eds.), International Conference on the Principles of Knowledge Representation and Reasoning, pp. 213–223 (2004)
10. Brewka, G., Eiter, T.: Preferred answer sets for extended logic programs. Artif. Intell. **109** (1-2), 297–356 (1999)
11. Chatalic, P., Froidevaux, C., Schwind, C.: Graded hypothesis theories. Theor. Comp. Sci. **171** (1-2), 247–280 (1997)
12. Damasio, C.V., Pereira, L.M.: Antitonic logic programs. In: Eiter, T. Faber, W., Truszczynski, M. (eds.), International Conference on Logic Programming and NonMonotonic Reasoning, volume 2173 of LNCS, pp. 379–392. Springer, Berlin Heidelberg New York (2001)
13. Damasio, C.V., Pereira, L.M.: Monotonic and residuated logic programs. In: Benferhat, S., Besnard, P. (eds.), European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty, volume 2143 of LNCS, pp. 748–759. Springer, Berlin Heidelberg New York (2001)
14. Dupin de Saint-Cyr, F., Prade, H.: Describing evolving systems by uncertain default transition rules. In: International Conference in Advances in Intelligent Systems – Theory and Applications (AISTA'04), Luxembourg (2004)
15. Delgrande, J., Schaub, T., Tompits, H., Wang, K.: A classification and survey of preference handling approaches in nonmonotonic reasoning. Comput. Intell. **20**(2), 308–334 (2004)

16. Dowling, W., Gallier, J.: Linear-time algorithms for testing the satisfiability of propositional horn formulae. J. Log. Program. **1**, 267–284 (1984)
17. Dubois, D., Lang, J., Prade, H.: Towards possibilistic logic programming. In: Furukawa, K. (ed.), International Conference on Logic Programming, pp. 581–595. MIT Press, Cambridge, Massachusetts (1991)
18. Dubois, D., Lang, J., Prade, H.: Possibilistic logic. In: Gabbay, D., Hogger, C., Robinson, J. (eds.), Handbook of Logic in Artificial Intelligence and Logic Programming, vol. 3, pp. 439–513. Oxford University Press, Oxford (1995)
19. Dubois, D., Prade, H.: Possibility Theory – An Approach to Computerized Processing of Uncertainty. Plenum Press, New-York (1988)
20. Dubois, D., Prade, H.: Possibility theory: qualitative and quantitative aspects. In: Smets, Ph. (ed.), Handbook of Defeasible Reasoning and Uncertainty Management Systems, vol. 1, pp. 169–226. Kluwer Academic Press, Dordrecht (1998)
21. Dubois, D., Prade, H.: Possibility theory, probability theory and multiple-valued logics: A clarification. Ann. Math. Artif. Intell. **32**, 35–66 (2001)
22. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming . In: Kowalski, R.A., Bowen, K. (eds.), International Conference on Logic Programming, pp. 1070–1080. The MIT Press, Cambridge, Massacusetts (1988)
23. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Gener. Comput., **9**(3-4), 363–385 (1991)
24. Jampel, M., Freuder, E.C., Maher, M.J. (eds.), Over-Constrained Systems, volume 1106 of LNCS. Springer, Berlin Heidelberg New York (1996)
25. Kraus, S., Lehmann, D., Magidor, M.: Nonmonotonic reasoning, preferential models and cumulative logics. Artif. Intell. **44**(1-2), 167–208 (1990)
26. Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., Scarcello, F.: The dlv system for knowledge representation and reasoning. ACM Trans. Comput. Log. **7**(3), (July 2006)
27. Lierler, Y., Maratea, M.: Cmodels-2: Sat-based answer set solver enhanced to non-tight programs. In: International Conference on Logic Programming and NonMonotonic Reasoning, volume 2923 of LNCS, pp. 346–350. Springer, Berlin Heidelberg New York (2004)
28. Lloyd, J.: Foundations of Logic Programming. Symbolic Computation, 2nd edition. Springer, Berlin Heidelberg New York (1987)
29. Loyer, Y., Straccia, U.: Default knowledge in logic programs with uncertainty. In: International Conference on Logic Programming, volume 2916 of LNCS, pp. 466–480, Mumbai, India. Springer, Berlin Heidelberg New York (2003)
30. Lukasiewicz, T.: Many-valued disjunctive logic programs with probabilistic semantics. In: Gelfond, M., Leone, N., Pfeifer, G. (eds.), Proceedings of the 5th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'99), volume 1730 of LNCS, pp. 277–289, El Paso, Texas, USA. Springer, Berlin Heidelberg New York (1999)
31. Lukasiewicz, T.: Fixpoint characterizations for many-valued disjunctive logic programs with probabilistic semantics. In: Eiter, T., Faber, W., Truszczynski, M. (eds.), Proceedings of the 6th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'01), volume 2173 of LNCS, pp. 336–350, Vienna, Austria. Springer, Berlin Heidelberg New York (2001)
32. Lukasiewicz, T.: Weak nonmonotonic probabilistic logics. Artif. Intell. **168**(1-2), 119–161 (2005)
33. Marek, W., Truszczyński, M.: Autoepistemic logic. J. ACM **38**(3), 588–619 (1991)
34. Mateis, C.: Extending disjunctive logic programming by t-norms. In: Gelfond, M., Leone, N., Pfeifer, G. (eds.), LPNMR, volume 1730 of LNCS, pp. 290–304. Springer, Berlin Heidelberg New York (1999)
35. Nerode, A., Remmel, J.B., Subrahmanian, V.S.: Annotated nonmonotonic rule systems. Theor. Comp. Sci. **171**(1-2), 111–146 (1997)
36. Nicolas, P., Garcia, L., Stéphan, I.: A possibilistic inconsistency handling in answer set programming. In: European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty, volume 3571 of LNCS, pp. 402–414, Barcelona, Spain. Springer, Berlin Heidelberg New York (2005)
37. Nicolas, P., Garcia, L., Stéphan, I.: Possibilistic stable models. In: International Joint Conference on Artificial Intelligence, pp. 248–253, Edinburgh, Scotland, August 2–5 (2005)
38. Nicolas, P., Lefèvre, C.: Possibilistic stable model computing. In: ASP'05 workshop, Bath, England (2005)
39. Niemelä, I.: Logic programs with stable model semantics as a constraint programming paradigm. Ann. Math. Artif. Intell. **25**(3-4), 241–273 (1999)

40. Pearl, J.: System Z: A natural ordering of defaults with tractable applications to default reasoning. In: Parikh, R., (ed.), Proceedings of Theoretical Aspects of Reasoning about Knowledge, pp. 121–135. Morgan Kaufmann Publishers, San Mateo (1990)
41. Reiter, R.: A logic for default reasoning. Artif. Intell. **13**(1-2), 81–132 (1980)
42. Saad, E., Pontelli, E.: Hybrid probabilistic logic programs with non-monotonic negation. In: Logic Programming, 21st International Conference, (ICLP'05), volume 3668 of LNCS, pp. 204–220, Sitges, Spain. Springer, Berlin Heidelberg New York (2005)
43. Schwind, C.: A tableaux-based theorem prover for a decidable subset of default logic. In Stickel, M. (ed.), Conference on Automated Deduction. Springer, Berlin Heidelberg New York (1990)
44. Simons, P.: Extending and implementing the stable model semantics. Research Report A58, Helsinki University of Technology, Department of Computer Science and Engineering, Laboratory for Theoretical Computer Science, Espoo, Finland, April 2000. Doctoral dissertation
45. Syrjänen, T., Niemelä, I.: The Smodels systems. In: International Conference on Logic Programming and NonMonotonic Reasoning, pp. 434–438, Vienna, Austria, September. Springer-Verlag, Berlin Heidelberg New York (2001)
46. Wagner, G.: A logical reconstruction of fuzzy inference in databases and logic programs. In: Intelligent Fuzzy Set Association World Congress, Prague, Czech (1997)
47. Wilson, M., Borning, A.: Hierarchical constraint logic programming. J. Log. Program. **16**(3), 277–318 (1993)
48. Yager, R.R.: An introduction to applications of possibility theory. Human Syst. Manag. **3**, 246–269 (1983)
49. Zadeh, L.A.: Fuzzy sets as a basis for a theory of possibility. In: Fuzzy Sets and Systems, vol. 1, pp. 3–28. Elsevier Science, Amsterdam (1978)