A Possibilistic Inconsistency Handling in Answer Set Programming

Pascal Nicolas, Laurent Garcia, and Igor Stéphan

LERIA, University of Angers, France
{pascal.nicolas, laurent.garcia, igor.stephan}@univ-angers.fr

Abstract. Both in classical logic and in Answer Set Programming, inconsistency is characterized by non existence of a model. Whereas every formula is a theorem for inconsistent set of formulas, an inconsistent program has no answer. Even if these two results seem opposite, they share the same drawback: the knowledge base is useless since one can not draw valid conclusions from it. Possibilistic logic is a logic of uncertainty able to deal with inconsistency in classical logic. By putting on every formula a degree of certainty, it defines a way to compute, with regard to these degrees, a consistent subset of formulas that can be then used in a classical inference process. In this work, we address the treatment of inconsistency in Answer Set Programming by a possibilistic approach that takes into account the non monotonic aspect of the framework.

1 Introduction

Answer Set Programming (ASP) [1] is an appropriate formalism to represent various problems issued from Artificial Intelligence and arising when available knowledge is incomplete as in non monotonic reasoning, planning, diagnosis... In ASP, knowledge is encoded by logical rules and solutions are obtained as models. Each model is a minimal set of atoms containing some facts and deductions obtained by applying by default some rules. So, conclusions rely on present and absent data, 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 or none. When there is no answer set, the program is said to be inconsistent and it is not possible to reason with it. But, as far as we know, there is no work in ASP that deals with inconsistent programs.

Possibilistic logic [2] is issued from Zadeh's possibility theory [3]. It 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 [4] 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 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. In other words, it deals with uncertainty by means of classical 2-valued interpretations that can be more or less certain.

L. Godo (Ed.): ECSQARU 2005, LNAI 3571, pp. 402–414, 2005.

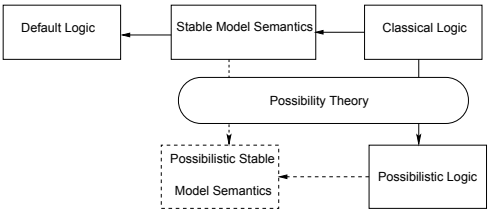

Fig. 1. Possibilistic Stable Model Semantics

In [5], we have defined *Possibilistic Stable Model Semantics* that is a new framework dealing with uncertainty in ASP. It has been developed to handle normal logic programs in which each rule is given with a certainty degree and it is based on the introduction into ASP of possibility theory concepts. Figure 1 positions our work within the linked formalisms.

In classical logic, inconsistency is characterized by the fact that every formula is a theorem. Conversely, in ASP inconsistency is characterized by the non existence of any answer. Even, if these two situations seem opposite, they share the same drawback: the knowledge base is useless since one can not draw valid conclusions from it. For our present work, the important point is that possibilistic logic can be used to handle the problem of inconsistency in classical logic bases. The certainty degrees attached to the formulas are seen as a rank-ordering of the classical base. This rank-ordering is then used to extract a consistent subbase of the initial inconsistent base. Thus it seems natural to us to developed a similar methodology with possibilistic normal logic programs to handle inconsistency in ASP. The idea is to extract, as it is done in possibilistic logic for classical bases, a consistent subprogram from which it is possible to compute some answers.

Next section 2 recalls some theoretical backgrounds about ASP and presents possibilistic stable model semantics [5]. In section 3 we expose our methodology to deal with inconsistency in ASP by a possibilistic approach. We show an equivalence result between our approach and this of possibilistic logic and we illustrate the use of our proposal in the context of combinatorial problems encoded in ASP. We conclude, in section 4, by briefly positioning our work relatively to others in the same area.

2 Theoretical Backgrounds

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* [6]. We consider given a non empty set of atoms $\mathcal{X}$ that determines the language of the programs. A *normal logic program* is a set of rules of the form: $c \leftarrow a_1, \ \ldots, \ a_k, \ not \ b_1, \ \ldots, \ not \ b_l.$ where $k \geq 0$, $l \geq 0$, $\{a_1, \ldots, a_k, b_1, \ldots, b_l, c\} \subseteq \mathcal{X}$. A term 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 no b_j's

then you can conclude c". For such a rule r we use the following notations[1]: 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, $head(r) = c$ and the positive projection of r, $r^+ = head(r) \leftarrow body^+(r)$. If a program P does not contain any default negation (ie: $body^-(P) = \emptyset$), then P is a *definite logic program* and it has one minimal Herbrand model $Cn(P)$ (see [7]). The *reduct* P^X of a program P wrt. 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* that is an atom set satisfying $S = Cn(P^S)$.

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*. Since we restrict our attention to normal logic programs we are not concerned by possibly inconsistent stable models. These ones may appear when we consider *Answer Set Semantics* for *extended logic programs* [1] in which literals (and not only atoms) are used in the rules.

Example 1. $P_1 = \{a \leftarrow ., c \leftarrow a, not\ b., d \leftarrow not\ c, not\ f.\}$ has one stable model $\{a, c\}$, $P_2 = \{a \leftarrow not\ b., b \leftarrow not\ a., c \leftarrow a., c \leftarrow b.\}$ has two stable models $\{a, c\}$ and $\{b, c\}$ and $P_3 = \{a \leftarrow not\ a.\}$ is inconsistent.

In this paper, we work with programs and their models using a "rule-based" approach that needs the following materials. Let A be an atom set, r be a rule and P be a program (definite or normal). We say that r is *applicable* in A if $body^+(r) \subseteq A$ and we denote $App(P, A)$ the subset of P of its applicable rules in A. A satisfies r (or r is satisfied by A), denoted $A \models r$, if when r is applicable in A, then $head(r) \in A$. P is *grounded* if it can be ordered as the sequence $\langle r_1, \ldots, r_n \rangle$ such that $\forall i, 1 \leq i \leq n, r_i \in App(P, head(\{r_1, \ldots, r_{i-1}\}))$.

In [5] we have extended stable model semantics in order to take into account some certainty degrees on rules. To achieve this goal, we consider given a finite set of atoms $\mathcal{X}$ and a finite, totally ordered set of necessity values $\mathcal{N} \subseteq]0, 1]$. Then, a *possibilistic atom* is a pair $p = (x, \alpha) \in \mathcal{X} \times \mathcal{N}$. We denote by $p^* = x$ the classical projection of p and by $n(p) = \alpha$ its necessity degree. These notations can also be extended to a *possibilistic atom set* (p.a.s.) A that is a set of possibilistic atoms in which every atom x occurs at most one time. A *possibilistic definite logic program* (p.d.l.p.) is a set of *possibilistic rules* of the form:

$$(c \leftarrow a_1, \ \ldots, \ a_k. \ , \ \alpha)$$

where $k \geq 0, \{a_1, \ \ldots, \ a_k, \ c\} \subseteq \mathcal{X}, \alpha \in \mathcal{N}$. The *classical projection* of a possibilistic r rule is $r^* = c \leftarrow a_1, \ \ldots, \ a_k.$ $n(r) = \alpha$ is a necessity degree representing the certainty level of the information described by r. If R is a set of possibilistic rules, then $R^* = \{r^* \mid r \in R\}$ is the program obtained from R by forgetting all the necessity values.

From a possibilistic definite logic program P, we can determine, as it is done in possibilistic logic, a least specific possibility distribution defined on all the sets in $2^{\mathcal{X}}$ and characterized as follows.

[1] These functions are extended to a rule set as usual.

Proposition 1. *Let P be a p.d.l.p. then $\pi_P : 2^{\mathcal{X}} \rightarrow [0,1]$ defined by $\forall A \in 2^{\mathcal{X}}$*

1. *if $A \not\subseteq head(App(P^*, A))$ then $\pi_P(A) = 0$*
2. *if $App(P^*, A)$ is not grounded then $\pi_P(A) = 0$*
3. *otherwise*
   - *if $\forall r \in P, A \models r^*$ then $\pi_P(A) = 1$*
   - *else $0 \leq \pi_P(A) < 1$ and $\pi_P(A) = 1 - \max_{r \in P}\{n(r) \mid A \not\models r^*\}$*

is the least specific possibility distribution.

By this way, we are able to rank every atom set with respect to its ability to be a model of P. If an atom set satisfies the condition in one of the two first items in proposition 1, then it is absolutely not possible for it to be a model of P. Otherwise, its possibility to be a model is related to the certainty degree of rules that it falsifies. Obviously, if A is the model of P, then its possibility is full and vice-versa (ie: $\pi_P(A) = 1 \iff A = Cn(P)$).

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

Definition 1. *Let P be a p.d.l.p. and π_P the least specific possibility distribution compatible with P, we define the two dual possibility and necessity measures:*

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

$\Pi_P(x)$ gives the level of consistency of x with respect to the p.d.l.p. P and $N_P(x)$ evaluates the level at which x is inferred from the p.d.l.p. 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. Furthermore, the necessity measure allows us to introduce the following definition of a *possibilistic model* of a p.d.l.p.

Definition 2. *Let P be a possibilistic definite logic program, then the set*

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

is its possibilistic model.

Proposition 2. *Let P be a p.d.l.p. then : $(\Pi\mathcal{M}(P))^*$ is the model of P^*.*

Example 2. Let us take $P = \{(a \leftarrow ., 0.8),\ (b \leftarrow a., 0.6),\ (d \leftarrow a., 0.5),\ (d \leftarrow c., 0.9)\}$. The least specific possibility distribution induced by P is null for every atom set included in $\{a, b, c, d\}$ except for $\pi_P(\emptyset) = 0.2$, $\pi_P(\{a\}) = 0.4$, $\pi_P(\{a, b\}) = 0.5$, $\pi_P(\{a, d\}) = 0.4$ and $\pi_P(\{a, b, d\}) = 1$ (the model). So, we can compute the possibility of each atom: $\Pi_P(a) = \Pi_P(b) = \Pi_P(d) = 1$ and $\Pi_P(c) = 0$ and its certainty in term of necessity degree: $N_P(a) = 0.8$, $N_P(b) = 0.6$, $N_P(c) = 0$, $N_P(d) = 0.5$. Thus, $\Pi\mathcal{M}(P) = \{(a, 0.8), (b, 0.6), (d, 0.5)\}$ is the possibilistic model of P.

Now, we allow default negation in programs and we summarize the notion of *possibilistic stable model*. It extends the stable model semantics by taking into account the necessity degree in the rules of a given *possibilistic normal logic program* (p.n.l.p.). Such a program is a finite set of rules of the form:

$$(c \leftarrow a_1,\ \ldots,\ a_k,\ not\ b_1,\ \ldots,\ not\ b_l.\ ,\ \alpha)\ k \geq 0, l \geq 0$$

for which we just have to precise that $\{b_1, \ldots, b_l\} \subseteq \mathcal{X}$, all the rest being the same as for a p.d.l.p. (see the beginning of this section). As for normal logic programs, we need to define what is the reduction of a program.

Definition 3. *Let P be a p.n.l.p. and A be an atom set. The* possibilistic reduct *of P wrt. A is the p.d.l.p. $P^A = \{(r^{*+}, n(r)) \mid r \in P, body^-(r) \cap A = \emptyset\}$.*

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

Definition 4. *Let P be a p.n.l.p. and S a p.a.s., S is a possibilistic stable model of P if and only if $S = \Pi\mathcal{M}(P^{S^*})$.*

By analogy with normal logic programs we say that a p.n.l.p. P is *consistent* if P has at least one possibilistic stable model. Otherwise P is said to be *inconsistent*. Let us note that there is a one to one mapping between the stable models of P^* and the p.s.m. of P. In particular, if P is inconsistent then P^* is also inconsistent and if S is a p.s.m. of P then S^* is a stable model of P^*.

Example 3. $\{(c \leftarrow a, not\ d., 0.6), (d \leftarrow a, not\ c., 1), (a., 0.8), (e \leftarrow d., 0.3)\}$ has two possibilistic stable models: $\{(a, 0.8), (c, 0.6)\}$ and $\{(a, 0.8), (d, 0.8), (e, 0.3)\}$.

Now, we examine the semantics given to this framework by the definition of a possibility distribution induced by the necessity values associated to each normal rules. This distribution $\tilde{\pi}$ over all the atom sets (ie : over $2^{\mathcal{X}}$) has to reflect the ability of every atom set to be a stable model of P^* and that is formalized in the proposition 3.

Definition 5. *Let P be a p.n.l.p. and A be an atom set, then $\tilde{\pi}_P$ is the possibility distribution defined by $\tilde{\pi}_P : 2^{\mathcal{X}} \rightarrow [0, 1]$ such that $\forall A \in 2^{\mathcal{X}}, \tilde{\pi}_P(A) = \pi_{P^A}(A)$.*

Proposition 3. *Let P be a p.n.l.p. and $A \in 2^{\mathcal{X}}$ be an atom set, then $\tilde{\pi}_P(A) = 1 \iff A$ is a stable model of P^*.*

Thus, if there is no atom set A such that $\tilde{\pi}_P(A) = 1$ then P is inconsistent. This ends the presentation of possibilistic stable model semantics that we have introduced in previous works to manage uncertainty in ASP and that we use here to deal with inconsistency.

3 Inconsistency Handling in ASP

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. The basic idea is to consider that every rule in the given program has a certainty degree. All rules are ranked by strata with respect to these degrees and the consistency restoring process has to keep the greatest number of most preferred strata.

3.1 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, obtained by forgetting the weights, 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. α-cut (resp. strict α-cut) of Σ, denoted by $\Sigma_{\geq\alpha}$ (resp. by $\Sigma_{>\alpha}$), is the set of classical formulas in Σ having a certainty degree greater than (resp. strictly greater than) α. The inconsistency degree of Σ is $Inc(\Sigma) = max\{\alpha : \Sigma_{\geq\alpha}\text{is inconsistent}\}$. $Inc(\Sigma) = 0$ means that Σ is consistent. If Σ has no model, then, by discarding formulas which 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.

The following presentation of our work is inspired by these general principles issued from possibilistic logic.

Definition 6. *Let P be a p.n.l.p.*

- *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*

$$ConsCutDeg(P) = \begin{cases} 0 \text{ if } P \text{ is consistent} \\ 1 \text{ if } \forall \alpha \in \mathcal{N}, P_{>\alpha} \text{ is inconsistent} \\ \min_{\alpha \in \mathcal{N}}\{P_{>\alpha} \text{ is consistent}\} \text{ otherwise} \end{cases}$$

The consistency cut degree of a p.n.l.p. 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 p.n.l.p. is to delete the minimum number of less certain rules.

Example 4. Let $P = \left\{ \begin{array}{l} (c \leftarrow ., 1), (f \leftarrow not\ e, not\ f.0.9), (e \leftarrow not\ b., 0.8), \\ (a \leftarrow not\ a, not\ b, 0.7), (d \leftarrow c, not\ d., 0.6), (b \leftarrow c., 0.5) \end{array} \right\}$
$ConsCutDeg(P) = 0.7$ since $P(= P_{>0}), P_{>0.5}$ 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.

Definition 7. *Let P be a p.n.l.p., 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 p.n.l.p. and to define a cut of an inconsistent p.n.l.p. that is still a superset of the consistent subprogram that we want to obtain.

Proposition 4. *Let P be a p.n.l.p., then*

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

We define our methodology of consistency restoration for a p.n.l.p. by means of the next function *cut* that computes the greatest (wrt. the certainty level of rules) consistent subprogram of P.

Definition 8. *Let cut be the function defined on a p.n.l.p by*

$$\left\{ \begin{array}{ll} cut(P) = P & \text{if } InconsDeg(P) = 0 \\ cut(P) = cut(P_{>InconsDeg(P)}) & \text{otherwise} \end{array} \right.$$

Proposition 5. *Let P be a p.n.l.p. then $cut(P) = P_{>ConsCutDeg(P)}$.*

Example 5. Let us come back to our program P in example 4 for which we have $InconsDeg(P) = 0.7$. The first call to *cut* is enough to compute the maximal consistent subprogram of P: $cut(P) = \{(c., 1), (f \leftarrow not\ e, not\ f.0.9), (e \leftarrow not\ b., 0.8)\}$ such that $cut(P)^*$ has one stable model $\{c, e\}$.

3.2 Relations with Possibilistic Logic

In this section, we focus our attention on possibilistic normal logic programs encoding classical possibilistic bases. Let $\mathcal{A}$ be an atom set from which a classical propositional base is built. Recall that every propositional base Σ can be encoded in a clause set. So, without loss of generality, we consider here only clause sets. On its turn, such a clause set Σ can be translated in a normal logic program $P(\Sigma)$ as following (a similar process is exposed in [8]). 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 $P(cl) = f \leftarrow a_1, \ldots, a_n, b'_1, \ldots, b'_m$. The encoding of a base Σ is

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

and the intuition behind this translation stands on the following remarks.

- 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 f (a new symbol for *false*) 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 f, then it corresponds to an interpretation of Σ (since every clause is satisfied).
- The goal of special rule $bug \leftarrow f, not\ bug.$, where bug is a new symbol, is to discard every stable model containing f. 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 f.

By this way there is a one to one correspondence between the propositional models of Σ and the stable models of $P(\Sigma)$. But, as stated in [9] 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 keep the same necessity degree α in $PP(\Sigma)$. A necessity value equal to 1 is assigned to all the other rules (the "technical" ones).

Definition 9. *Let $\Sigma = \{(cl_i, \alpha_i), i = 1, \ldots, n\}$ be a possibilistic base (in CNF), its encoding in a p.n.l.p. is:*

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

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

Definition 10. *$X \subseteq \mathcal{X}$ is a* pseudo interpretation *if*

$$\forall a \in \mathcal{A}, (a \in X \vee a' \in X) \wedge (a \notin X \vee a' \notin X) \wedge\ bug \notin X \wedge f \notin X$$

The interesting point for p.n.l.p. encoding a possibilistic logic base is that, in this case, we are able to restore the consistency of a p.n.l.p. 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 $\neg a$,

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 $PP(\Sigma)$		

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

we obtain the interpretation[2] ω. By this way, every stable model of $PP(\Sigma)^*$ is a pseudo interpretation corresponding to a classical model for Σ and conversely.

Proposition 6. *Let Σ be a possibilistic base and $P = PP(\Sigma)$ its encoding in a p.n.l.p., $\forall X \subseteq \mathcal{X}$ we have*

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

Proposition 7. *Let Σ be a possibilistic base, then*

- $Inc(\Sigma) = InconsDeg(PP(\Sigma))$.
- *if* $Inc(\Sigma) = \alpha$, $PP(\Sigma_{>\alpha}) = (PP(\Sigma))_{>\alpha}$
- $InconsDeg(PP(\Sigma)) = 0 \Longrightarrow (PP(\Sigma))^*$ *has at least one stable model S that corresponds to a propositional model of Σ*
- $InconsDeg(PP(\Sigma)) = \alpha > 0 \Longrightarrow (PP(\Sigma)_{>\alpha})^*$ *has at least one stable model S that 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 management of classical bases with possibilistic logic and the management of the corresponding p.n.l.p

Example 6. Let $\Sigma = \left\{ \begin{array}{l} (\neg e, 0.9), (b \vee c, 0.8), (\neg b \vee e, 0.7), (\neg a \vee b, 0.7), \\ (\neg d, 0.5), (a, 0.5), (\neg b \vee d, 0.3) \end{array} \right\}$ be a possibilistic base. Its encoding as a p.n.l.p. is

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

[2] A pseudo interpretation leads necessary to an interpretation since it contains one occurrence of each atom (ie a or its negation) and no occurrence of f nor *bug*.

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

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

So, 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.

3.3 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. 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 can not 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 p.n.l.p $PP(A)$. 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 the following example of a 2-coloration of a graph.

Example 7. Let us consider A, the problem of coloring, by *red* or *green* the undirected graph $G = (\{v1, v2, v3\}, \{(v1, v2), (v2, v3), (v3, v1)\})$. Its encoding is

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

Fig. 3. Constraint relaxation

But, $P(A)$ is inconsistent since it is obvious that it is impossible to color G with only two colors in such a way that two connected vertices have different colors. 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 first graph of figure 3. The corresponding possibilistic normal logic program[3] that encodes this additional information is:

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

Then, $InconsDeg(PP(A)) = 0.7$ and $cut(PP(A)) = PP(A)_{>0.7}$ is a consistent p.n.l.p. This subprogram $PP(A)$ encodes a relaxation of the initial problem A in which we eliminated the less important constraint as illustrated in the second graph of figure 3. Finally, the stable models, $\{red(1), green(2), green(3)\}$ and $\{red(1), green(2), green(3)\}$, of $cut(PP(A))^*$ encode some approximate solutions of the initial problem A.

Our proposal deals with over-constrained logic programs for which other works exist as Hierarchical Constraint Logic Programming [10]. 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 [11] for a survey on over-constrained systems).

4 Conclusion

In this work, we have proposed a methodology to restore the consistency of a normal logic program. Our proposal is underpinned by possibilistic stable model semantics that allows to rank the rules of a program by order of certainty or importance. We have defined a *cut* function that returns a consistent subprogram of the initial inconsistent one. We have shown that our approach is equivalent to

[3] 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.

that in possibilistic logic and illustrated how it can be used to relax a program encoding a combinatorial problem. This is useful in order to find an approach solution when the initial given problem has no solution.

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 [12] 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 8. 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 [12] 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 p.n.l.p. $PP = \{(b \leftarrow not\ a., 1), (a \leftarrow not\ a., 0.8), (a \leftarrow not\ b., 0.6), (b \leftarrow not\ b., 0.4)\}$. Then, we find $IncDeg(PP) = 0.4$ 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 $\{a\}$.

For an inconsistent logic base Σ dealt with a possibilistic approach, the consistent subbase $\Sigma_{>Inc(\Sigma)}$ is always a subset of the preferred subtheories of Σ. Whereas the example 8 shows that it is not always the case for the normal logic programs. This difference comes from the non monotonic nature of stable model semantics. In future works, we envisage to apply in ASP other strategies for consistency restoring. Particularly, it would be interesting to study how to keep all rules not directly involved in the inconsistency.

References

1. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Generation Computing **9(3-4)** (1991) 363–385
2. 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. Volume 3. Oxford University Press (1995) 439–513
3. Zadeh, L.: Fuzzy sets as a basis for a theory of possibility. In: Fuzzy Sets and Systems. Volume 1. (1978) 3–28
4. Dubois, D., Prade, H.: Possibility theory: qualitative and quantitative aspects. In Smets, P., ed.: Handbook of Defeasible Reasoning and Uncertainty Management Systems. Volume 1. Kluwer Academic Press (1998) 169–226

5. Nicolas, P., Garcia, L., Stéphan, I.: Possibilistic stable models. In: International Joint Conference on Artificial Intelligence, Edinburgh, Scotland (2005)
6. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In Kowalski, R.A., Bowen, K., eds.: International Conference on Logic Programming, The MIT Press (1988) 1070–1080
7. Lloyd, J.: Foundations of Logic Programming. 2nd edn. Symbolic Computation. Springer (1987)
8. 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 (2000) Doctoral dissertation.
9. Niemelä, I.: Logic programs with stable model semantics as a constraint programming paradigm. Annals of Mathematics and Artificial Intelligence **25** (1999) 241–273
10. Wilson, M., Borning, A.: Hierarchical constraint logic programming. Journal of Logic Programming **16** (1993) 277–318
11. Jampel, M., Freuder, E.C., Maher, M.J., eds.: Over-Constrained Systems. In Jampel, M., Freuder, E.C., Maher, M.J., eds.: Over-Constrained Systems. Volume 1106 of Lecture Notes in Computer Science., Springer (1996)
12. Brewka, G.: Preferred subtheories: An extended logical framework for default reasoning. In: International Joint Conference on Artificial Intelligence. (1989) 1043–1048