Semantics for possibilistic answer set programs: uncertain rules versus rules with uncertain conclusions

Kim Bauters[a,], Steven Schockaert[b], Martine De Cock[a], Dirk Vermeir[c]

[a]*Department of Applied Mathematics and Computer Science*
Universiteit Gent, Krijgslaan 281 (S9), 9000 Gent, Belgium
[b]*School of Computer Science & Informatics, Cardiff University*
5 The Parade, Cardiff CF24 3AA, United Kingdom
[c]*Department of Computer Science*
Vrije Universiteit Brussel, Pleinlaan 2, 1050 Brussel, Belgium

Abstract

Although Answer Set Programming (ASP) is a powerful framework for declarative problem solving, it cannot in an intuitive way handle situations in which some rules are uncertain, or in which it is more important to satisfy some constraints than others. Possibilistic ASP (PASP) is a natural extension of ASP in which certainty weights are associated with each rule. In this paper we contrast two different views on interpreting the weights attached to rules. Under the first view, weights reflect the certainty with which we can conclude the head of a rule when its body is satisfied. Under the second view, weights reflect the certainty that a given rule restricts the considered epistemic states of an agent in a valid way, i.e. it is the certainty that the rule itself is correct. The first view gives rise to a set of weighted answer sets, whereas the second view gives rise to a weighted set of classical answer sets.

Keywords: logic programming, non-monotonic reasoning, possibility theory, uncertainty, stable model semantics

*Funded by a joint Research Foundation-Flanders (FWO) project
Corresponding author. Tel.: +32 9 264 49 05
Email addresses: kim.bauters@ugent.be (Kim Bauters), s.schockaert@cs.cardiff.ac.uk (Steven Schockaert), martine.decock@ugent.be (Martine De Cock), dvermeir@vub.ac.be (Dirk Vermeir)

1. Introduction

Answer set programming (ASP) is a form of logic programming which uses the stable model semantics to define negation-as-failure in a purely declarative way. An ASP program consists of a set of rules, e.g.

$$beach \leftarrow \neg weekday, not\ rain. \tag{1}$$

This rule intuitively encodes that we go to the beach when we know that today is not a weekday and when there is no indication of rain. Reasoning in ASP is non-monotonic because of the semantics of '*not*', which is called the negation-as-failure operator. In particular, '*not rain*' is true unless we have evidence that '*rain*' is true, e.g. we may read in the newspaper that rainy weather is predicted at the beach and we may consequentially need to revise our plans to go to the beach.

While ASP is well-suited for reasoning over incomplete information, it lacks the means to reason over uncertain information in a natural way. Nevertheless, uncertain information is an important and pervasive component of common-sense reasoning. For example, in (1) it may happen that we are driving an old car, in which case we may be uncertain as to whether or not we will reach the beach, even when all the premises are satisfied.

In this paper we are mainly interested in the potential of ASP as a tool for epistemic reasoning, i.e. for reasoning about what another agent believes. Under this view answer sets are interpreted as possible epistemic states of that agent. Rules are then seen as pieces of knowledge which constrain the epistemic state that an agent may have.

Answer sets, however, have a number of limitations for modeling epistemic states. One limitation is that an answer set can only express that the agent knows that a given literal is true, or that the agent has insufficient knowledge to assess the truth of the literal. An answer set cannot express in a natural way the belief of an agent that "*either literal* l_1 *or literal* l_2 *is true, but not both*". This issue has been the topic of [1, 2]. An orthogonal issue is that classical ASP cannot intuitively be used to express that knowledge may be more or less certain. This particular issue is the focus of this paper.

There are two natural ways of introducing degrees of uncertainty in the setting of ASP. On the one hand, we may wish to model weighted epistemic states, in which an agent can be completely certain about the truth of some literals, while believing in the truth of other literals without complete certainty. On the other hand, we may wish to keep epistemic states Boolean,

but rather express that the rules which constrain the possible epistemic states are not fully certain. In the latter case, an ASP program should correspond to a weighted set of classical answer sets. In this paper, we will compare and contrast both views, using possibility theory to model the semantics of both types of uncertainty degrees.

Combining ASP with possibility theory is not a new idea. In fact, two particular forms of possibilistic ASP were already investigated in [3] and [4]. Both approaches adhere to the first view, i.e. they model weighted epistemic states. The approaches do, however, differ in the way they treat negation-as-failure. In particular, the semantics from [3] consider a naf-literal of the form '*not l*' to be false as soon as there is *some* evidence that '*l*' is true. The semantics from [4], on the other hand, adhere to the intuition that '*not l*' is certain to the extent that '$\neg l$' is possible. Which semantics to use is often dependent on the context of the problem. Consider the example:

$$\begin{aligned}
\mathbf{1}&: \neg breathing \leftarrow \\
\mathbf{1}&: dead \leftarrow \neg breathing, \neg pulse \\
\mathbf{0.6}&: dead \leftarrow \neg pulse \\
\mathbf{0.2}&: dead \leftarrow \neg breathing \\
\mathbf{0.9}&: first_aid_successful \leftarrow not\ dead.
\end{aligned}$$

This example models the reasoning of the first responder arriving at the scene of an incident and faced with an unconscious victim. A quick examination shows that the victim is no longer breathing. From his experience, the first responder knows that lack of breathing does not necessarily indicate that the person is dead. However, if the victim also lacked a pulse, then this would considerably increase his certainty that the victim has died. The first responder has a strong certainty that applying first aid will be successful when there is no indication that the person is dead. Alternatively, we could replace the last rule with the rule '$\mathbf{0.9}: apply\_first\_aid \leftarrow not\ dead$'. Then, the intuition of the last rule becomes that the first responder has a high certainty that he should at least attempt to apply first aid when there is no indication that the victim is dead.

Regardless of the last rule, both the semantics from [3] and [4] agree that $\neg breathing$ is fully certain and *dead* is certain to degree 0.2. Given the original last rule, the semantics from [3] furthermore conclude that the certainty of *first_aid_successful* is 0, whereas the semantics from [4] conclude that *first_aid_successful* is certain to degree 0.8. The conclusion that we are

certain of *first_aid_successful* to degree 0 is sound since *dead* is certain to degree 0.2 while the certainty of $\neg$*dead* is 0, i.e. we are more certain that the victim is dead than that he is alive. If, however, we consider the alternative last rule, then given the approach from [3] we conclude that the certainty of *apply_first_aid* is 0 and *apply_first_aid* is certain to degree 0.8 under the approach from [4]. In this case, however, we can argue that the conclusion must be that *apply_first_aid* is certain to degree 0.8 since we only have a low certainty that the victim is dead, i.e. we entertain a high possibility that the victim is still alive. As this example illustrates, the desired answer greatly depends on the particular understanding of the problem at hand. In Section 2.3 we recall the details of both approaches.

The rules in the example above are themselves not uncertain. Rather, the rules are considered valid, but they only allow us to reach conclusions with limited certainty. Often, however, we may be uncertain whether the rules are actually meaningful, e.g. when they are coming from possibly unreliable sources. Consider this example:

$$\begin{aligned}
\mathbf{0.7}&: \mathit{paper_title}(\mathit{title}) \leftarrow \\
\mathbf{0.9}&: \mathit{author}(\mathit{John_Doe}) \leftarrow \mathit{paper_title}(\mathit{title}) \\
\mathbf{0.2}&: \mathit{author}(\mathit{Jane_Roe}) \leftarrow \mathit{paper_title}(\mathit{title}) \\
\mathbf{1}&: \leftarrow \mathit{author}(\mathit{John_Doe}), \mathit{author}(\mathit{Jane_Roe}).
\end{aligned}$$

During a conference, a colleague shares the title of an interesting paper with us. We are quite certain that we recall the name of the title correctly and we would like to find out who the principal author of the paper is. We consult the university website, which in the past has given reliable answers. However, a quick search on the internet results in a different name for the same paper. Evidently, they cannot both be the principal author of the paper.

The uncertainty attached to each rule now expresses how certain we are that the information encoded in the rule is indeed valid. In particular, any world in which both John Doe and Jane Roe are the principal author of the paper can immediately be discarded due to the absolute certainty of the last rule. We do, however, acknowledge that neither of the candidate authors may be correct because we do not have absolute certainty as to whether we correctly recall the title of the paper. Of the two remaining rules, we have the most confidence in the rule that identifies John Doe as the author. Thus, we expect that the conclusion that the actual principal author of the paper is John Doe can be deduced with a higher certainty that the conclusion

that the principal author is Jane Roe. The semantics that agree with this intuition, i.e. that agree with the idea of uncertain rules rather than uncertain conclusions, are discussed in Section 3. In particular, it will turn out that under the semantics from [3] and [4] the above program does not have any answer sets.

The remainder of this paper is organized as follows. In Section 2 we provide the reader with some important notions from answer set programming, possibilistic logic and current work on possibilistic answer set programs, in which weights refer to the certainty with which the conclusions of rules can be derived, given that their body is known to hold. In Section 3 we take another view on the meaning of the weights in PASP, where we treat weights as an indication of whether or not the information encoded in the rules is valid. In Section 4 we show how the new semantics based on the intuition of uncertain rules can be simulated using classical ASP. A special case of the semantics presented in Section 3 are programs with optional rules. We show in Section 5 how some important problems in AI can be expressed in terms of programs with such optional rules. Related work is discussed in Section 6 and we formulate our conclusions in Section 7.

This paper extends parts of our work from [5]. In particular, the work is extended to also cover literals (rather than just considering atoms) and we extend the semantics to cover disjunctive programs. Additionally, we provide simulations using classical ASP for all reasoning tasks presented in [5]. Complexity results are now provided for all reasoning tasks and proofs are provided in the appendix.

2. Preliminaries

We start by reviewing the definitions from both answer set programming and possibility theory that will be used in the remainder of the paper. Furthermore, we review current approaches that combine ASP and possibility theory and we present the general framework of possibilistic ASP that will be used throughout the paper. Finally, we recall some notions from complexity theory.

2.1. Answer set programming (ASP)

To define ASP programs, we start from a finite set of atoms $\mathcal{A}$. A *literal* is defined as an atom 'a' or its classical negation '$\neg a$'. For a set of literals L, we use $\neg L$ to denote the set $\{\neg l \mid l \in L\}$ where, by definition, $\neg\neg a = a$. A set

of literals is said to be *consistent* when $L \cap \neg L = \emptyset$, i.e. L does not contain two contradictory literals. The set of all literals is written as $\mathcal{L} = \mathcal{A} \cup \neg\mathcal{A}$. A *naf-literal* is either a literal 'l' or an expression of the form '*not* l', where '*not*' denotes negation-as-failure. Intuitively, we have that '*not* l' is true when we have no proof for 'l'. A *disjunctive rule* is an expression of the form

$$l_0; ...; l_k \leftarrow l_{k+1}, ..., l_m, not\ l_{m+1}, ..., not\ l_n$$

where l_i is a literal for every $0 \leq i \leq n$. We say that $l_0; ...; l_k$ is the *head* of the rule (interpreted as a disjunction) and that $l_{k+1}, ..., l_m, not\ l_{m+1}, ..., not\ l_n$ is the *body* of the rule (interpreted as a conjunction). For a given rule r we use $head(r)$ and $body(r)$ to denote the set of literals in the head and the body of the rule, respectively. Specifically, we use $body_+(r)$ to denote the set of literals in the body that are not preceded by the negation-as-failure operator '*not*' and $body_-(r)$ for those literals that are preceded by '*not*'.

Whenever a disjunctive rule does not contain negation-as-failure, i.e. when $m = n$, we say that the rule is a *positive disjunctive rule*. A rule with an empty body, i.e. a rule of the form $(l_0; ...; l_k \leftarrow)$, is called a *fact rule* and is used as a shorthand for $(l_0; ...; l_k \leftarrow \top)$ with $\top$ a special language construct denoting tautology. Conversely, a rule with an empty head, i.e. a rule of the form $(\leftarrow l_{k+1}, ..., l_m, not\ l_{m+1}, ..., not\ l_n)$ is called a *constraint rule* and is used as a shorthand for $(\bot \leftarrow l_{k+1}, ..., l_m, not\ l_{m+1}, ..., not\ l_n)$ with $\bot$ another special language construct denoting contradiction.

A *(positive) disjunctive program* P is a set of (positive) disjunctive rules. A *normal rule* is a disjunctive rule with at most one literal in the head. A *simple rule* is a normal rule without negation-as-failure. A *definite rule* is a simple rule with no classical negation, i.e. a rule in which all literals are atoms. A *normal (resp. simple, definite) program* is a set of normal (resp. simple, definite) rules.

The *Herbrand base* $\mathcal{B}_P$ of a disjunctive program P is the set of atoms appearing in P. The set of literals relevant for a disjunctive ASP program is defined as $Lit_P = (\mathcal{B}_P \cup \neg\mathcal{B}_P)$. A *partial Herbrand interpretation* I of a disjunctive program P is any set of literals $I \subseteq Lit_P$. A partial Herbrand interpretation I is said to be consistent when it does not contain both 'a' and '$\neg a$' for some $a \in I$. A partial Herbrand interpretation I is said to *satisfy* a positive disjunctive rule r if $head(r) \cap I \neq \emptyset$ or $body(r) \not\subseteq I$, i.e. the body is false or the head is true. In particular, I is said to *satisfy* a constraint rule r if $body(r) \not\subseteq I$. If for a partial Herbrand interpretation I and a constraint rule

r we have that $body(r) \subseteq I$, then we say that I *violates* the constraint rule r. Notice that for a fact rule we require that $head(r) \cap I \neq \emptyset$, i.e. at least one of the literals in the head must be true. A partial Herbrand interpretation I of a positive disjunctive program P is a *partial Herbrand model* of P either if I is consistent and I satisfies every rule $r \in P$, or if $I = Lit_P$. It follows from this definition that Lit_P is always a partial Herbrand model of P, and that all other partial Herbrand models of P (if any) are consistent partial Herbrand interpretations, which we will further on also refer to as *consistent partial Herbrand models*. We say that I is an *answer set* of the positive disjunctive program P when I is minimal among the partial Herbrand models of P w.r.t. set inclusion.

The semantics of an ASP program with negation-as-failure are based on the idea of a stable model [6]. The *reduct* P^I of a disjunctive program P w.r.t. the partial Herbrand interpretation I is defined as:

$$P^I = \{l_0; ...; l_k \leftarrow l_{k+1}, ..., l_m \mid (\{l_{m+1}, ..., l_n\} \cap I = \emptyset) \\ \text{and } (l_0; ...; l_k \leftarrow l_{k+1}, ..., l_m, not\ l_{m+1}, ..., not\ l_n) \in P\}.$$

I is said to be an *answer set* of the disjunctive program P when I is an answer set of the positive disjunctive program P^I (hence the notion of stable model). Note that we can also write the disjunctive program P as $P = P' \cup C$ where C is the set of constraint rules in P. A partial Herbrand interpretation I then is an answer set of the disjunctive program P when I is an answer set of P' and I is a model of C, i.e. I does not violate any constraints in C.

Whenever P has *consistent answer sets*, i.e. answer sets that are consistent partial Herbrand interpretations, we say that P is a *consistent program*. When P has the answer set Lit_P, then this is the unique [7] *inconsistent answer set* and we say that P is an *inconsistent program*.

Example 1. Consider the normal program P with the rules:

$$a \leftarrow \qquad b \leftarrow not\ c \qquad c \leftarrow not\ b \qquad \leftarrow c$$

The first rule is a fact rule, whereas the last rule is a constraint rule. We have that $\mathcal{B}_P = \{a, b, c\}$. Three consistent partial Herbrand interpretations of P are $\{a, b, c\}$, $\{a, b\}$ and $\{a, c\}$. Both $\{a, b, c\}$ and $\{a, c\}$ violate the constraint rule. Furthermore, $\{a, b, c\}$ is not minimal w.r.t. the other two interpretations. Only $\{a, b\}$ is a partial Herbrand model of P and it is the unique minimal partial Herbrand model, i.e. answer set, of P. Furthermore, P is a consistent program since it has consistent answer sets.

Answer sets of simple programs can also be defined in a more procedural way. By using the *immediate consequence operator* T_P, which is defined for a simple program P w.r.t. an interpretation I as:

$$T_P(I) = \{l_0 \mid (l_0 \leftarrow l_1, ..., l_m) \in P \text{ and } l_1, ..., l_m \subseteq I\}.$$

We use $fp(P)$ to denote the fixpoint which is obtained by repeatedly applying T_P starting from the empty interpretation $\emptyset$, i.e. it is the least fixpoint of T_P w.r.t. set inclusion. When the partial Herbrand interpretation $fp(P)$ is consistent, $fp(P)$ is the (unique and consistent) answer set of the simple program P without constraint rules. When we allow constraint rules, a partial Herbrand interpretation is a (consistent) answer set of $P = P' \cup C$ iff I is a (consistent) answer set of P and I is a partial Herbrand model of C. For both simple and normal programs, with or without constraint rules, we have that Lit_P is the (unique and inconsistent) answer set of P if P has no consistent answer set(s).

Finally, we use $\models^{\text{b}}$ (resp. $\models^{\text{c}}$) to denote brave (resp. cautious) inference in classical ASP, i.e. $P \models^{\text{b}} l$ iff $\exists M, l \in M \cdot M$ is an answer set of P, and $P \models^{\text{c}} l$ iff $\nexists M, l \notin M \cdot M$ is an answer set of P.

2.2. Possibilistic logic

Interpretations in ASP are different from interpretations in classical logic as they may be partial. The semantics of possibilistic logic, on the other hand, is defined w.r.t. classical interpretations. We represent such an interpretation as a set of atoms ω, where $\omega \models a$ if $a \in \omega$ and otherwise $\omega \models \neg a$, with $\models$ the satisfaction relation from classical logic. The set of all interpretations is defined as $\Omega = 2^{\mathcal{A}}$, with $\mathcal{A}$ a finite set of atoms.

At the semantic level, possibilistic logic [8] is defined in terms of a *possibility distribution* π on the universe of interpretations where a possibility distribution is a $\pi : \Omega \rightarrow [0, 1]$ mapping. A possibility distribution π encodes for each interpretation (or world) ω to what extent it is plausible that ω is the actual world. By convention, $\pi(\omega) = 0$ means that ω is impossible and $\pi(\omega) = 1$ means that no available information prevents ω from being the actual world. A possibility distribution π is said to be *normalized* if $\exists \omega \in \Omega \cdot \pi(\omega) = 1$, i.e. at least one interpretation is entirely plausible. Conversely, when $\forall \omega \in \Omega \cdot \pi(\omega) = 0$ we say that a possibility distribution π is *vacuous*. Note that possibility degrees are mainly interpreted qualitatively:

when $\pi(\omega) > \pi(\omega')$, ω is considered more plausible than ω'. For two possibility distributions π_1 and π_2 with the same domain Ω we write $\pi_1 \geq \pi_2$ when $\forall \omega \in \Omega \cdot \pi_1(\omega) \geq \pi_2(\omega)$ and $\pi_1 > \pi_2$ when $\pi_1 \geq \pi_2$ as well as $\pi_1 \neq \pi_2$.

A possibility distribution π induces two uncertainty measures that allow us to rank propositions. The *possibility measure* Π is defined by [8]:

$$\Pi(p) = \max \{\pi(\omega) \mid \omega \models p\}$$

and evaluates the extent to which a proposition p is consistent with the beliefs expressed by π. The dual *necessity measure* N is defined by:

$$N(p) = 1 - \Pi(\neg p)$$

and evaluates the extent to which a proposition p is entailed by the available beliefs [8]. Note that we always have $N(\top) = 1$ for any possibility distribution, while $\Pi(\top) = 1$ (and $N(\bot) = 0$) only holds when the possibility distribution is normalized, i.e. only normalized possibility distributions can express consistent beliefs [8]. To identify the possibility/necessity measure associated with a specific possibility distribution π_{X}, we will use a subscript notation, i.e. Π_{X} and N_{X} are the corresponding possibility and necessity measure, respectively. We omit the subscript when the possibility distribution is clear from the context.

An important property of necessity measures is their min-decomposability w.r.t. conjunction: $N(p \wedge q) = \min \{N(p), N(q)\}$ for all propositions p and q. However, for disjunction only the inequality $N(p \vee q) \geq \max \{N(p), N(q)\}$ holds. As possibility measures are the dual measures of necessity measures, they have the property of max-decomposability w.r.t. disjunction, while for the conjunction we have that only the inequality $\Pi(p \wedge q) \leq \min \{\Pi(p), \Pi(q)\}$ holds.

At the syntactic level, a *possibilistic knowledge base* consists of pairs (p, c) where p is a propositional formula and $c \in]0, 1]$ expresses the certainty that p is the case. Formulas of the form $(p, 0)$ are not explicitly represented in the knowledge base since they encode trivial information. A formula (p, c) is interpreted as the constraint $N(p) \geq c$, i.e. a possibilistic knowledge base Σ corresponds to a set of constraints on possibility distributions. Typically, there can be many possibility distributions that satisfy these constraints. In practice, we are usually only interested in the *minimally specific possibility distributions*, which are the possibility distributions that make minimal commitments, i.e. the maximal possibility distributions w.r.t. the ordering $>$. For

the constraints induced by a possibilistic logic base, there is a unique minimally specific distribution, which is called the least specific distribution [8].

2.3. Possibilistic answer set programming (PASP)

Possibilistic ASP (PASP) combines ASP and possibility theory by associating a weight with each rule. Such a weight may be interpreted in different ways. In [3, 4] this weight is the necessity with which the head of the rule can be concluded, given that the body is known to hold. If it is uncertain whether the body holds, the necessity with which the head can be derived is the minimum of the weight associated with the rule and the degree to which the body is necessarily true. Another interpretation of the weights associated with rules will be introduced in Section 3.

We use the name PASP for a family of approaches that share a common syntax and which all rely on possibility theory and ASP. Syntactically, a possibilistic disjunctive (resp. normal, simple, definite) program is a set of pairs $p = (r, c)$ with r a disjunctive (resp. normal, simple, definite) rule and $c \in]0, 1]$ a certainty associated with r. We will also write a pair $p = (r, c)$ with r a disjunctive rule of the form $(l_0; ...; l_k \leftarrow l_{k+1}, ..., l_m, not\ l_{m+1}, ..., not\ l_n)$ as:

$$\mathbf{c} : l_0; ...; l_k \leftarrow l_{k+1}, ..., l_m, not\ l_{m+1}, ..., not\ l_n.$$

For a possibilistic rule $p = (r, c)$ we use p^* to denote r, i.e. the classical rule obtained by ignoring the certainty. Similarly, for a possibilistic program P we use P^* to denote the set of rules $\{p^* \mid p \in P\}$. The set of all weights found in a possibilistic program P is denoted by $cert(P) = \{c \mid p = (r, c) \in P\}$ and we also use the extended set of weights $cert^+(P) = \{c \mid c \in cert(P)\} \cup \{1 - c \mid c \in cert(P)\} \cup \{0, 1/2, 1\}$ where the intermediate weight $1/2$ is needed due to the particular treatment of negation-as-failure.[1]

Current approaches to PASP are based on a generalization of the concept of a partial Herbrand interpretation. In classical ASP, a partial Herbrand interpretation can be seen as a mapping $I : Lit_P \rightarrow \{0, 1\}$, i.e. a literal $l \in Lit_P$ is either true or false. This notion is generalized in PASP to a *valuation*, which is a function $V : Lit_P \rightarrow [0, 1]$. The underlying intuition of $V(l) = c$ is that the literal 'l' is true with certainty 'c'. Note that, like partial Herbrand interpretations in ASP, these valuations are of an epistemic nature,

[1]Specifically, programs such as the program with the single rule $(\mathbf{1} : a \leftarrow not\ a)$ will give rise to the answer set $\{a^{1/2}\}$ in the semantics from [4].

i.e. they reflect what we know about the truth of literals. For notational convenience, we often also use the set notation $V = \{l^c, \ldots\}$. In accordance with this set notation, we write $V = \emptyset$ to denote the valuation in which each literal is mapped to 0. For $c \in [0, 1]$ a certainty and V a valuation, we use V^c to denote the classical projection $\{l \mid l \in Lit_P, V(l) \geq c\}$. We also use $V^{\underline{c}} = \{l \mid l \in Lit_P, V(l) > c\}$, i.e. those literals that can be derived to be true with certainty strictly greater than 'c'. A valuation is said to be *consistent* when $V^{\underline{0}}$ is consistent. In such a case, there always exists a normalized possibility distribution π_V such that $N_V(l) = V(l)$.

Before we discuss the approaches from [3] and [4] in detail, we highlight that, semantically, both approaches are essentially the same in terms of the effect that they have on the reduct of a rule. Given a possibilistic rule r of the form:

$$\mathbf{c} : l_0; ...; l_k \leftarrow l_{k+1}, ..., l_m, not\ l_{m+1}, ..., not\ l_n$$

the certainty of the reduct w.r.t. a valuation V is given by:

$$min(c, F_N(V(l_{m+1})), ..., F_N(V(l_n)))$$

with F_N a fuzzy negator, i.e. F_N a decreasing function with $F_N(0) = 1$ and $F_N(1) = 0$. In particular, for the semantics of [3] we have that F_N is the Gödel negator F_{G}, defined as $F_{\mathrm{G}}(0) = 1$ and $F_{\mathrm{G}}(c) = 0$ with $0 < c \leq 1$. In [4], F_N is the Łukasievicz negator $F_{\mathrm{Ł}}(c) = 1 - c$ with $0 \leq c \leq 1$. Thus, for a rule such as:

$$\mathbf{0.9} : b \leftarrow not\ a$$

and a valuation $V = \{a^{0.2}\}$ we obtain under the approach from [3] the reduct $\mathbf{0} : b \leftarrow$, whereas under the semantics of [4] we obtain the reduct $\mathbf{0.8} : b \leftarrow$. In the remainder of the paper, due to the difference in the negator, we will refer to PASP under the semantics of [3] and [4] as $\mathrm{PASP}_{\mathrm{G}}$ and $\mathrm{PASP}_{\mathrm{Ł}}$, respectively. However, even though the semantics share similarities, there is a notable difference in the underlying intuition of both approaches. Indeed, both approaches have their own applications, just as the choice of a fuzzy connective in fuzzy logic is dependent on the problem at hand.

2.3.1. Using Gödel negation

We present a trivial extension of the semantics for PASP introduced in [3]. Let the c-cut P_c of a possibilistic program P, with $c \in [0, 1]$, be defined as:

$$P_c = \{r \mid (r, c') \in P \text{ and } c' \geq c\},$$

i.e. the rules in P with an associated certainty higher than or equal to 'c'.

Definition 1. Let P be a possibilistic simple program and V a valuation. The immediate consequence operator T_P is defined as:

$$T_P(V)(l_0) = \max\{c \in [0,1] \mid V^c \models l_1, ..., l_m \text{ and } ((l_0 \leftarrow l_1, ..., l_m), c') \in P_c\}.$$

The intuition of Definition 1 is that we can derive the head only with the certainty of the weakest piece of information, i.e. the necessity of the conclusion is restricted either by the certainty of the rule itself or the lowest certainty of the literals used in the body of the rule. Note that the immediate consequence operator defined in Definition 1 is equivalent to the one proposed in [3], although we formulate it somewhat differently. Also, the work from [3] only considered definite programs, even though adding classical negation does not impose any problems.

As before, we use $fp(P)$ to denote the fixpoint obtained by repeatedly applying T_P starting from the minimal valuation $V = \emptyset$, i.e. the least fixpoint of T_P w.r.t. set inclusion. A valuation V is said to be the answer set of a possibilistic definite program if $V = fp(P)$ and V is consistent. Answer sets of possibilistic normal programs are defined using a reduct. Let L be a set of literals. The reduct P^L of a possibilistic normal program is defined as [3]:

$$P^L = \{(head(r) \leftarrow body_+(r), c) \mid (r, c) \in P \text{ and } body_-(r) \cap L = \emptyset\}.$$

A consistent valuation V is said to be an answer set of the possibilistic normal program P iff $fp(\left(P^{(V^{\underline{0}})}\right)) = V$, i.e. if V is the answer set of the reduct $P^{(V^{\underline{0}})}$. In [3] classical negation was not considered, but adding classical negation does not impose any problems. Indeed, classical negation can easily be simulated in ASP [7].

Example 2. Consider the possibilistic normal program P with the rules:

$$\mathbf{0.1}: normal \leftarrow \quad \mathbf{1}: abnormal \leftarrow not\ normal \quad \mathbf{0.8}: problematic \leftarrow abnormal$$

This program describes an automated computer system. We have very limited evidence that this system is still operating normally. If the system is no longer operating normally, it is operating abnormally. And if the system is operating abnormally, it is very likely that keeping the system running will result in problematic behavior.

It is easy to verify that $\{normal^{0.1}\}$ is a possibilistic answer set of P. Indeed, $P^{\{normal\}}$ is the set of rules:

$$\mathbf{0.1}: normal \leftarrow \qquad\qquad \mathbf{0.8}: problematic \leftarrow abnormal$$

from which it trivially follows that $fp(P) = \{normal^{0.1}\}$ since the body of the second rule is never true.

2.3.2. Using Łukasiewicz negation

Alternative semantics for PASP were proposed in [4]. This approach was later extended in [2] to possibilistic disjunctive programs. Intuitively, the semantics from [4, 2] take the certainty of literals into account when determining the reduct. The underlying intuition of '*not l*' is that '*it cannot be established that l is certain*', or, that it is possible that '$\neg l$' is true.

Definition 2. Let P be a possibilistic disjunctive program and let V be a valuation. For every $p \in P$, the constraint $\gamma_V(p)$ induced by $p = (r, c)$ with $r = (l_0; ...; l_k \leftarrow l_{k+1}, ..., l_m, not\ l_{m+1}, ..., not\ l_n)$ and V, with V a valuation, is given by:

$$\max\{N(l_0), ..., N(l_k)\} \geq \min\{N(l_{k+1}), ..., N(l_m), 1 - V(l_{m+1}), ..., 1 - V(l_n), c\}. \qquad (2)$$

$C_{(P,V)} = \{\gamma_V(p) \mid p \in P\}$ is the set of constraints imposed by program P and V, and $S_{(P,V)}$ is the set of all minimally specific possibilistic models of $C_{(P,V)}$.

Definition 3. Let P be a possibilistic disjunctive program and let V be a valuation. Let $\pi \in S_{(P,V)}$ be such that

$$\forall l \in Lit_P \cdot N(l) = V(l)$$

then $V = \{l^{N(l)} \mid l \in Lit_P\}$ is called a possibilistic answer set of P.

Example 3. Consider the possibilistic normal program P from Example 2. We verify that $V = \{normal^{0.1}, abnormal^{0.9}, problematic^{0.8}\}$ is a possibilistic answer set of P under the semantics of [4]. The set of constraints $C_{(P,V)}$ induced by the program and the valuation V is given by:

$$\begin{aligned} N(normal) &\geq 0.1 \\ N(abnormal) &\geq \min\{1 - 0.1, 1\} \\ N(problematic) &\geq \min\{N(abnormal), 0.8\} \end{aligned}$$

from which it readily follows that $N(normal) \geq 0.1$, $N(abnormal) \geq 0.9$ and $N(problematic) \geq 0.8$. It thus trivially holds for the unique minimally specific possibility distribution $\pi \in S_{(P,V)}$ that $N(l) = V(l)$ for every $l \in Lit_P$.

It should be noted that under the semantics from [4] there is no longer a one-on-one mapping between the classic answer sets of a program and the possibilistic answer sets of the corresponding PASP program where we attach certainty $c = 1$ to each of the classical rules. Indeed, a possibilistic program such as $P = \{(\mathbf{1}: a \leftarrow not\ b), (\mathbf{1}: b \leftarrow not\ a)\}$ has an infinite number of possibilistic answer sets, i.e. $\{a^c, b^{1-c}\}$ for every $c \in [0, 1]$.

2.4. Complexity theory

We now recall some notions from complexity theory. The complexity classes Σ_k^{P} and Π_k^{P}, $0 \leq k \leq 3$, are defined as follows [9]:

$$\Sigma_0^{\mathsf{P}} = \Pi_0^{\mathsf{P}} = \mathsf{P}$$

$$\Sigma_1^{\mathsf{P}} = \mathsf{NP} \qquad \Sigma_2^{\mathsf{P}} = \mathsf{NP}^{\mathsf{NP}} \qquad \Sigma_3^{\mathsf{P}} = \mathsf{NP}^{\Sigma_2^{\mathsf{P}}}$$

$$\Pi_1^{\mathsf{P}} = \mathsf{coNP} \qquad \Pi_2^{\mathsf{P}} = \mathsf{co}\Sigma_2^{\mathsf{P}} \qquad \Pi_3^{\mathsf{P}} = \mathsf{co}\Sigma_3^{\mathsf{P}}$$

where $\mathsf{NP}^{\mathsf{NP}}$ is the class of problems that can be solved in polynomial time on a non-deterministic machine with an NP oracle, i.e. assuming a procedure that can solve NP problems in constant time. Likewise, $\mathsf{NP}^{\Sigma_2^{\mathsf{P}}}$ is the class of problems that can be solved in polynomial time on a non-deterministic machine with an Σ_2^{P} oracle. Then Π_2^{P} and Π_3^{P} are the classes of problems whose complement is Σ_2^{P} and Σ_3^{P}, respectively. For a general complexity class C, a problem is C-hard if any other problem in C can be reduced to this problem in polynomial time. A problem is said to be C-complete if the problem is in C and the problem is C-hard. Deciding the validity of a Quantified Boolean Formula or QBF of the form $\phi = \exists X_1 \forall X_2 \cdot p(X_1, X_2)$ with $p(X_1, X_2)$ a formula in disjunctive normal form is the canonical Σ_2^{P}-complete problem. Deciding the validity of a QBF of the form $\phi = \forall X_1 \exists X_2 \forall X_3 \cdot p(X_1, X_2)$ with $p(X_1, X_2)$ a formula in disjunctive normal form is the canonical Π_3^{P}-complete problem. These two canonical problems will later be used in the proofs listed in this paper.

3. Semantics and Complexity Results of Uncertain Rules

In PASP_{G} and PASP_{L} the weight associated with a rule is interpreted as the certainty with which we can deduce the head when the body is known

to hold. As such, we obtain a semantics based on weighted epistemic states, where we relate exactly one possibility distribution with each possibilistic answer set of the program. We can, however, look at these certainties in another way. Rather than considering weighted epistemic states, we can use Boolean epistemic states and use the certainties associated with the rules to express that some epistemic states are more plausible than others. We thus look at the weight associated with a rule as expressing our uncertainty as to whether the rule is valid. Indeed, the information encoded in the various rules may e.g. come from different sources and this may affect the degree to which we believe the rule to be valid.

Example 4. Consider the program P with the rules:

$$\mathbf{0.2}: \mathit{raining} \leftarrow$$
$$\mathbf{0.9}: \mathit{slippery} \leftarrow \mathit{raining}$$
$$\mathbf{0.7}: \mathit{safe} \leftarrow \mathit{not\ slippery}.$$

We will use this program to clarify the semantics proposed in this subsection. Intuitively, the program encodes the knowledge that it is raining, that raining makes the floor slippery and that a floor which is not slippery is safe to walk on without risk of injury.

Clearly, if we incorrectly consider a rule to be valid we may draw incorrect conclusions. The usual strategy, which is adopted in e.g. possibilistic logic, is to discard the least reliable pieces of knowledge when we want to ensure that what we derive is reliable. However, such a strategy would not work in ASP due to its non-monotonic nature. Indeed, while failing to discard incorrect rules may lead to erroneous conclusions, the same may happen when we incorrectly discard a valid rule. For example, if we omit the information that it is raining because it is insufficiently reliable, then we are not able to conclude that the floor is slippery and thus we are able to derive that it is safe to walk on the floor. Hence, to assess the certainty with which a literal can be derived, we need to consider all the subprograms of a given program, including the complete program itself. Some of these subprograms are more likely than others to correspond with an accurate representation of the considered problem. An answer set is then said to be necessary when it is an answer set of all the plausible subprograms. Similarly, we say that an answer set is possible to the extent that it is an answer set of some plausible subprogram.

Each subprogram corresponds with the assumption that some particular rules are wrong, namely those rules from the program that are missing, while the rules in the subprogram are assumed to be correct. Ideally, we would thus want to associate a possibility degree with each subprogram, i.e. the degree with which the assumption that the subprogram consists exactly of the valid rules is compatible with the certainty that we have of the rules. In practice, however, it is not feasible to list all subprograms and associate a possibility with each individual subprogram. Instead, we encode a possibility distribution over subprograms by associating a certainty with each rule in our possibilistic program P. The possibility of each subset is then determined by looking at the certainties of the rules that are omitted from the program. If we omit a rule with a certainty of 1, i.e. a rule of which we are certain that it is valid, then the rules in the subprogram can never be the set of all rules that are valid. Thus, our possibility that the rules in the subprogram correspond with the set of valid rules is 0. Conversely, if we only omit rules with a low certainty, then we retain a high possibility that the rules in our subprogram are exactly those rules that are valid. In particular, in the above example, omitting only the first rule would result in a subprogram with a high possibility of 0.8. However, omitting the second rule with a certainty of 0.9 would result in a subprogram where we only have a possibility of 0.1 that it contains all the valid rules.

Following this line of reasoning, we conceptually need a possibility distribution over subprograms of P. Specifically, a possibilistic rule (r, c) is interpreted as the constraint $N(r) \geq c$, where the necessity measure $N(r)$ stands for $N(\{P' \mid P' \subseteq P \text{ and } P' \text{ contains the rule } r\})$. The possibility distribution π_P is then the least specific possibility distribution that satisfies these constraints. Thus, a subprogram is considered possible to the extent that it contains all of the certain rules.

Definition 4. Let P be a possibilistic ASP program. We define the possibility distribution π_P over the subsets $P' \subseteq P$ as

$$\pi_P(P') = \begin{cases} 1 - \max\{c \mid (r, c) \in P \setminus P'\} & \text{when } P'^* \text{ consistent} \\ 0 & \text{otherwise} \end{cases}$$

Intuitively, this definition states that the less certain the rules are that we omit from the subprogram P', the more possible it is that P' is the correct program. It is not hard to see that this definition corresponds with the least specific possibility distribution that satisfies the constraints $N(r) \geq c$ for

every $(r, c) \in P$, with the additional constraint that inconsistent programs are impossible. Indeed, we have that:

$$\begin{aligned}
&\forall (r, c) \in P \cdot N(r) \geq c \\
&\equiv \forall (r, c) \in P \cdot N(\{P' \mid P' \subseteq P \text{ and } (r, c) \in P'\}) \geq c \\
&\equiv \forall (r, c) \in P \cdot \Pi(\{P' \mid P' \subseteq P \text{ and } (r, c) \in (P \setminus P')\}) \leq 1 - c \\
&\equiv \forall (r, c) \in P \cdot \max\{\pi(P') \mid P' \subseteq P, (r, c) \in (P \setminus P')\} \leq 1 - c \\
&\equiv \forall (r, c) \in P \cdot \forall P' \subseteq P, (r, c) \in (P \setminus P') \cdot \pi(P') \leq 1 - c \\
&\equiv \pi(P') \leq \min\{1 - c \mid P' \subseteq P, (r, c) \in (P \setminus P')\} \\
&\equiv \pi(P') \leq 1 - \max\{c \mid P' \subseteq P, (r, c) \in (P \setminus P')\} \\
&\equiv \pi(P') \leq \pi_P(P')
\end{aligned}$$

Notice furthermore that π_P is a normalized possibility distribution whenever P^* is consistent since then $\pi_P(P) = 1$.

Example 5. Consider the program P from Example 4. For compactness, we name the rules r_1, r_2 and r_3 from top to bottom. The possibility distribution π over the subprograms of P is defined as:

$$\begin{aligned}
\pi(\{r_1, r_2, r_3\}) &= 1 & \qquad \pi(\{r_2, r_3\}) &= 0.8 \\
\pi(\{r_1, r_3\}) &= 0.1 & \qquad \pi(\{r_3\}) &= 0.1 \\
\pi(\{r_1, r_2\}) &= 0.3 & \qquad \pi(\{r_2\}) &= 0.3 \\
\pi(\{r_1\}) &= 0.1 & \qquad \pi(\{\}) &= 0.1
\end{aligned}$$

We now define the main reasoning tasks for these new semantics.

Definition 5. Let P be a possibilistic ASP program. Let π_P be as in Definition 4. The degree to which it is possible that 'l' is a brave/cautious consequence of P is defined as:

$$\begin{aligned}
\Pi\left(P \models^{\mathrm{b}} l\right) &= \max\left\{\pi_P(P') \mid P' \subseteq P \text{ and } P'^* \models^{\mathrm{b}} l\right\} \\
\Pi\left(P \models^{\mathrm{c}} l\right) &= \max\left\{\pi_P(P') \mid P' \subseteq P \text{ and } P'^* \models^{\mathrm{c}} l\right\}
\end{aligned}$$

i.e. this is the degree to which some program $P' \subseteq P$ is possible which has 'l' as a brave/cautious consequence. The degree to which it is necessary that P has 'l' as a brave/cautious consequence is defined as:

$$\begin{aligned}
N\left(P \models^{\mathrm{b}} l\right) &= 1 - \max\left\{\pi_P(P') \mid P' \subseteq P \text{ and } P'^* \not\models^{\mathrm{b}} l\right\} \\
N\left(P \models^{\mathrm{c}} l\right) &= 1 - \max\left\{\pi_P(P') \mid P' \subseteq P \text{ and } P'^* \not\models^{\mathrm{c}} l\right\}.
\end{aligned}$$

Note that this is the degree to which all programs $P' \subseteq P$ that do not have 'l' as a brave/cautious consequence are impossible.

In the remainder of this section, we will also write $P \models^{\mathrm{b}}_{\Pi} l^{\lambda}$ to denote that $\Pi\left(P \models^{\mathrm{b}} l\right) \geq \lambda$, and similar for the notations $P \models^{\mathrm{c}}_{\Pi} l^{\lambda}$, $P \models^{\mathrm{b}}_{N} l^{\lambda}$ and $P \models^{\mathrm{c}}_{N} l^{\lambda}$. The differences between these types of inference are shown in the next example.

Example 6. Consider the PASP program P with the rules:

$$\mathbf{0.8}: b \leftarrow not\ c \qquad \mathbf{0.3}: c \leftarrow d, not\ b \qquad \mathbf{0.9}: d \leftarrow .$$

We have that

$$\begin{aligned}
&\pi_P(P) = 1 && \{b,d\}, \{c,d\}\\
&\pi_P(\mathbf{0.8}: b \leftarrow not\ c; \mathbf{0.9}: d \leftarrow) = 0.7 && \{b,d\}\\
&\pi_P(\mathbf{0.3}: c \leftarrow d, not\ b; \mathbf{0.9}: d \leftarrow) = 0.2 && \{c,d\}\\
&\pi_P(\mathbf{0.9}: d \leftarrow) = 0.2 && \{d\}\\
&\pi_P(\mathbf{0.8}: b \leftarrow not\ c; \mathbf{0.3}: c \leftarrow d, not\ b) = 0.1 && \{b\}\\
&\pi_P(\mathbf{0.8}: b \leftarrow not\ c) = 0.1 && \{b\}\\
&\pi_P(\mathbf{0.3}: c \leftarrow d, not\ b) = 0.1 && \{\}\\
&\pi_P(\{\}) = 0.1 && \{\}.
\end{aligned}$$

where the possibility associated with each subprogram is shown on the left and the classical answer set(s) of each subprogram is shown on the right. We obtain the following conclusions:

$$\begin{aligned}
&P \models^{\mathrm{b}}_{N} \left\{b^{0.8}, c^{0.3}, d^{0.9}\right\} && P \models^{\mathrm{c}}_{N} \left\{b^{0}, c^{0}, d^{0.9}\right\}\\
&P \models^{\mathrm{b}}_{\Pi} \left\{b^{1}, c^{1}, d^{1}\right\} && P \models^{\mathrm{c}}_{\Pi} \left\{b^{0.7}, c^{0.2}, d^{1}\right\}.
\end{aligned}$$

Furthermore, when the particular understanding of negation-as-failure in [3] prevents us from obtaining intuitive results, we can use the approach presented in this section, which offers results that are in line with PASP$_{\text{Ł}}$. We illustrate this in the next example.

Example 7. Consider the possibilistic normal program P from Example 3 with the rules

$$\begin{aligned}
&\mathbf{0.1}: normal \leftarrow\\
&\mathbf{1.0}: abnormal \leftarrow not\ normal\\
&\mathbf{0.8}: problematic \leftarrow abnormal.
\end{aligned}$$

Note that the intuition of the rules has changed slightly under the new semantics. For example, we are certain with a fairly high degree that the last rule, which describes that an abnormal system is behaving problematically, is indeed valid. In $\mathrm{PASP}_{\mathrm{L}}$, one would expect a conclusion in which we deduce with a high certainty that the system will give problematic errors. Indeed, the second rule states that we will assume that the system is working abnormally, unless we are very certain that the system is working normally, i.e. we act cautiously. Using the last rule, we then obtain with a high certainty that the system will cause problematic behavior. The results obtained by the different semantics are:

semantics from [3]	$\{normal^{0.1}\}$
semantics from [4]	$\{normal^{0.1}, abnormal^{0.9}, problematic^{0.8}\}$
semantics from Section 3	$P \models^{\mathrm{c}}_{N} \{normal^{0.1}, abnormal^{0}, problematic^{0}\}$
	$P \models^{\mathrm{b}}_{N} \{normal^{0.1}, abnormal^{0}, problematic^{0}\}$
	$P \models^{\mathrm{c}}_{\Pi} \{normal^{1}, abnormal^{0.9}, problematic^{0.9}\}$
	$P \models^{\mathrm{b}}_{\Pi} \{normal^{1}, abnormal^{0.9}, problematic^{0.9}\}$

Note that in $\mathrm{PASP}_{\mathrm{G}}$, because the certainty is ignored when determining the reduct, '*normal*' is assumed to be true without doubt. Hence, the second rule is removed from the reduct and we have no way of concluding that the system is performing abnormally. In $\mathrm{PASP}_{\mathrm{L}}$, however, the certainty is taken into consideration and we conclude, with a fairly high certainty, that '*abnormal*' is true. The semantics proposed in this section, for this given example, agree with $\mathrm{PASP}_{\mathrm{G}}$ when we are interested in the inference based on cautious necessity. On the other hand, the conclusions are closer to those of $\mathrm{PASP}_{\mathrm{L}}$ when we look at the inference based on brave possibility. Furthermore, note that since each subprogram has a unique answer set both brave and cautious reasoning coincide.

In general though, neither of the semantics for PASP need to agree with each other, as can be seen in the next example:

Example 8. Consider the PASP program P with the rules:

$$\begin{aligned} \mathbf{1}&: lost \leftarrow not\ visible \\ \mathbf{1}&: visible \leftarrow not\ hidden \\ \mathbf{0.5}&: hidden \leftarrow \end{aligned}$$

For compactness, we name these rules from top to bottom r_1, r_2 and r_3. Intuitively, this example describes a simple game where an agent loses when he cannot see an object. However, there is uncertainty as to whether or not the object itself is hidden. We have that:

$$\begin{aligned} \pi_P(P) &= 1 & \{hidden, lost\} \\ \pi_P(r_1, r_2) &= 0.5 & \{visible\} \end{aligned}$$

whereas the possibility of all the other subprograms $P' \subseteq P$ is 0. Since each subprogram has a unique answer set, brave and cautious reasoning once again coincide.
The results obtained by the different semantics are:

semantics from [3]	$\left\{hidden^{0.5}, lost^{1}\right\}$
semantics from [4]	$\left\{hidden^{0.5}, visible^{0.5}, lost^{0.5}\right\}$
semantics from Section 3	$P \models^{\mathrm{c}}_{N} \left\{hidden^{0.5}, visible^{0}, lost^{0.5}\right\}$
	$P \models^{\mathrm{b}}_{\Pi} \left\{hidden^{1}, visible^{0.5}, lost^{1}\right\}$

Neither of these conclusions agree. Nevertheless, the semantics proposed in this section provide an intuitively satisfiable answer to the outcome of the game that the agent plays. Indeed, we can conclude that it is entirely possible that the agent has lost (since $P \models^{\mathrm{b}}_{\Pi} lost^{1}$), while at the same time we know that this is not necessarily so (since $P \models^{\mathrm{c}}_{N} lost^{0.5}$).

Still, some interesting links exist between the semantics for PASP.

Proposition 1. *Let P be a simple possibilistic ASP program. For each literal 'l' we have that $N\left(P \models^{\mathrm{c}} l\right) \geq \lambda$ iff $V(l) \geq \lambda$ with V the possibilistic answer set of P under the semantics from Bauters [4], which in turn coincides with the semantics from Nicolas [3].*

The previous result is not surprising because, without negation-as-failure, all semantics for PASP adhere to the semantics of possibilistic logic. Furthermore, notice that for simple programs, which have a unique answer set, checking whether $N\left(P \models^{\mathrm{b}} l\right) \geq \lambda$ is equivalent to checking whether $N\left(P \models^{\mathrm{c}} l\right) \geq \lambda$ and, similarly, checking whether $\Pi\left(P \models^{\mathrm{c}} l\right) \geq \lambda$ is equivalent to checking whether $\Pi\left(P \models^{\mathrm{b}} l\right) \geq \lambda$. Thus, also the complexity of these reasoning types coincide. This is not the case for possibilistic normal/disjunctive programs.

Proposition 2. *Let P be a possibilistic normal program. Deciding whether*

$$\Pi\left(P \models^{\mathrm{b}} l\right) \geq \lambda \text{ is } \mathsf{NP}\text{-complete};$$
$$N\left(P \models^{\mathrm{c}} l\right) \geq \lambda \text{ is } \mathsf{coNP}\text{-complete};$$
$$\Pi\left(P \models^{\mathrm{c}} l\right) \geq \lambda \text{ is } \Sigma_2^{\mathsf{P}}\text{-complete};$$
$$N\left(P \models^{\mathrm{b}} l\right) \geq \lambda \text{ is } \Pi_2^{\mathsf{P}}\text{-complete}.$$

A similar jump in the polynomial hierarchy can be seen for possibilistic disjunctive programs.

Proposition 3. *Let P be a possibilistic disjunctive program. Deciding whether*

$$\Pi\left(P \models^{\mathrm{b}} l\right) \geq \lambda \text{ is } \Sigma_2^{\mathsf{P}}\text{-complete};$$
$$N\left(P \models^{\mathrm{c}} l\right) \geq \lambda \text{ is } \Pi_2^{\mathsf{P}}\text{-complete};$$
$$\Pi\left(P \models^{\mathrm{c}} l\right) \geq \lambda \text{ is } \Sigma_3^{\mathsf{P}}\text{-complete};$$
$$N\left(P \models^{\mathrm{b}} l\right) \geq \lambda \text{ is } \Pi_3^{\mathsf{P}}\text{-complete}.$$

Thus far, we have considered a possibility distribution over the subprograms. However, from an application perspective, it often makes more sense to consider the possibility or necessity of an answer set. Clearly, each subprogram $P' \subseteq P$ may have zero or more answer sets. Furthermore we may have that two subprograms $P' \subseteq P$ and $P'' \subseteq P$ have the same answer set, even if $\pi_P(P') \neq \pi_P(P'')$. This leads us to the following definition.

Definition 6. Let P be a possibilistic ASP program. Let π_P be the possibility distribution over the subsets $P' \subseteq P$. We define the possibility distribution π_A over the partial Herbrand interpretations M:

$$\pi_A(M) = \max\left\{\pi_P(P') \mid M \text{ is an answer set of } P'^*\right\}$$

Note that this definition implies that $\pi_A(M) = 0$ whenever M is not an answer set of any subprogram $P' \subseteq P$. Let π_A be the possibility distribution over the partial Herbrand interpretations M. The possibility that l is a literal in the epistemic state of the agent is given by $\Pi(l) = \max\{\pi_A(M) \mid l \in M\}$. The necessity that l is a literal in the epistemic state of the agent is given by $N(l) = 1 - \max\{\pi_A(M) \mid l \notin M\}$.

Interestingly, we have that $\Pi(l) = \Pi\left(P \models^{\mathrm{b}} l\right)$ and $N(l) = N\left(P \models^{\mathrm{c}} l\right)$.
We now show that the new semantics are a proper extension of ASP.

Example 9. Consider the program from Example 6. We can verify that:

$$\Pi(b) = 1 \qquad \Pi(c) = 1 \qquad \Pi(d) = 1$$
$$N(b) = 0 \qquad N(c) = 0 \qquad N(d) = 0.9$$

Since all the associated weights are 1, only the subprogram consisting of all the rules has a non-negative weight. Thus, only the classical answer sets are considered to be possible and the reasoning tasks from Definition 5 reduce to cautious and brave reasoning.

We now provide a more elaborate example, which highlights a complex setting in which humans can fairly easily come to a satisfiable conclusion, but which is not easy to encode using e.g. classical ASP.

Example 10. Triage at an accident site with a large number of casualties is an essential part of medical treatment when resources are limited. With the help of triage, it becomes possible to distinguish which casualties can wait for medical attention at a hospital and which casualties need to be treated on the spot. For brevity of this example, we consider a triage system with three levels. The casualty may have minor injuries (*minor*), which means that the person can wait for treatment at the hospital. The casualty may need to be treated immediately because of life-threatening, yet treatable injuries (*nowait*). The final category is beyond urgency (*beyond*) and encompasses those casualties which are so severely injured that, for the time being, medical attention is better directed towards casualties in the *nowait* category as the chances of survival of casualties in this latter category are far higher.

A rescue helper is faced with a casualty with extensive external injuries (*extensive*), which indicates that he/she either falls in the *nowait* or *beyond* category. The casualty is faintly moaning (*moaning*), which, with a very low certainty, is an indication of the casualty still being conscious (*conscious*). Similarly, the casualty is exhibiting a bleeding nose (*nosebleed*), which might indicate internal bleeding (*internal*). The rescue helper would be a lot more certain that the casualty is experiencing internal bleeding when he/she also had low blood pressure (*lowblood*), but this has not been established. Whenever the casualty does not appear to be conscious, he/she is assumed to be in the *nowait* or *beyond* category. When there is no indication of internal bleeding, the casualty is in the *nowait* category. A classification in one of the categories is never entirely certain since it is not possible, due to time constraints, to perform all the required tests. We have the program with the rules:

$\mathbf{1} : extensive \leftarrow$

$\mathbf{0.9}: minor \leftarrow not\ extensive$

$\mathbf{1}: moaning \leftarrow$

$\mathbf{0.1}: conscious \leftarrow moaning$

$\mathbf{0.9}: nowait \leftarrow not\ beyond, not\ internal, not\ conscious, extensive$

$\mathbf{0.9}: beyond \leftarrow not\ nowait, not\ conscious, extensive$

$\mathbf{1}: nosebleed \leftarrow$

$\mathbf{0.1}: internal \leftarrow nosebleed$

$\mathbf{0.7}: internal \leftarrow nosebleed, lowblood$

$\mathbf{1}: \leftarrow nowait, beyond, extensive$

$\mathbf{1}: \leftarrow not\ nowait, not\ beyond, extensive$

The last two rules encode that one and exactly one category needs to be assigned to the casualty (either *nowait* or *beyond*), a requirement for an efficient triage.

Notice that at least one rule needs to be omitted to make the program consistent. Indeed, if we look at the classical program by ignoring the weights, then it is clear that we have information (with varying degrees of certainty) to support both *nowait* and *beyond*. In PASP_{G}, we are unable to take the low certainty of the literal *conscious* into account when reasoning with negation-as-failure. As such, the literal *conscious*, for the purpose of determining the reduct, is considered as entirely true, which immediately removes the rules with *nowait* or *beyond* in the head of the rule from the reduct. In PASP_{L}, on the other hand, we are unable to choose between *nowait* and *beyond*. Indeed, we obtain an infinite number of answer sets such that the sum of the necessities of *nowait* and *beyond* equals 0.9 and such that the necessity for both *nowait* and *beyond* is higher or equal to 0.1. Under the semantics proposed in this paper, however, we obtain that $P \models^{\text{b}}_{\Pi} \{beyond^{0.9}, nowait^{0.9}\}$, $P \models^{\text{c}}_{\Pi} \{beyond^{0.9}, nowait^{0.1}\}$, $P \models^{\text{b}}_{N} \{beyond^{0.9}, nowait^{0.1}\}$ and $P \models^{\text{c}}_{N} \{beyond^{0.1}, nowait^{0.1}\}$. The new semantics are thus capable of arriving at the desired conclusion. Indeed, since the necessity associated with *beyond* is higher or equal to the necessity associated with *nowait* for all reasoning tasks, a reasonable classification for the casualty is *beyond*. This corresponds with our intuition, as a number of indications hint towards this worst case scenario (e.g. the bleeding nose). If we added the fact that the casualty has low blood pressure, then even a brave conclusion with possibility measures would indicate that the casualty

is beyond urgency, i.e. it would further reaffirm our conclusion.

4. Simulation of Uncertain Rules

We now show how the semantics presented in the previous section can be simulated using existing formalisms. The decision problems $\Pi(P \models^{\mathrm{b}} l) \geq \lambda$ or $N(P \models^{\mathrm{c}} l) \geq \lambda$ will be simulated using brave and cautious reasoning over classical programs. The remaining decision problems, which have a higher expressiveness, will be simulated by means of cautious abductive reasoning. We start by describing the simulation of the decision problems $\Pi(P \models^{\mathrm{b}} l) \geq \lambda$ or $N(P \models^{\mathrm{c}} l) \geq \lambda$, which share a common base program P_{basic}.

Definition 7. Let P be a possibilistic disjunctive program. We define $P_{\mathrm{basic}}(\lambda)$ as the set of rules:

$$\{r'_i \leftarrow \mathit{not}\ nr'_i \mid (r_i, c_i) \in P, c_i \leq 1 - \lambda\}$$
$$\{nr'_i \leftarrow \mathit{not}\ r'_i \mid (r_i, c_i) \in P, c_i \leq 1 - \lambda\} \tag{3}$$
$$\cup \{r'_i \leftarrow \mid (r_i, c_i) \in P, c_i > 1 - \lambda\} \tag{4}$$
$$\cup \{\mathit{head}(r_i) \leftarrow \mathit{body}(r_i) \cup \{r'_i\} \mid (r_i, c_i) \in P\} \tag{5}$$

Intuitively, the program $P_{\mathrm{basic}}(\lambda)$ simulates the semantics from Section 3 using classical ASP. In particular, the rules from (3) generate as many answer sets as there are choices of rules such that the possibility of the associated subprograms remains sufficiently high, i.e. greater than or equal to λ. The rules in (4) ensure that all rules with a sufficiently high certainty are considered as valid. Depending on the choice made in (3), the information encoded in the respective rules is applied using (5).

Example 11. Consider the possibilistic normal program P with the rules

$$\mathbf{0.8}\colon b \leftarrow \mathit{not}\ c \qquad \mathbf{0.3}\colon c \leftarrow d, \mathit{not}\ b \qquad \mathbf{0.9}\colon d \leftarrow .$$

We have the classical normal program $P_{\mathrm{basic}}(0.7)$ with the rules

$$r'_1 \leftarrow \qquad r'_2 \leftarrow \mathit{not}\ nr'_2 \qquad r'_3 \leftarrow$$
$$nr'_2 \leftarrow \mathit{not}\ r'_2$$
$$b \leftarrow \mathit{not}\ c, r'_1 \qquad c \leftarrow d, \mathit{not}\ b, r'_2 \qquad d \leftarrow r'_3$$

The program $P_{\mathrm{basic}}(\lambda)$ can now be extended to solve the main reasoning tasks on the second level of the polynomial hierarchy.

Definition 8. Let P be a possibilistic disjunctive program. The classical disjunctive program $P^{\Pi}_{brave}(l,\lambda)$ to verify $\Pi(P \models^{\mathrm{b}} l) \geq \lambda$ is defined as $P_{\mathrm{basic}}(\lambda) \cup \{\leftarrow not\ l\}$.

Intuitively, the simulation $P^{\Pi}_{brave}(l,\lambda)$ will look for a world in which 'l' is true, and has an associated possibility of λ. If such a world exists, i.e. if we have an answer set, then $\Pi(P \models^{\mathrm{b}} l) \geq \lambda$ is true.

Proposition 4. *Let P be a possibilistic disjunctive program and $P^{\Pi}_{brave}(l,\lambda)$ the classical disjunctive program as defined in Definition 8. We have that $\Pi(P \models^{\mathrm{b}} l) \geq \lambda$ iff $P^{\Pi}_{brave}(l,\lambda)$ has a classical consistent answer set.*

Definition 9. Let P be a possibilistic disjunctive program. The classical disjunctive program $P^{N}_{cautious}(l,\lambda)$ to verify $N(P \models^{\mathrm{c}} l) \geq \lambda$ with $\lambda > 0$ is defined as $P_{\mathrm{basic}}(1-\lambda') \cup \{\leftarrow l\}$ with $\lambda' \in cert^{+}(P)$ such that $\lambda' < \lambda$ and for which we have that $\nexists \lambda'' \in cert^{+}(P) \cdot \lambda' < \lambda'' < \lambda$.

Note that when $\lambda = 0$, it trivially holds that $N(P \models^{\mathrm{c}} l) \geq \lambda$ is true.

Intuitively, to determine whether $N(P \models^{\mathrm{c}} l) \geq \lambda$ we need to verify that $\max\{\pi_P(P') \mid P' \subseteq P \text{ and } P'^{*} \not\models^{\mathrm{c}} l\} \geq 1-\lambda$. In other words, whether we do not have any subprogram P' such that $P'^{*} \not\models^{\mathrm{c}} l$ and $\pi_P(P') > 1-\lambda$. The simulation $P^{N}_{cautious}(l,\lambda)$ intuitively looks for a world with a certainty higher than $1-\lambda$ in which 'l' is false, i.e. l is not a cautious consequence. If such a world does not exist, i.e. if we find no answer sets, then $N(P \models^{\mathrm{c}} l) \geq \lambda$.

Proposition 5. *Let P be a possibilistic disjunctive program, $\lambda > 0$ and $P^{N}_{cautious}(l,\lambda)$ the classical disjunctive program as defined in Definition 9. We have that $N(P \models^{\mathrm{c}} l) \geq \lambda$ iff $P^{N}_{cautious}(l,\lambda)$ has no classical consistent answer set.*

For the decision problems in Proposition 4 and 5, it was sufficient to find *one* answer set of a particular subprogram that satisfies a given condition. However, to decide whether $\Pi\left(P \models^{\mathrm{c}} l\right) \geq \lambda$ or $N\left(P \models^{\mathrm{b}} l\right) \geq \lambda$ we need to verify a particular condition *for each* answer set of a particular subprogram. Our simulation of the decision problems $\Pi\left(P \models^{\mathrm{c}} l\right) \geq \lambda$ and $N\left(P \models^{\mathrm{b}} l\right) \geq \lambda$ is based on the idea that a disjunctive ASP program can be used to reason about the answer sets of a normal program P. Specifically, it is possible to translate a normal program P to a set of clauses to ensure that the program is free of negation-as-failure. When a program is free of negation-as-failure and free of classical negation (a trivial transformation, see e.g. [7]) we can use saturation techniques [7] to reason about each answer set of P. Intuitively,

the saturation technique exploits the property that answer sets are minimal models. Desirable partial Herbrand models are saturated to ensure that these models cannot be minimal when undesirable partial Herbrand models exist (i.e. when partial Herbrand models exist that represent counter-examples). Using the saturation technique we both validate whether a given partial Herbrand interpretation is a valid model of the subprogram and whether a given literal is a desired cautious conclusion of the given subprogram. Crucially, every normal program can be translated to a set of clauses, e.g. by using the translation presented in [10]. Our simulation uses such a translation as a black box, which allows us to take advantage of state of the art methods.

Definition 10. Let $P = \{p_1, ..., p_n\}$ be a possibilistic normal program and every $p_i = (r_i, c_i)$ for $1 \leq i \leq n$ a possibilistic normal rule. The disjunctive program $P_{complex}(\lambda)$ is defined as the set of rules:

$$\{\mathrm{r}_i \leftarrow not\ \neg \mathrm{r}_i \mid 1 \leq i \leq n, c_i \leq 1 - \lambda\}$$
$$\cup \{\neg \mathrm{r}_i \leftarrow not\ \mathrm{r}_i \mid 1 \leq i \leq n, c_i \leq 1 - \lambda\} \quad (6)$$
$$\cup \{\mathrm{r}_i \leftarrow \mid 1 \leq i \leq n, c_i > 1 - \lambda\} \quad (7)$$

$$\cup \{cl \leftarrow \mid cl \in cls(P^{\mathrm{r}})^{\dagger}\} \quad (8)$$
$$\cup \{(sat \leftarrow a, na) \mid a \in at(cls(P^{\mathrm{r}})^{\dagger})\} \quad (9)$$
$$\cup \{(a \leftarrow sat) \mid a \in at(cls(P^{\mathrm{r}})^{\dagger})\}$$
$$\cup \{(na \leftarrow sat) \mid a \in at(cls(P^{\mathrm{r}})^{\dagger})\} \quad (10)$$
$$\cup \{\leftarrow not\ sat\} \quad (11)$$

$$\cup \{cl'_r \leftarrow \mid cl \in cls(P^{\mathrm{r}})^{\dagger}\} \quad (12)$$
$$\cup \{\leftarrow a', na' \mid a \in at(cls(P^{\mathrm{r}})^{\dagger})\} \quad (13)$$

where $cls(P)$ is a representation of the normal program P as a set of clauses (e.g. [10] or [11]), P^{r} is the set of rules $\{head(r_i) \leftarrow body(r_i), \mathrm{r}_i \mid (r_i, c_i) \in P\}$ with 'r_i' a fresh atom, $C^{\dagger}$ is the set of clauses obtained from C by replacing every occurrence of a negated atom $\neg a$ with a fresh atom na except for the atoms 'r_i'. Furthermore, we use $at(C)$ to denote the set of atoms appearing in C from which we remove the atoms 'r_i'. Finally, cl'_r is obtained from a clause cl by replacing every literal from Lit_P with l'.

Note that, when using ASP solvers which support choice rules [12], the set of rules in (6) can be defined as $\{1\{\mathrm{r}_i, \neg \mathrm{r}_i\}1 \leftarrow \mid 1 \leq i \leq n, c_i \leq 1 - \lambda\}$.

The intuition of this program is as follows. Rules (6) and (7) ensure that each answer set of $P_{complex}(\lambda)$ corresponds to exactly one subprogram $P' \subseteq P$. Furthermore, the partial Herbrand interpretations of $P_{complex}(\lambda)$ that satisfy (8) then correspond with the answer sets of P'. The rules in (9) use saturation to simulate classical negation. The saturation itself is encoded in the rules (10). Clearly, we do not want answer sets that represent counter-examples, so we use a constraint rule in (11) to eliminate those models that are not saturated. However, if P' itself is inconsistent, we do not want to consider this program since then $\pi(P') = 0$. The rules in (8–11) are too weak to eliminate such an inconsistent answer set. As such, we need to simulate P' once more with the rules (12) and (13) to ensure that subprograms with inconsistent answer sets are not considered in our simulation. Notice that while this simulation mostly uses fresh literals, the literals r_i remain unchanged. Thus the subprogram simulated in (12) is exactly the subprogram that is simulated in (8).[2]
Proposition 6 and 7 further extend upon $P_{complex}(\lambda)$ to effectively simulate the reasoning tasks $\Pi\left(P \models^{\mathrm{c}} l\right) \geq \lambda$ or $N\left(P \models^{\mathrm{b}} l\right) \geq \lambda$, respectively, while at the same time verifying their correctness.

Proposition 6. *Let P be a possibilistic normal program and $P^{\mathrm{c}}_{\Pi}(l, \lambda)$ the disjunctive program defined as $P_{complex}(\lambda) \cup \{sat \leftarrow l\}$. Then $\Pi\left(P \models^{\mathrm{c}} l\right) \geq \lambda$ iff $P^{\mathrm{c}}_{\Pi}(l, \lambda)$ has a classical answer set.*

Proposition 7. *Let P be a possibilistic normal program and $P^{\mathrm{b}}_{\mathrm{N}}(l, \lambda)$ the disjunctive program defined as $P_{complex}(1 - \lambda') \cup \{sat \leftarrow not\ l\}$ with λ' defined as in Proposition 5. Then $N\left(P \models^{\mathrm{b}} l\right) \geq \lambda$ iff $P^{\mathrm{b}}_{\mathrm{N}}(l, \lambda)$ has no classical answer set.*

5. Certain programs with optional rules

The simulations provided in the previous section for the decision problems $\Pi\left(P \models^{\mathrm{c}} l\right) \geq \lambda$ and $N\left(P \models^{\mathrm{b}} l\right) \geq \lambda$ are not only useful for reasoning with uncertain answer set programs, but can be applied to the much wider range of problems that can be encoded as programs with optional rules. In particular, in this section we prove how two interesting AI problems, namely cautious

[2] A prototype implementation of this simulation using disjunctive ASP is under continual development and can be found online at: http://www.cwi.ugent.be/kim/pasp2asp/ .

abductive reasoning and conformant planning, can be expressed in terms of programs with optional rules. As such, we can use the simulation presented in the previous section, along with off-the-shelf ASP solvers, to efficiently compute cautious abductive reasoning and the first single-step implementation of conformant planning in ASP. Both problems can also trivially be extended with weights.

5.1. Cautious abductive reasoning

An abductive diagnosis program [13] is encoded as a triple $\langle H, T, O\rangle$ where H is a set of literals referred to as hypotheses, T is a (normal) ASP program referred to as the theory and O is a set of literals referred to as observations. Intuitively, the theory T describes the dynamics of a system, the observations O describe the observed state of the system and the hypotheses H are those literals that can be used to try and explain such observations within the theory. Cautious abductive reasoning is concerned with the problem of finding hypotheses that could explain the observations in O. Thus, we are interested in a set $E \subseteq H$ such that $T \cup E \models^{\mathrm{c}} O$, where E is said to be a cautious explanation.

Proposition 8. *Let P_{abd} be the possibilistic normal program defined for an abductive diagnosis program $\langle H, T, O\rangle$ as*

$$\{\mathbf{0.5} : block\_h \leftarrow \mid h \in H\} \tag{14}$$
$$\cup \{\mathbf{1} : h \leftarrow not\ block\_h \mid h \in H\} \tag{15}$$
$$\cup \{\mathbf{1} : goal \leftarrow O\} \tag{16}$$
$$\cup \{\mathbf{1} : r \mid r \in T\}\,. \tag{17}$$

It holds that $\langle H, T, O\rangle$ has a cautious explanation iff $\Pi\,(P_{\mathrm{abd}} \models^{\mathrm{c}} goal) \geq 0.5$. In particular, we have that E is a cautious explanation if and only if we have for $P' = P_{\mathrm{abd}} \setminus \{block\_h \leftarrow \mid h \in E\}$ that $P' \models^{\mathrm{c}} goal$.

It is furthermore trivial to extend cautious abductive reasoning with weights, which further increases the expressiveness of cautious abductive reasoning.

Example 12. John wants to become rich. He can either choose to invest his money in stocks, or to invest it in bonds (he lacks the money to do both). John knows that the stock market will either end with a gain (*gain*) or with a loss (*loss*). When the overall stock market has gained value, then he is very certain that also the stocks he bought will have gained value, i.e. he wins

a lot of money (*win*). When the stock market has lost money, he is fairly certain that he will fail to recuperate his investment (*fail*). When he fails to do so, he goes bankrupt (*bankrupt*). As an alternative to the stock market, he can also invest his money in bonds. He is somewhat certain that bonds will make him rich, i.e. bonds are safer but John has a lower certainty that bonds will help him to become rich. We have:

$$\begin{array}{rr} \mathbf{1}: gain \leftarrow not\ loss & \mathbf{1}: loss \leftarrow not\ gain \\ \mathbf{0.9}: win \leftarrow gain, stocks & \mathbf{0.8}: fail \leftarrow loss, stocks \\ \mathbf{1}: rich \leftarrow win & \mathbf{1}: bankrupt \leftarrow fail \\ \mathbf{0.5}: rich \leftarrow bonds & \mathbf{1}: \leftarrow stocks, bonds \end{array}$$

Given the hypotheses $H = \{stocks, bonds\}$, it is clear that when we ignore the weights only $E = \{bonds\}$ is a cautious abductive explanation for the observation $O = \{rich\}$. If we take the certainties into account, then $E_1 = \{bonds\}$ is only a cautious explanation when we take $\lambda = 0.5$. In other words: we are only somewhat certain that the action *bonds* will cautiously make us rich. Notice that $E_2 = \{stocks\}$ will only be a cautious explanation for $\lambda = 0.2$. Indeed, we are far less certain that buying stocks will be a guaranteed way to make us rich. The cautious explanation with the highest certainty to make us rich is therefore to buy bonds. If, however, we were looking for a brave explanation, then we would sooner be advised to buy stocks as these have a high potential for making us rich.

5.2. Conformant planning

Conformant planning is the problem of determining whether a plan (i.e. a series of actions) exists that always leads to the desired goal, regardless of the incompletely known initial state of the agent. Such problems are typically expressed using an action language.

An action language is built from a finite number of *fluents* $f_1, ..., f_n$. A *state* is a finite set of fluents. The properties of the *initial state* s_0 are described by formulas of the form '$\mathsf{initially}\ f$', which are called *value propositions*, with f a *fluent literal*, i.e. a fluent or a fluent preceded by $\neg$. Changes of states are defined using a finite set of *action*s, which are formulas of the form '$a\ \mathsf{causes}\ f\ \mathsf{if}\ f_1, ..., f_m$', which are called *effect propositions*, with $f, f_1, ..., f_m$ fluent literals. A *domain* D is a finite set of value and effect propositions. A *proper* domain, to which we limit ourselves in this paper, is a domain in which we can determine in polynomial time what the successor state is, given

the current state and an action. A *plan* is a sequence of actions $[a_1, ..., a_m]$. The *planning problem* is to determine for a given domain D and a fluent literal f whether a plan exists leading from s_0 to a state in which f is true, where we call f the *goal fluent*. To solve a planning problem, the domain is translated to ASP. Particularly, such a translation can be written as $P_{\text{act}} \cup P_{\text{rem}}$ where P_{act} are those rules used to describe the actions, whereas P_{rem} are the remaining rules that among others describe the (incomplete) initial state and rules to ensure inertia. Then, a plan exists when an answer set contains the goal fluent.

However, not all forms of planning problems can be solved in this way. When we say that we have an incomplete domain, this means that the initial values of some fluents are unknown. *Conformant planning* is the problem of determining whether for an incomplete domain and a fluent f a plan exists leading to a state in which f is true, regardless of the initial values of the unknown fluents. Only some action languages, e.g. $\mathcal{K}$ [14, 15], have the expressive power to describe conformant planning problems. For solving such problems, $DLV^{\mathcal{K}}$ relies on a two-step translation to ASP where a plan is generated (that is not necessarily a conformant plan) and verified to be an actual conformant plan, until an actual conformant plan is found. However, these methods are not designed to work with uncertainty and cannot, e.g. compute the most reliable plan when no conformant plan can be found.

We now show how conformant planning can be expressed in terms of a decision problem of the form $N\left(P \models^{\text{b}} l\right) \geq \lambda$. Note that the existence of a conformant plan can be also written as $\exists p\, \forall iv \cdot P(p, iv, pp)$ where $P(p, iv, pp)$ describes that for the planning problem pp and for all initially unknown values iv the plan p leads to the goal fluent.

Proposition 9. *Let P_{con} be the possibilistic normal program defined for a conformant planning problem with the atom 'goal' the desired goal fluent. We express the domain knowledge as a normal ASP program $P_{\text{act}} \cup P_{\text{rem}}$. Then P_{con} is:*

$$\{\mathbf{0.5} : block\_i \leftarrow \mid r_i \in P_{\text{act}}\} \tag{18}$$

$$\cup \{\mathbf{1} : H(r_i) \leftarrow B(r_i) \cup \{not\ block\_i\} \mid r_i \in P_{\text{act}}\} \tag{19}$$

$$\cup \{\mathbf{1} : r \mid r \in P_{\text{rem}}\} \tag{20}$$

$$\cup \{\mathbf{1} : \leftarrow not\ goal\} \tag{21}$$

A conformant plan exists iff $\Pi\left(P_{\text{con}} \models^{\text{c}} goal\right) \geq 0.5$.

6. Related Work

The combination of logic programming and uncertainty handling in a single framework has been an active topic of research during the last decennia. The idea of combining logic programming and possibility theory was pioneered in [16]. However, this approach was limited to classical formulas and as such did not include default negation, i.e. non-monotonic reasoning. Shortly thereafter, in [17], a framework was proposed in which stable models are combined with a semi-possibilistic first-order logic, used as a logic of graded truth, to deal with uncertainty. Specifically, this semi-possibilistic logic is a compositional version of possibilistic logic, in which compositionality is preserved on the basis of a Heyting algebra. As a direct consequence, classical Boolean tautologies are no longer preserved in semi-possibilistic logic [18]. A more recent approach is the work from [19], which combines defeasible logic, a form of non-monotonic reasoning involving both strict and defeasible rules, with possibility theory in a single framework. This allows, among other things, to resolve conflicts between contradictory goals. Possibility theory has also been combined with argumentation frameworks, e.g. in [20], where revision rules allow an agent to revise its beliefs and goals.

One of the first works to explore the combination of possibility theory and ASP was [3] in which the PASP framework for normal programs was introduced. In [21], an extension to PASP was proposed to make it applicable to possibilistic disjunctive programs. Both approaches, however, share the same reduct operator in which the certainty is not taken into account when removing negation-as-failure. Later, in [4], alternative semantics for PASP were introduced that adhere to a different intuition when dealing with negation-as-failure. While in [3] we have that '*not* l' is true when we are more certain that '$\neg l$' rather than 'l' is true, in [4] we have that '*not* l' is true to the degree in which it is possible that '$\neg l$' is true. Rules in [4] are interpreted as constraints on possibility distributions, where answer sets correspond with epistemic states and where the rules are used to reason over these epistemic states. The work from [22] later showed that answer set programming can indeed be seen as a form of meta-epistemic reasoning. Other semantics for PASP were proposed in [23] based on the idea of pstable models [24]. Pstable models are a framework characterized by a fusion of ASP and paraconsistent logic. Such pstable models are closer to the intuition of classical models and possibilistic logic than they are to stable models, as in our approach. Indeed, the focus of pstable models is more on handling inconsistency. For instance,

the program containing the rule $(\mathbf{c}: a \leftarrow not\ a)$ has (a, c) as its unique possibilistic pstable model, which is not compatible with a reading of '*not a*' as "*it cannot be established that a is certain*".

Many probabilistic extensions of logic programming have also been considered. One of the first generalizations of propositional logic based on probability theory is [25]. In this work, a logic is defined in terms of probability distributions over possible worlds, where the probability attached to a formula corresponds with the probability that the real world is among those that make the formula true. This idea was later extended to probabilistic logic programming [26], where maximum entropy takes on a role that is very similar to the role of minimal specificity, as shown in the transformation from [27]. Similar work on combining probability theory with logic programming had already been done in [28]. Indeed, [28] is one of the earliest works where, in the setting of probabilistic deductive databases, probability theory is combined with non-monotonic negation. While many works exist that combine probability theory and logic programming in general, only few have tried to combine probability theory and ASP. One of the most notable exceptions is the work from [29], where probabilistic atoms are used to encode the probability that an associated random variable will take on a given value.

Thus far we only considered frameworks which either use possibility theory or probability theory to deal with uncertainty. Bayesian Logic Programming [30], where a generalization of Bayesian networks is used to reason over Horn clauses, can also be used to deal with uncertainty. In particular, Bayesian networks employ well-understood Bayesian models for representing joint probabilities and offer a good graphical representation of local influences. Other popular approaches for dealing with uncertainty include Markov Logic Networks [31], where first-order logic is used to compactly specify a Markov Network and as such allow for uncertain inference. Markov Logic Networks make it easy to specify interactions between random variables, and allow for dealing with cyclic dependencies. However, inference in Markov Logic Networks is often computationally quite complex. Furthermore, the weights attached to the formulas in Markov Logic Networks tend to be counterintuitive in that their influence in the network as a whole is not immediately obvious.

From a practical point of view, being able to deal with uncertainty plays an important (though often implicit) role in economics and in dealing with preferences, handling inconsistencies and dealing with weak constraints. Indeed, the uncertainty of costs [32] is an important problem, where factors such

as demand, production and actual costs are all pervaded by uncertainty [33]. Unsurprisingly, this is a very active domain where probability theory plays a significant role [34]. Preferences, on the other hand, are an important topic within the ASP community. For example, in [35] ordered logic programs are used to deal with preferences. Ordered logic programs assume a partial order among rules, allowing less important rules to be violated in order to satisfy rules with higher importance. In some sense, the use of such preferences among rules is related to using certainty weights, although the resulting semantics are closer in spirit to the approach from [3] than to the semantics we have developed throughout this paper. Quite a number of other works also deal with preference handling in non-monotonic reasoning; we refer to [36] for a thorough overview. Weak constraints [37] are yet another example of a problem that can be seen as a problem of preferences amongst rules. Indeed, weak constraints are constraints that we try to apply, while we are still willing to violate such constraints if applying the constraint would otherwise prevent us from finding an answer set. When dealing with inconsistencies in ASP, a number of different approaches can be taken. Indeed, approaches exist to highlight inconsistencies (e.g. [38]), to resolve inconsistencies (e.g. [39]) or to reason in inconsistent knowledge bases. Examples of the latter case include pstable models [24] and the semantics introduced in Section 3. Indeed, we have seen that the semantics from Section 3 are able to deal with inconsistencies in settings where the other semantics for PASP [3, 4] are not.

We noted in Section 5 that a special case of the semantics presented in Section 3 are certain programs with optional rules. We discussed how some important problems in AI can be expressed in terms of programs with such optional rules. One such problems is cautious abductive reasoning. To the best of our knowledge, the approach presented in this paper is the first implementation of cautious abductive reasoning. Brave abductive reasoning, on the other hand, has seen a myriad of implementations and many solvers for answer set programming, e.g. [40, 41], have incorporated very performant mechanisms for brave abductive reasoning. Interestingly, [42] illustrated how abductive reasoning can benefit from possibility theory, allowing to order the possible cautious explanations. Also of interest is the work from [43] on abduction in multi-adjoint logic programs. In multi-adjoint logic programs, it is possible to associate confidence factors the rules. Furthermore, it allows to specify for each rule the type of the implication, i.e. the residual implicators induced by the Gödel, Łukasiewicz or the product t-norm. Still, no practical implementation has been suggested for this particular type of abduction.

Conformant planning, which is also known under other names such as secure planning and strong planning, has also seen a lot of interest. For a fixed plan length, conformant planning is a Σ_3^{P}-complete problem and secure checking, i.e. verifying whether a plan is a secure plan, is a Π_2^{P}-complete problem. If we restrict ourselves to proper planning domains, then conformant planning and secure checking is Σ_2^{P}-complete and coNP-complete, respectively. Many implementations of conformant planning exist, including $\mathcal{C}$-Plan [44], CMBT [45], Conformant-FF [46], and DLV$^{\mathcal{K}}$ [15], where the latter is an ASP-based approach. The latter approach, in particular, is an ASP-based approach in which the planning problem is expressed in the action language $\mathcal{K}$. Contrary to our approach, however, these implementations cannot readily be extended to deal with certainties.

Finally, the combination of ASP with weights has been used for reasons other than the modelling of incomplete information. Fuzzy ASP (FASP) [47] is a generalization of ASP where the truth of literals is graded, i.e. literals are no longer strictly true or false. In FASP, there is considerable flexibility w.r.t. the interpretation of the connectives used in ASP (i.e. negation, conjunction, disjunction, implication). All connectives map the values from $[0, 1]$ onto values from $[0, 1]$, agree with the classical connectives and adhere to specific properties, e.g. conjunction needs to be a monotonic, symmetric and associative operator. In addition, extensions of FASP have been considered, e.g. aggregated FASP [47], that further extend the modeling power of FASP. Clearly, FASP is a powerful tool for the treatment of gradual information in the presence of complete information. A core language for FASP was introduced in [48], which highlights that a fairly simple and compact variant of FASP is sufficiently expressive to simulate a wide variety of FASP formalisms. At the same time, it is easier to reason about and implement such a core language. Current implementations of FASP translate programs to instances of other formalisms such as bilevel programming [49, 50]. Other authors have looked at fixpoint operators for finding upper and lower bounds on the truth value of atoms [51, 52], which can be useful as a preprocessing step for exact solvers, as an approximation of exact solvers, or as en exact method for restricted classes of FASP programs. While FASP is a framework to deal with multi-valuedness, it is not designed to model uncertainty. In fact, both multi-valuedness and uncertainty are perpendicular problems and can be combined in a single framework to deal with gradual truths in the presence of incomplete information, e.g. [53, 54].

7. Conclusions

In this paper we contrasted two different semantics for PASP based on the idea that an ASP program can be seen as a means to reason over the epistemic states of an agent, where each answer set corresponds with an epistemic state. When extending this idea to PASP, the weights attached to the rules can be treated in two dual ways. On the one hand, weighted epistemic states can be considered where the weight attached to each rule reflects the certainty an agent would have in the conclusion of the rule, knowing that the body is satisfied. In other words, weighted rules are interpreted as rules with uncertain conclusions. This view was proposed in [4] and was formalized in [4] by describing the semantics of weighted rules in terms of constraints on possibility distributions.

Alternatively, we can maintain crisp epistemic states and use the weights associated with rules to express that some epistemic states are more compatible with available meta-knowledge than others. This is the view developed in this paper. Given this understanding of a PASP program, we treat the weight attached to each rule as the certainty that the rule is valid. As such, weighted rules are seen as rules whose validity is uncertain. We showed how treating a PASP program like this comes down to an efficient encoding of a possibility distribution over the exponentially many subprograms of an ASP program. This gives rise to four distinct types of inference, for which we have examined the computational complexity. We find that two of these inference types are as complex as the corresponding inference types in classical ASP, while the complexity of the other two inference types goes up one level in the polynomial hierarchy. We showed that all inference tasks can be simulated using classical ASP. In addition, we discussed how the concept of optional rules is a special case of the semantics presented in this paper. Important problems in AI, including cautious abductive reasoning and conformant planning, can be expressed in terms of programs with optional rules.

Finally, other feasible options exist to encode the possibility distribution of an exponential number of subprograms. Possibilistic networks, the possibilistic counterpart of Bayesian networks, offer another approach to encode the possibility distribution. In particular, using possibilistic networks, we would be able to able to sidestep the implicit assumption made in this paper that the validity of rules is independent of the validity of other rules. This could be useful in a number of practical scenarios, although it would not require any conceptual changes in the semantics that was developed in this

paper. For example, some rules may encode information obtained from a single source; finding that one of the rules is invalid may increase our doubt over the certainty of the other rules. Also, some of the rules may encode special cases of a particular rule. For example, some rules may classify a bird as a penguin, and others may classify birds as royal penguins when they have yellow ornaments. Finding that the basic rule is invalid (since, obviously, not all birds are penguins) may affect the certainty of the special cases. While using possibilistic networks shows great potential, a full treatment of this topic is left for future work.

Proofs

Proposition 1. *Let P be a simple possibilistic ASP program. For each literal 'l' we have that $N(P \models^{\mathrm{c}} l) \geq \lambda$ iff $V(l) \geq \lambda$ with V the possibilistic answer set of P under the semantics from Bauters [4], which in turn coincides with the semantics from Nicolas [3].*

Proof. For each literal 'l' we have that $N(P \models^{\mathrm{c}} l) \geq \lambda$ iff $V(l) \geq \lambda$ with V the possibilistic answer set of P under the semantics from [3] and as given in Section 2.3. To see this, note that $V(l) \geq \lambda$ iff l is in the unique answer set V_λ of the λ-cut of P. Indeed, we can only conclude $V(l) \geq \lambda$ if we have only used rules with associated weights equal or greater than λ to deduce 'l'. Similarly, $N(P \models^{\mathrm{c}} l) \geq \lambda$ iff there does not exist some subprogram P' such that $\pi_P(P') > 1-\lambda$ for which we have that $P' \not\models^{\mathrm{c}} l$. But, the only subprograms for which $\pi_P(P') > 1-\lambda$ are exactly those subprograms from which we did not remove any rules with associated weights equal or greater than λ. Thus we have that l can be deduced from every subprogram P' with $\pi_P(P') > 1 - \lambda$. In particular, because of the monotonicity of inference of simple programs, it suffices to consider the subprogram that corresponds exactly with the λ-cut of P. Hence $N(P \models^{\mathrm{c}} l) \geq \lambda$ iff $V(l) \geq \lambda$. Finally, [4] proved that, for simple programs, the semantics of [3] and [4] coincide. Thus, the result also holds for V the possibilistic answer set of P under the semantics from [4]. □

Proposition 2. *Let P be a possibilistic normal program. Deciding whether*

$$\Pi\left(P \models^{\mathrm{b}} l\right) \geq \lambda \text{ is } \mathsf{NP}\text{-complete};$$
$$N\left(P \models^{\mathrm{c}} l\right) \geq \lambda \text{ is } \mathsf{coNP}\text{-complete};$$
$$\Pi\left(P \models^{\mathrm{c}} l\right) \geq \lambda \text{ is } \Sigma_2^{\mathsf{P}}\text{-complete};$$
$$N\left(P \models^{\mathrm{b}} l\right) \geq \lambda \text{ is } \Pi_2^{\mathsf{P}}\text{-complete}.$$

Proof. Part 1: deciding whether $\Pi\left(P \models^{\mathrm{b}} l\right) \geq \lambda$ is NP-complete.
(membership) To determine whether $\Pi\left(P \models^{\mathrm{b}} l\right) \geq \lambda$ we need to guess a subset P' of rules from P such that $\pi_P(P') \geq \lambda$ and an partial Herbrand interpretation M which includes 'l'. Given such a non-deterministic guess, we can verify in polynomial time whether M is indeed an answer set and hence whether $P'^{*} \models^{\mathrm{b}} l$. Hence determining whether $\Pi\left(P \models^{\mathrm{b}} l\right) \geq \lambda$ is in NP.

(hardness) NP-hardness follows trivially from the NP-hardness of brave reasoning for classical normal programs. □

Proof. Part 2: deciding whether $N\,(P \models^{\mathrm{c}} l) \geq \lambda$ *is* coNP*-complete.*
(membership) We will show that the complementary problem is in NP. To determine whether $N\,(P \models^{\mathrm{c}} l) \not\geq \lambda$ we guess a subset P' of rules from P such that $\pi_P(P') > 1 - \lambda$ and a consistent partial Herbrand interpretation M, which does not include 'l'. Given such a non-deterministic guess, we can verify in polynomial time that M is an answer set of P'^* and hence that $P'^* \not\models^{\mathrm{c}} l$. We know that $N\,(P \models^{\mathrm{c}} l) = 1 - \max\{\pi_P(Q) \mid Q \subseteq P \text{ and } Q^* \not\models^{\mathrm{c}} l\} \leq 1 - \pi_P(P') < \lambda$ from Definition 5. In other words: determining whether $N\,(P \models^{\mathrm{c}} l) \not\geq \lambda$ is in NP. Deciding whether $N\,(P \models^{\mathrm{c}} l) \geq \lambda$ is thus in coNP.
(hardness) The coNP-hardness follows trivially from the coNP-hardness of cautious reasoning for classical normal programs. □

Proof. Part 3: deciding whether $\Pi\,(P \models^{\mathrm{c}} l) \geq \lambda$ *is* Σ_2^{P}*-complete.*
(membership) To determine whether $\Pi\,(P \models^{\mathrm{c}} l) \geq \lambda$ we need to guess a subset P' of rules from P such that $\pi_P(P') \geq \lambda$. We cannot immediately guess a partial Herbrand interpretation to determine whether $P'^* \models^{\mathrm{c}} l$ since this requires that 'l' is true in every single partial Herbrand interpretation. Given a non-deterministic guess of P', however, we can rely on an NP-oracle [7] to verify in constant time whether $P'^* \models^{\mathrm{c}} l$, as P'^* is a classical normal program. Hence determining whether $\Pi\,(P \models^{\mathrm{c}} l) \geq \lambda$ is in $\mathsf{NP}^{\mathsf{NP}}$, i.e. in Σ_2^{P}.
(hardness) We reduce the problem of determining the satisfiability of a QBF of the form $\phi = \exists X_1 \forall X_2 \cdot p(X_1, X_2)$ with $p(X_1, X_2)$ in DNF, i.e. of the form $\theta_1 \vee ... \vee \theta_n$ with each θ_i a conjunction of literals, to the problem of deciding whether $\Pi\,(P \models^{\mathrm{c}} l) \geq \lambda$. We define the possibilistic normal program P_ϕ corresponding to ϕ as

$$\begin{aligned} P_\phi = &\{\mathbf{0.5}\colon x \leftarrow \mid x \in X_1\} \cup \{\mathbf{0.5}\colon \neg x \leftarrow \mid x \in X_1\} && (22)\\ &\cup \{\mathbf{1}\colon x \leftarrow \mathit{not}\ \neg x \mid x \in X_2\} \\ &\cup \{\mathbf{1}\colon \neg x \leftarrow \mathit{not}\ x \mid x \in X_2\} && (23)\\ &\cup \{\mathbf{1}\colon \mathit{sat} \leftarrow \theta_t \mid 1 \leq t \leq n\} && (24) \end{aligned}$$

where we identify the conjunction of literals θ_t in (24) with a set of literals. We now show that the QBF is satisfiable if and only if $\Pi\,(P_\phi \models^{\mathrm{c}} \mathit{sat}) \geq 0.5$.

The rules in (22) ensure that there are as many subprograms $P' \subseteq P_\phi$ as there are interpretations of X_1. The subprograms P' with $\pi_{P_\phi}(P') > 0$ then contain the rules (23) that generate as many answer sets as there are interpretations of X_2. The rules from (24) ensure that '*sat*' is contained in the classical answer set whenever for a chosen interpretation of X_1 and X_2

it holds that $p(X_1, X_2)$ is satisfied. Notice that the certainty attached to the rules ensures that removing any of the rules from (23) or (24) results in $\pi_{P_\phi}(P') = 0$, i.e. it indicates that these rules are completely necessary.

We then have that $\Pi\left(P_\phi \models^{\mathrm{c}} sat\right) \geq 0.5$ if and only if the QBF is satisfiable. Indeed, from the construction of P_ϕ, and in particular from the rules (22), we know that for every interpretation of X_1 there will be a corresponding consistent subprogram for which the possibility is 0.5. Also, $P'^* \models^{\mathrm{c}} sat$ if and only if P' corresponds to an interpretation of X_1 such that $p(X_1, X_2)$ is consistent for every interpretation of X_2. Using the consistent possibility measure (i.e. finding $\max\left\{\pi_{P_\phi}(P') \mid P' \subseteq P_\phi \text{ and } P'^* \models^{\mathrm{c}} sat\right\}$) implies that the QBF is satisfied whenever we find at least one such an interpretation X_1.

Some of the subprograms of P_ϕ may either be inconsistent subprograms or may correspond to partial interpretations of X_1. However, the inconsistent subprograms P' have $\pi_{P_\phi}(P') = 0$ by definition and can therefore never be used to derive $\Pi\left(P_\phi \models^{\mathrm{c}} sat\right) \geq 0.5$. Furthermore, any subprogram P' with incomplete assignments for the variables in X_1 from which we can conclude that $P'^* \models^{\mathrm{c}} sat$ can trivially be extended to a subprogram P'' to which we add some rules from (22) to complete the assignment for the variables in X_1 and we will still be able to conclude that $P''^* \models^{\mathrm{c}} sat$. □

Proof. Part 4: deciding whether $N\left(P \models^{\mathrm{b}} l\right) \geq \lambda$ *is* Π_2^{P}*-complete.*
(membership) We will show that the complementary problem is in Σ_2^{P}. To determine whether $N\left(P \models^{\mathrm{b}} l\right) \not\geq \lambda$ we guess a subset P' of rules from P such that $\pi_P(P') > 1 - \lambda$. Given a non-deterministic guess for P', we rely on an NP-oracle [7] to verify in constant time that $P'^* \not\models^{\mathrm{b}} l$. Similar as in *Part 2* of this proof this gives us a counterexample for $N\left(P \models^{\mathrm{b}} l\right) \geq \lambda$. Hence determining whether $N\left(P \models^{\mathrm{b}} l\right) \geq \lambda$ is in $\mathsf{co}\left(\mathsf{NP}^{\mathsf{NP}}\right)$, i.e. in Π_2^{P}.
(hardness) Let Q be the program defined as $P \cup \{\mathbf{1}\colon x \leftarrow not\ l\}$ with x a fresh literal. Then $N\left(Q \models^{\mathrm{b}} l\right) \geq \lambda$ if and only if $\Pi\left(Q \models^{\mathrm{c}} x\right) \leq 1 - \lambda$. Indeed, we know from Definition 5 that $N\left(Q \models^{\mathrm{b}} l\right) \geq \lambda$ is true whenever we have that $1 - \max\left\{\pi_Q(P') \mid P' \subseteq Q \text{ and } P'^* \not\models^{\mathrm{b}} l\right\} \geq \lambda$, i.e. whenever we have that $\max\left\{\pi_Q(P') \mid P' \subseteq Q \text{ and } P'^* \not\models^{\mathrm{b}} l\right\} \leq 1 - \lambda$. Because the newly added rule $(x \leftarrow not\ l)$ will be in every subprogram P' with $\pi_Q(P') > 0$ (since the certainty attached to this rule is 1), we know that there is at least one answer set in which l is true if and only if it is not the case that x is true in every answer set. Thus, the previous inequality is equivalent to $\max\left\{\pi_Q(P') \mid P' \subseteq Q \text{ and } P'^* \models^{\mathrm{c}} x\right\} \leq 1 - \lambda$ and, by applying Definition 5,

to $\Pi\left(Q \models^{\mathrm{c}} x\right) \leq 1-\lambda$. Since the set of certainty values associated with the rules is finite, this equation is equivalent to $\Pi\left(Q \models^{\mathrm{c}} x\right)<\lambda'$ for some λ'. Hence we have that $\neg(\Pi\left(Q \models^{\mathrm{c}} x\right) \geq \lambda')$. This problem is therefore the complement of the decision problem from *Part 3* of this proof. Thus deciding whether $N\left(P \models^{\mathrm{b}} l\right) \geq \lambda$ is Π_2^{P}-hard. □

Proposition 3. *Let P be a possibilistic disjunctive program. Deciding whether*

$$\Pi\left(P \models^{\mathrm{b}} l\right) \geq \lambda \text{ is } \Sigma_2^{\mathsf{P}}\text{-complete};$$
$$N\left(P \models^{\mathrm{c}} l\right) \geq \lambda \text{ is } \Pi_2^{\mathsf{P}}\text{-complete};$$
$$\Pi\left(P \models^{\mathrm{c}} l\right) \geq \lambda \text{ is } \Sigma_3^{\mathsf{P}}\text{-complete};$$
$$N\left(P \models^{\mathrm{b}} l\right) \geq \lambda \text{ is } \Pi_3^{\mathsf{P}}\text{-complete}.$$

Proof. Part 1: deciding whether $\Pi\left(P \models^{\mathrm{b}} l\right) \geq \lambda$ is Σ_2^{P}-complete.
(membership) Analogous to the proof in *Part 1* of the proof of Proposition 2, where we are able to verify in constant time that M is an answer set of P^*, which now is a positive possibilistic program, by using an NP-oracle.
(hardness) Analogous to the proof of the hardness in *Part 1* of the proof of Proposition 2. □

Proof. Part 2: deciding whether $N\left(P \models^{\mathrm{c}} l\right) \geq \lambda$ is Π_2^{P}-complete.
Entirely analogous to the proof in *Part 2* of the proof of Proposition 2. □

Proof. Part 3: deciding whether $\Pi\left(P \models^{\mathrm{c}} l\right) \geq \lambda$ is Σ_3^{P}-complete.
(membership) Analogous to the membership proof in *Part 3* of the proof of Proposition 2, but where we now require a Σ_2^{P}-oracle to verify in constant time whether for P'^*, with P'^* being a classical disjunctive program, we have that $P'^* \models^{\mathrm{c}} l$.
(hardness) Analogous to the hardness proof in *Part 4* of the proof of Proposition 2, where we can now reduce the complement of this problem to an instance of the decision problem from *Part 4* of this proof. □

Proof. Part 4: deciding whether $N\left(P \models^{\mathrm{b}} l\right) \geq \lambda$ is Π_3^{P}-complete.
(membership) Analogous to the proof of membership proof in *Part 4* of the proof of Proposition 2.
(hardness) We reduce the problem of determining the satisfiability of a QBF of the form $\psi = \forall X_1 \exists X_2 \forall X_3 \cdot p(X_1, X_2, X_3)$ with $p(X_1, X_2, X_3)$ in DNF, i.e. of the form $\theta_1 \vee ... \vee \theta_n$ with each θ_i a conjunction of literals, to the problem

of deciding whether $N\left(P \models^{\mathrm{b}} l\right) \geq \lambda$. We define the possibilistic disjunctive program P_ψ corresponding to ψ as

$$
\begin{aligned}
P_\psi = & \{\mathbf{1}: x; x' \leftarrow \mid x \in X_3\} && (25)\\
& \cup \{\mathbf{0.5}: x \leftarrow \mid x \in X_1\} \cup \{\mathbf{0.5}: \neg x \leftarrow \mid x \in X_1\} && (26)\\
& \cup \{\mathbf{1}: x \leftarrow not\ \neg x \mid x \in X_2\} && \\
& \cup \{\mathbf{1}: \neg x \leftarrow not\ x \mid x \in X_2\} && (27)\\
& \cup \{\mathbf{1}: sat \leftarrow \theta'_t \mid 1 \leq t \leq n\} && (28)\\
& \cup \{\mathbf{1}: x \leftarrow sat \mid x \in X_3\} \cup \{\mathbf{1}: x' \leftarrow sat \mid x \in X_3\} && (29)
\end{aligned}
$$

where we identify the conjunction of literals θ'_t in (28) with a set of literals and we have furthermore replaced all negative literals of the form $\neg x$ by a fresh atom of the form x' for every $x \in X_3$. We now show that the QBF is satisfiable if and only if $N\left(P_\psi \models^{\mathrm{b}} sat\right) \geq 0.75$.

The rules in (26) ensure that there are at least as many subprograms $P' \subseteq P_\psi$ as there are interpretations of X_1. Furthermore, the subprograms P' with $\pi_{P_\psi}(P') > 0$ contain the rules (25) and (27), which generate as many answer sets as there are interpretations of $(X_2 \cup X_3)$. The rule (28) ensures that '*sat*' is contained in the classical answer set whenever for a chosen interpretation of X_1, X_2 and X_3 it holds that $p(X_1, X_2, X_3)$ is satisfied. Notice that the certainty attached to the rules ensures that removing any of the rules from (25), (27), (28) or (29) results in $\pi_{P_\psi}(P') = 0$, i.e. it indicates that these rules are completely necessary.

Thus far, we have not discussed the rules from (29). These rules work together with the rules from (25) to resolve the last $\forall$. Indeed, the rules from (29) implement a saturation technique [7] over a disjunctive program to ensure that *sat* will only be true in an answer set when $p(X_1, X_2, X_3)$ is satisfied for every interpretation of X_3, given some interpretation of (X_1, X_2). In particular, let P' be a subprogram of P_ψ with $\pi_{P_\psi}(P') > 0$, and let M be an answer set of P' that contains *sat*. Because of the rules from (26) and (27) in P', M contains literals corresponding to the variables of X_1 and X_2, and as such defines an interpretation of X_1 and X_2. Furthermore, because of the saturation rules (29), M contains the literals x and x' for every $x \in X_3$. Now suppose that there would exist a partial Herbrand interpretation M' of P' that contains the same literals as M corresponding to the variables of X_1 and X_2 but that does not contain *sat*, then we have that $M' \subset M$. Furthermore, since M' contains the same literals as M corresponding to the variables of

X_2, $(P')^M = (P')^{M'}$. If M' would be an answer set of P', then by definition it would be a minimal partial Herbrand model of $(P')^M$. This, together with $M' \subset M$, would contradict the fact that M is an answer set of P'. Hence M' is not an answer set of P'. We conclude that when *sat* is contained in an answer set M of P', then *sat* is contained in all answer sets of P' that contain the same literals corresponding to the variables of X_1 and X_2 as M, regardless of which choice is made by rules (25) for the literals corresponding to the variables of X_3, i.e. regardless of the interpretation of X_3.

We then have that the QBF is satisfied iff $N\left(P_\psi \models^{\mathrm{b}} sat\right) \geq 0.75$. Indeed, from the construction of P_ψ, and in particular from the rules (26), we know that for every interpretation of X_1 there will be a corresponding consistent subprogram P' for which the possibility is 0.5. Also, $P'^* \models^{\mathrm{b}} sat$ if and only if P' has an answer set such that $p(X_1, X_2, X_3)$ is satisfied for every interpretation of X_3. Using the necessity measure (i.e. finding $\min\left\{1 - \pi_{P_\psi}(P') \mid P' \subseteq P_\psi \text{ and } P'^* \not\models^{\mathrm{b}} sat\right\} \geq 0.75$), it then holds that the QBF is satisfied for every interpretation of X_1. Finally, note that we need to consider a necessity strictly greater than 0.5, e.g. 0.75. Indeed, we have

$$
\begin{aligned}
& N(P_\psi \models^{\mathrm{b}} sat) \geq 0.75 \\
\equiv\ & \min\left\{1 - \pi_{P_\psi}(P') \mid P' \subseteq P_\psi \text{ and } P'^* \not\models^{\mathrm{b}} sat\right\} \geq 0.75 \\
\equiv\ & \forall P', P' \subseteq P_\psi, P'^* \not\models^{\mathrm{b}} sat \cdot 1 - \pi_{P_\psi}(P') \geq 0.75 \\
\equiv\ & \forall P', P' \subseteq P_\psi, P'^* \not\models^{\mathrm{b}} sat \cdot \pi_{P_\psi}(P') \leq 0.75 \\
\equiv\ & \forall P', P' \subseteq P_\psi, P'^* \models^{\mathrm{b}} sat \cdot \pi_{P_\psi}(P') > 0.25.
\end{aligned}
$$

Furthermore, the possibility associated with each subprogram P' is $\pi_P(P') \in \{0, 0.5, 1\}$. Hence, by verifying $N(P_\psi \models^{\mathrm{b}} sat) \geq 0.5$, we have only verified that *sat* is a brave conclusion of those subprograms P' with $\pi_P(P') = 1$, whereas we want to verify for those subprograms P' with $\pi_P(P') = 0.5$ whether $P' \models^{\mathrm{b}} sat$, i.e. we need verify whether $N\left(P_\psi \models^{\mathrm{b}} sat\right) \geq \lambda$ for an arbitrary lambda in $]0.5, 1]$.

Some of the subprograms of P_ψ may either be inconsistent subprograms or may correspond to partial interpretations of X_1. However, the inconsistent subprograms P' have $\pi_{P_\psi}(P') = 0$ by definition and can therefore never be used to derive $N\left(P_\psi \models^{\mathrm{b}} sat\right) \geq 0.75$ (also, we already established that for every interpretation of X_1 there will be a corresponding consistent subprogram P'). Furthermore, any subprogram P' with incomplete assignments for the variables in X_1 from which we can conclude that $P'^* \models^{\mathrm{b}} sat$ can trivially

be extended to a subprogram P'' to which we add some rules from (26) to complete the assignment for the variables in X_1 and we will still be able to conclude that $P''^* \models^{\mathrm{b}} sat$. Thus, these additional subprograms do not affect our ability to derive $N\left(P_\psi \models^{\mathrm{b}} sat\right) \geq 0.75$. □

Proposition 4. *Let P be a possibilistic disjunctive program and $P^{\Pi}_{brave}(l,\lambda)$ the classical disjunctive program as defined in Definition 8. We have that $\Pi(P \models^{\mathrm{b}} l) \geq \lambda$ iff $P^{\Pi}_{brave}(l,\lambda)$ has a classical consistent answer set.*

Proof. We want to determine whether $\Pi\left(P \models^{\mathrm{b}} l\right) \geq \lambda$. By Definition 5 we know this is equivalent to $\max\left\{\pi_P(P') \mid P' \subseteq P \text{ and } P'^* \models^{\mathrm{b}} l\right\} \geq \lambda$, or, determining whether there exists some subprogram $P' \subseteq P$ with $P'^* \models^{\mathrm{b}} l$ such that $\pi_P(P') \geq \lambda$. Since we want $\pi_P(P') \geq \lambda$, this implies that $P_{\mathrm{req}} \subseteq P'^*$ with $P_{\mathrm{req}} = \{r \mid (r,c) \in P, c > 1-\lambda\}$. Thus the problem reduces to determining whether for some set of rules $P_{\mathrm{opt}} \subseteq \{r \mid (r,c) \in P, c \leq 1-\lambda\}$ we have $P'^* = (P_{\mathrm{req}} \cup P_{\mathrm{opt}})$ such that $P'^* \models^{\mathrm{b}} l$. By construction of $P_{\mathrm{basic}}(\lambda)$, in particular due to the rules (4), we know that every rule in P_{req} is chosen. Furthermore, every choice made in (3) corresponds with a choice of P_{opt}. This choice P_{opt}, along with the rules P_{req}, will be applied by the rules in P_{basic} due to the rules (5). Finally, the addition of the rule $\{\leftarrow \mathit{not}\ l\}$ ensures that 'l' must be a conclusion of some answer set of the simulation $P^{\Pi}_{\mathrm{brave}}(l,\lambda)$, or otherwise $P^{\Pi}_{\mathrm{brave}}(l,\lambda)$ will not have any answer sets. Clearly, then $\Pi\left(P \models^{\mathrm{b}} l\right) \geq \lambda$ when $P^{\Pi}_{\mathrm{brave}}(l,\lambda)$ has a classical consistent answer set. □

Proposition 5. *Let P be a possibilistic disjunctive program, $\lambda > 0$ and $P^{N}_{cautious}(l,\lambda)$ the classical disjunctive program as defined in Definition 9. We have that $N(P \models^{\mathrm{c}} l) \geq \lambda$ iff $P^{N}_{cautious}(l,\lambda)$ has no classical consistent answer set.*

Proof. Analogous to the proof in Proposition 4. □

Proposition 6. *Let P be a possibilistic normal program and $P^{\mathrm{c}}_{\Pi}(l,\lambda)$ the disjunctive program defined as $P_{complex}(\lambda) \cup \{sat \leftarrow l\}$. Then $\Pi\left(P \models^{\mathrm{c}} l\right) \geq \lambda$ iff $P^{\mathrm{c}}_{\Pi}(l,\lambda)$ has a classical answer set.*

Proof. We want to determine whether $\Pi\left(P \models^{\mathrm{c}} l\right) \geq \lambda$, i.e. whether there exists a $P' \subseteq P$ such that $P'^* \models^{\mathrm{c}} l$ and $\pi_P(P') \geq \lambda$. The latter condition means that $(r,c) \in P'$ for every $(r,c) \in P$ with $c > 1-\lambda$. Similar as in Proposition 4, the rules in (6) and (7) generate as many answer sets as there are subprograms $P' \subseteq P$ for which $\pi_P(P') \geq \lambda$.

For each such subprogram P' we want to determine whether P'^* has 'l' as is a cautious conclusion. By construction, $\{cl \leftarrow \mid cl \in cls(P^r)\}$ is equivalent to P^r. In particular, every model of these rules corresponds to an answer set of P^r. Since we removed classical negation in (8), however, we need to add the rules in (9) to ensure that 'sat' is contained in the answer set whenever 'a' and the opposite atom 'na' are true at the same time. The intuition of making 'sat' true is thus to indicate that this is not a valid answer set of the subprogram P'. The rule $(l \leftarrow sat)$ is used to try to make 'l' false, by once again ensuring that 'sat' is contained in the answer set whenever 'l' is in the answer set. Intuitively, we thus say that an answer set in which 'l' is true is undesirable, i.e. we prefer answer sets of P' in which 'l' is false. The rule (11) is then used to block all answer sets in which 'sat' is false. In other words: unless for every answer set of P' we have that 'l' is true in the answer set, we have not found that 'l' is a cautious consequence of P'.

Thus far we have not discussed the use of the rules (10). Together with the atom 'sat', these rules are used to implement a saturation technique [7] over our disjunctive simulation and we refer to this work for a detailed overview of how saturation works. The intuition of saturation is that we use the property that an answer set is a *minimal* model. In particular, the rules in (10) will add all the atoms under consideration to the model M to try and prevent it from being an answer set. Indeed, if we find a model $M' \subseteq M$ then clearly M cannot be an answer set. As such, we can ensure that consistent models of P' are preferred over inconsistent models, and that models of P' in which 'l' is false are preferred over models in which 'l' is true. Then, only if no consistent answer set (in which 'l' is false) exists for P', will we have that 'sat' is true in an answer set of $P^{\mathrm{c}}_{\Pi}(l, \lambda)$.

Finally, when a subprogram P' is inconsistent, then $\pi(P') = 0$, i.e. we do not want to consider this subprogram. Notice, however, that the rule (9) would not work as expected in this case. Indeed, if P' is inconsistent is does not have a consistent model and the saturation technique would not exclude this subprogram. As such, we repeat our simulation of the subprogram P' in (12) and use constraints in (13) to effectively block inconsistent subprograms. □

Proposition 7. *Let P be a possibilistic normal program and $P^{\mathrm{b}}_{\mathrm{N}}(l, \lambda)$ the disjunctive program defined as $P_{complex}(1 - \lambda') \cup \{sat \leftarrow not\ l\}$ with λ' defined as in Proposition 5. Then $N\left(P \models^{\mathrm{b}} l\right) \geq \lambda$ iff $P^{\mathrm{b}}_{\mathrm{N}}(l, \lambda)$ has no classical answer set.*

Proof. This proof is analogous to the proof of Proposition 6, similar as how the proof of Proposition 5 was analogous to the proof of Proposition 4. □

Proposition 8. *Let P_{abd} be the possibilistic normal program defined for an abductive diagnosis program $\langle H, T, O\rangle$ as*

$$\{\mathbf{0.5} : block\_h \leftarrow \mid h \in H\} \tag{14}$$
$$\cup \{\mathbf{1} : h \leftarrow not\ block\_h \mid h \in H\} \tag{15}$$
$$\cup \{\mathbf{1} : goal \leftarrow O\} \tag{16}$$
$$\cup \{\mathbf{1} : r \mid r \in T\}\,. \tag{17}$$

It holds that $\langle H, T, O\rangle$ has a cautious explanation iff $\Pi\,(P_{\text{abd}} \models^{\text{c}} goal) \geq 0.5$. In particular, we have that E is a cautious explanation if and only if we have for $P' = P_{\text{abd}} \setminus \{block\_h \leftarrow \mid h \in E\}$ that $P' \models^{\text{c}} goal$.

Proof. For $\Pi\,(P_{\text{abd}} \models^{\text{c}} goal) \geq 0.5$, we must have some $P' \subseteq P_{\text{abd}}$ such that $P' \models^{\text{c}} goal$ and $\pi(P') \geq 0.5$. Thus, clearly, all the rules defined in (15), (16) and (17) must be in P'. It is furthermore easy to see that for every $h \in H \setminus E$ we have that $(h \leftarrow) \in ((P')^*)^M$ for every answer set M of P'^* and, since $P'^* \models^{\text{c}} goal$, that E is a cautious explanation. □

Proposition 9. *Let P_{con} be the possibilistic normal program defined for a conformant planning problem with the atom 'goal' the desired goal fluent. We express the domain knowledge as a normal ASP program $P_{\text{act}} \cup P_{\text{rem}}$. Then P_{con} is:*

$$\{\mathbf{0.5} : block\_i \leftarrow \mid r_i \in P_{\text{act}}\} \tag{18}$$
$$\cup \{\mathbf{1} : H(r_i) \leftarrow B(r_i) \cup \{not\ block\_i\} \mid r_i \in P_{\text{act}}\} \tag{19}$$
$$\cup \{\mathbf{1} : r \mid r \in P_{\text{rem}}\} \tag{20}$$
$$\cup \{\mathbf{1} : \leftarrow not\ goal\} \tag{21}$$

A conformant plan exists iff $\Pi\,(P_{\text{con}} \models^{\text{c}} goal) \geq 0.5$.

Proof. When $\Pi\,(P_{\text{con}} \models^{\text{c}} goal) \geq 0.5$ then, by definition, there exists a subprogram $P' \subseteq P$ such that $(P')^* \models^{\text{c}} goal$ with $\pi_P(P') \geq 0.5$. Since $\pi_P(P') \geq 0.5$ we know that all the rules from (19), (20) and (21) are in P'. Thus, only tules from (18) may be in $P \setminus P'$. In that case, the corresponding rule from (19) ensures that for every answer set M of $(P')^*$ we have that $(H(r_i) \leftarrow B(r_i)) \in ((P')^*)^M$. Thus, the action is no longer blocked and can

be applied. Because of the available actions we can, regardless of the initial state described in (20), cautiously derive '*goal*'. Indeed, otherwise we know due to (21) that M is not be a model. In other words: the choice made in (18) corresponds with a set of actions that form a cautious plan for the given planning problem. □

References

[1] Y. Zhang, Epistemic reasoning in logic programs, in: proceedings of the 20th international joint conference on Artifical intelligence (IJCAI'07), 2007, pp. 647–652.

[2] K. Bauters, S. Schockaert, M. De Cock, D. Vermeir, Weak and strong disjunction in possibilistic ASP, in: Proceedings of the 5th International Conference on Scalable Uncertainty Management (SUM'11), 2011, pp. 475–488.

[3] P. Nicolas, L. Garcia, I. Stéphan, C. Lefèvre, Possibilistic uncertainty handling for answer set programming, Annals of Mathematics and Artificial Intelligence 47 (1–2) (2006) 139–181.

[4] K. Bauters, S. Schockaert, M. De Cock, D. Vermeir, Possibilistic answer set programming revisited, in: Proceedings of the 26th Conference on Uncertainty in Artificial Intelligence (UAI'10), 2010, pp. 48–55.

[5] K. Bauters, S. Schockaert, M. De Cock, D. Vermeir, Possible and necessary answer sets of possibilistic answer set programs, in: Proceedings of the 24th International Conference on Tools with Artificial Intelligence (ICTAI), 2012, pp. 836–843.

[6] M. Gelfond, V. Lifzchitz, The stable model semantics for logic programming, in: Proceedings of the 5th Joint International Conference and Symposium on Logic Programming (ICLP'88), 1988, pp. 1081–1086.

[7] C. Baral, Knowledge, Representation, Reasoning and Declarative Problem Solving, Cambridge University Press, 2003.

[8] D. Dubois, J. Lang, H. Prade, Possibilistic logic, Handbook of Logic for Artificial Intelligence and Logic Programming 3 (1) (1994) 439–513.

[9] C. Papadimitriou, Computational complexity, Addison-Wesley, 1994.

[10] T. Janhunen, Representing normal programs with clauses, in: Proceedings of the 16th Eureopean Conference on Artificial Intelligence (ECAI'04), 2004, pp. 358–362.

[11] F. Lin, J. Zhao, On tight logic programs and yet another translation from normal logic programs to propositional logic, in: Proceedings of the 18th international joint conference on Artificial intelligence (IJCAI'03), 2003, pp. 853–858.

[12] I. Niemelä, P. Simons, Smodels – an implementation of the stable model and well-founded semantics for normal logic programs, in: Proceedings of the 4th International Conference on Logic Programming and Non-Monotonic Reasoning (LPNMR'97), Vol. 1265 of Lecture Notes in Artificial Intelligence, 1997, pp. 420–429.

[13] T. Eiter, G. Gottlob, N. Leone, Abduction from logic programs: Semantics and complexity, Theoretical Computer Science 189 (1–2) (1997) 129–177.

[14] T. Eiter, W. Faber, N. Leone, G. Pfeifer, A. Polleres, Planning under incomplete knowledge, in: Proceedings of the First International Conference on Computational Logic (CL'2000), 2000, pp. 807–821.

[15] T. Eiter, W. Faber, N. Leone, G. Pfeifer, A. Polleres, A logic programming approach to knowledge-state planning: Semantics and complexity, ACM Transactions on Computational Logic 5 (2) (2004) 206–263.

[16] D. Dubois, J. Lang, H. Prade, Towards possibilistic logic programming, in: Proceedings of the 8th Joint International Conference and Symposium on Logic Programming (ICLP'91), 1991, pp. 581–595.

[17] G. Wagner, Negation in fuzzy and possibilistic logic programs, in: Proceedings of the 2nd International Workshop on Logic Programming and Soft Computing (LPSC'98), 1998, pp. 113–128.

[18] D. Dubois, H. Prade, Can we enforce full compositionality in uncertainty calculi?, in: Proceedings of the 12th National Conference on Artificial Intelligence (AAAI'94), 1994, pp. 149–154.

[19] C. Chesñevar, G. Simari, T. Alsinet, L. Godo, A logic programming framework for possibilistic argumentation with vague knowledge, in: Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence (UAI'04), 2004, pp. 76–84.

[20] L. Amgoud, H. Prade, Reaching agreement through argumentation: A possibilistic approach, in: Proceedings of the 9th International Conference on Principles of Knowledge Representation and Reasoning (KR'04), 2004, pp. 175–182.

[21] J. C. Nieves, M. Osorio, U. Cortés, Semantics for possibilistic disjunctive programs, in: Proceedings of the 9th International Workshop on Logic Programming and Non-monotonic Reasoning (LPNMR'07), 2007, pp. 315–320.

[22] D. Dubois, H. Prade, S. Schockaert, Stable models in generalized possibilistic logic, in: Proceedings of the 13th International Conference on Principles of Knowledge Representation and Reasoning (KR'12), 2012, pp. 519–529.

[23] R. Confalonieri, J. C. Nieves, J. Vázquez-Salceda, Pstable semantics for logic programs with possibilistic ordered disjunction, in: Proceedings of the 11th International Conference on Advances in Artificial Intelligence (AI*IA'09), 2009, pp. 52–61.

[24] M. Osorio, J. A. N. Pérez, J. R. A. Ramírez, V. B. Macías, Logics with common weak completions, Journal of Logic and Computation 16 (6) (2006) 867–890.

[25] N. J. Nilsson, Probabilistic logic, Artificial Intelligence 28 (1) (1986) 71–87.

[26] T. Lukasiewicz, Probabilistic default reasoning with conditional constraints, Annals of Mathematics and Artificial Intelligence 34 (1–3) (2002) 35–88.

[27] D. Dubois, H. Prade, S. Sandri, On possibility/probability transformations, in: Proceedings of the 10th International Fuzzy Systems Association World Congress, 1993, pp. 103–112.

[28] R. Ng, V. Subrahmanian, Stable model semantics for probabilistic deductive databases, in: Proceedings of the 6th International Symposium on Methodologies for Intelligent Systems (ISMIS'91), Vol. 542, 1991, pp. 162–171.

[29] C. Baral, M. Gelfond, N. Rushton, Probabilistic reasoning with answer sets, Theory and Practice of Logic Programming 9 (1) (2009) 57–144.

[30] K. Kersting, L. De Raedt, Bayesian logic programs, CoRR.

[31] M. Richardson, P. Domingos, Markov logic networks, Machine Learning 62 (1-2) (2006) 107–136.

[32] E. Mills, Uncertainty and price theory, The Quarterly Journal of Economics 73 (1) (1959) 116–130.

[33] A. Sandmo, On the theory of the competitive firm under price uncertainty, The American Economic Review 61 (1) (1971) 65–73.

[34] P. Garvey, Probability Methods for Cost Uncertainty Analysis: A Systems Engineering Perspective, Taylor & Francis, 2000.

[35] D. Van Nieuwenborgh, D. Vermeir, Preferred answer sets for ordered logic programs, in: Proceeings of the European Conference on Logics in Artificial Intelligence, JELIA 2002, Cosenza, Italy, Vol. 2424 of Lecture Notes in Computer Science, 2002, pp. 432–443.

[36] J. P. Delgrande, T. Schaub, H. Tompits, K. Wang, A classification and survey of preference handling approaches in nonmonotonic reasoning, Computational Intelligence 20 (2) (2004) 308–334.

[37] F. Buccafurri, N. Leone, P. Rullo, Strong and weak constraints in disjunctive datalog, in: Proceedings of the 4th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'97), 1997, pp. 2–17.

[38] M. Gebser, J. Pührer, T. Schaub, H. Tompits, A meta-programming technique for debugging answer-set programs, in: Proceedings of the 23rd National Conference on Artificial Intelligence (AAAI'08), 2008, pp. 448–453.

[39] M. Balduccini, M. Gelfond, Logic programs with consistency-restoring rules, in: International Symposium on Logical Formalization of Commonsense Reasoning, AAAI 2003 Spring Symposium Series, 2003, pp. 9–18.

[40] N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, F. Scarcello, The dlv system for knowledge representation and reasoning, ACM Transactions on Computational Logic 7 (3) (2006) 499–562.

[41] M. Gebser, B. Kaufmann, R. Kaminski, M. Ostrowski, T. Schaub, M. Schneider, Potassco: The potsdam answer set solving collection, AI Communications 24 (2011) 107–124.

[42] D. Dubois, H. Prade, Fuzzy relation equations and causal reasoning, Fuzzy Sets and Systems 75 (2) (1995) 119–134.

[43] J. Medinaús, O.-M. Aciego, V. s, A multi-adjoint logic approach to abductive reasoning, in: Proceedings of the 17th International Conference on Logic Programming (ICLP'01), 2001, pp. 269–283.

[44] P. Ferraris, E. Giunchiglia, Planning as satisfiability in nondeterministic domains, in: Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI'00), 2000, pp. 748–753.

[45] A. Cimatti, M. Roveri, Conformant planning via symbolic model checking and heuristic search, Artificial Intelligence 159 (1–2) (2004) 127–206.

[46] J. Hoffmann, R. Brafman, Conformant planning via heuristic forward search: A new approach, Artificial Intelligence 170 (6-7) (2006) 507–541.

[47] D. V. Nieuwenborgh, M. De Cock, D. Vermeir, Fuzzy answer set programming, in: Proceedings of the 10th European Conference on Logics in Artificial Intelligence (JELIA'06), Vol. 4160 of Lecture Notes in Computer Science, 2006, pp. 359–372.

[48] J. Janssen, S. Schockaert, D. Vermeir, M. De Cock, A core language for fuzzy answer set programming, International Journal of Approximate Reasoning 53 (4) (2012) 660–692.

[49] S. Schockaert, J. Janssen, D. Vermeir, Fuzzy equilibrium logic: Declarative problem solving in continuous domains, ACM Transactions on Computational Logic (TOCL) 13 (4) (2012) 33:1–33:39.

[50] M. Blondeel, S. Schockaert, M. De Cock, D. Vermeir, NP-completeness of fuzzy answer set programming under lukasiewicz semantics, in: Proceedings of the 1st Workshop on Weighted Logics for Artificial Intelligence (ECAI'12), 2012, pp. 43–50.

[51] C. V. Damásio, L. M. Pereira, Antitonic logic programs, in: Proceedings of the 6th International Conference on Logic Programming and Non-monotonic Reasoning (LPNMR'01), Lecture Notes in Computer Science, 2001, pp. 379–392.

[52] M. Alviano, R. P. naloza, Fuzzy answer sets approximations, in: Proceedings of the 29th International Conference on Logic Programming (ICLP'13), 2013.

[53] B. Bouchon-Meunier, D. Dubois, L. Godo, H. Prade, Fuzzy sets and possibility theory in approximate and plausible reasoning, in: Fuzzy Sets in Approximate Reasoning and Information Systems, Vol. 5 of The Handbooks of Fuzzy Sets Series, Springer US, 1999, pp. 15–190.

[54] K. Bauters, S. Schockaert, M. De Cock, D. Vermeir, Towards possibilistic fuzzy answer set programming, in: Proceedings of the 13th International Workshop on Non-monotonic reasoning (NMR), 2010.