TLP **15** (1): 79–116, 2015. © Cambridge University Press 2014
doi:10.1017/S147106841300063X First published online 10 January 2014

Characterizing and extending answer set semantics using possibility theory

KIM BAUTERS
Department of Applied Mathematics, Computer Science and Statistics, Universiteit Gent
Krijgslaan 281 (WE02), 9000 Gent, Belgium
(*e-mail:* kim.bauters@gmail.com)

STEVEN SCHOCKAERT
School of Computer Science & Informatics, Cardiff University
5 The Parade, Cardiff CF24 3AA, UK
(*e-mail:* s.schockaert@cs.cardiff.ac.uk)

MARTINE DE COCK
Department of Applied Mathematics, Computer Science and Statistics, Universiteit Gent
Krijgslaan 281 (WE02), 9000 Gent, Belgium
(*e-mail:* martine.decock@ugent.be)

DIRK VERMEIR
Department of Computer Science, Vrije Universiteit Brussel,
Pleinlaan 2, 1050 Brussel, Belgium
(*e-mail:* dirk.vermeir@vub.ac.be)

submitted 20 March 2012; revised 19 August 2013; accepted 5 November 2013

Abstract

Answer Set Programming (ASP) is a popular framework for modelling combinatorial problems. However, ASP cannot be used easily for reasoning about uncertain information. Possibilistic ASP (PASP) is an extension of ASP that combines possibilistic logic and ASP. In PASP a weight is associated with each rule, whereas this weight is interpreted as the certainty with which the conclusion can be established when the body is known to hold. As such, it allows us to model and reason about uncertain information in an intuitive way. In this paper we present new semantics for PASP in which rules are interpreted as constraints on possibility distributions. Special models of these constraints are then identified as possibilistic answer sets. In addition, since ASP is a special case of PASP in which all the rules are entirely certain, we obtain a new characterization of ASP in terms of constraints on possibility distributions. This allows us to uncover a new form of disjunction, called weak disjunction, that has not been previously considered in the literature. In addition to introducing and motivating the semantics of weak disjunction, we also pinpoint its computational complexity. In particular, while the complexity of most reasoning tasks coincides with standard disjunctive ASP, we find that brave reasoning for programs with weak disjunctions is easier.

KEYWORDS: logic programming, answer set programming, possibility theory

1 Introduction

Answer set programming (ASP) is a form of logic programming with a fully declarative semantics, centred around the notion of a stable model. Syntactically, an ASP program is a set of rules of the form ($head \leftarrow body$) where *head* is true whenever *body* is true. Possibilistic ASP (PASP) extends upon ASP by associating a weight with every rule, which is interpreted as the necessity with which we can derive the head of the rule when the body is known to hold. Semantics for PASP have been introduced in Nicolas *et al.* (2006) for possibilistic normal programs and later extended to possibilistic disjunctive programs in Nieves *et al.* (2013). Under these semantics, a possibilistic rule with certainty λ allows us to derive *head* with certainty $\min(\lambda, N(body))$, where $N(body)$ denotes the necessity of the body, i.e. the certainty of *head* is restricted by the least certain piece of information in the derivation chain. Specifically, to deal with the PASP rules without negation-as-failure (NAF), the semantics from Nicolas *et al.* (2006) treat such rules as implications in possibilistic logic (Dubois *et al.* 1994). When faced with negation-as-failure, the semantics from Nicolas *et al.* (2006) rely on the reduct operation from classical ASP. Essentially, this means that the weights associated with the rules are initially ignored, the classical reduct is determined and the weights are then reassociated with the corresponding rules in the reduct. Given this particular treatment of negation-as-failure, the underlying intuition of '*not l*' is '*"l" cannot be derived with a strictly positive certainty*'. Indeed, as soon as '*l*' can be derived with a certainty $\lambda > 0$, '*l*' is treated as true when determining the reduct. However, this particular understanding of negation-as-failure is not always the most intuitive one.

Consider the following example. You want to go to the airport, but you notice that your passport will expire in less than three months. Some countries require that the passport is at least valid for an additional three months on the date of entry. As such, you have some certainty that your passport might be invalid (*invalid*). When you are not entirely certain that your passport is invalid, you should still go to the airport (*airport*) and check-in nonetheless. Indeed, since you are not absolutely certain that you will not be allowed to board, you might still get lucky. We have the following possibilistic program:

$$\begin{aligned} \mathbf{0.1:}\ & invalid \leftarrow \\ \mathbf{1:}\ & airport \leftarrow not\ invalid \end{aligned}$$

where 0.1 and 1 are the weights associated with the rules ($invalid \leftarrow$) and $airport \leftarrow invalid$, respectively. Clearly, what we would like to be able to conclude with a high certainty is that you need to go to the airport to check-in. However, as the semantics from Nicolas *et al.* (2006) adhere to a different intuition of negation-as-failure, the conclusion is that you need to go to the airport with a necessity of 0. Or, in other words, you should not go to the airport at all.

As a first contribution in this paper, we present new semantics for PASP by interpreting possibilistic rules as constraints on possibility distributions. These semantics do not correspond with the semantics from Nicolas *et al.* (2006) when

https://doi.org/10.1017/S147106841300063X Published online by Cambridge University Press

considering programs with negation-as-failure. Specifically, the semantics presented in this paper can be used in settings in which the possibilistic answer sets according to Nicolas *et al.* (2006) do not correspond with the intuitively acceptable results. For the example mentioned above, the conclusion under the new semantics is that you need to go to the airport with a necessity of 0.9.

In addition, the new semantics allow us to uncover a new characterization of ASP in terms of possibility theory. Over the years, many equivalent approaches have been proposed to define the notion of an answer set. One of the most popular characterizations is in terms of the Gelfond–Lifschitz reduct (Gelfond and Lifzchitz 1988) in which an answer set is guessed and verified to be stable. This characterization is used in the semantics for PASP as presented in Nicolas *et al.* (2006). Alternatively, the answer set semantics of normal programs can be defined in terms of autoepistemic logic (Marek and Truszczyński 1991), a well-known non-monotonic modal logic. An important advantage of the latter approach is that autoepistemic logic enjoys more syntactic freedom, which opens the door to more expressive forms of logic programming. However, as has been shown early on in Lifschitz and Schwarz (1993), the characterization in terms of autoepistemic logic does not allow us to treat classical negation or disjunctive rules in a natural way, which weakens its position as a candidate for generalizing ASP from normal programs to e.g. disjunctive programs. Equilibrium logic (Pearce 1997) offers yet another way for characterizing and extending ASP, but does not feature modalities which limits its potential for epistemic reasoning as it does not allow us to reason over the established knowledge of an agent. The new characterization of ASP, as presented in this paper, is a characterization in terms of necessary and contingent truths, where possibility theory is used to express our certainty in logical propositions. Such a characterization is unearthed by looking at ASP as a special case of PASP in which the rules are certain and no uncertainty is allowed in the answer sets. It highlights the intuition of ASP that the head of a rule is certain when the information encoded in its body is certain. Furthermore, this characterization stays close to the intuition of the Gelfond–Lifschitz reduct while sharing the explicit reference to modalities with autoepistemic logic.

As a second contribution, we show in this paper how this new characterization of ASP in terms of possibility theory can be used to uncover a new form of disjunction in both ASP and PASP. As indicated, we have that the new semantics offer us an explicit reference to modalities, i.e. operators with which we can qualify a statement. Epistemic logic is an example of a modal logic in which we use the modal operator K to reason about knowledge, where K is intuitively understood as '*we know that*'. A statement such as $a \vee b \vee c$ can then be treated in two distinct ways. On the one hand, we can interpret this statement as $Ka \vee Kb \vee Kc$, which makes it explicit that we know that one of the disjuncts is true. This treatment corresponds with the understanding of disjunction in disjunctive ASP and will be referred to as *strong disjunction*. Alternatively, we can interpret $a \vee b \vee c$ as $K(a \vee b \vee c)$, which only states that we know that the disjunction is true, i.e. we do not know which of the disjuncts is true. We will refer to this form of disjunction as *weak disjunction*. This is the new form of disjunction that we will discuss in this paper, as it allows us to reason

https://doi.org/10.1017/S147106841300063X Published online by Cambridge University Press

in settings where a choice cannot or should not be made. Still, such a framework allows for non-trivial forms of reasoning.

Consider the following example. A supervisory control and data acquisition (SCADA) system is used to monitor the brewing of beer in an industrialised setting. To control the fermentation, the system regularly verifies an air-lock for the presence of bubbles. An absence of bubbles may be due to a number of possible reasons. On the one hand, there may be a production problem such as a low yeast count or low temperature. Adding yeast when the temperature is low results in a beer with a strong yeast flavour, which should be avoided. Raising the temperature when there is too little yeast present will kill off the remaining yeast and will ruin the entire batch. On the other hand, there may be technical problems. There may be a malfunction in the SCADA system, which can be verified by running a diagnostic. The operator runs a diagnostic (*diagnostic*), which reports back that there is no malfunction ($\neg$*malfunction*). Or, alternatively, the air-lock may not be sealed correctly (*noseal*). The operator furthermore checks the temperature because he suspects that the temperature is the problem (*verifytemp*), but the defective temperature sensor returns no temperature when checked (*notemp*). These three technical problems require physical maintenance and the operator should send someone out to fix them. Technical problems do not affect the brewing. As such, the brewing process should not be interrupted for such problems as this will ruin the current batch. If there is a production problem, however, the brewing process needs to be interrupted as soon as possible (in addition, evidently, to interrupting the brewing process when the brewing is done). This prevents the current batch from being ruined due to over-brewing but also allows the interaction with the contents of the kettle. In particular, when the problem is diagnosed to be low yeast, the solution is to add a new batch of yeast and restart the process. Similarly, low temperature can be solved by raising the kettle temperature and restarting the fermentation process. Obviously, the goal is to avoid ruining the current batch. An employer radios in that the seal is okay. We have the following program:

$$
\begin{aligned}
\mathit{lowyeast} \vee \mathit{lowtemp} \vee \mathit{noseal} \vee \mathit{malfunction} &\leftarrow \mathit{not\ bubbles} \\
\mathit{diagnostic} &\leftarrow \\
\neg\mathit{malfunction} &\leftarrow \mathit{diagnostic} \\
\mathit{verifytemp} &\leftarrow \\
\mathit{notemp} &\leftarrow \mathit{verifytemp} \\
\mathit{maintenance} &\leftarrow \mathit{noseal} \vee \mathit{malfunction} \vee \mathit{notemp} \\
\mathit{brew} &\leftarrow \mathit{not}\ (\mathit{lowyeast} \vee \mathit{lowtemp} \vee \mathit{done}) \\
\mathit{addyeast} &\leftarrow \mathit{lowyeast} \\
\mathit{raisetemp} &\leftarrow \mathit{lowtemp} \\
\mathit{ruin} &\leftarrow \mathit{raisetemp}, \mathit{not\ lowtemp} \\
\mathit{ruin} &\leftarrow \mathit{addyeast}, \mathit{not\ lowyeast} \\
\mathit{ruin} &\leftarrow \mathit{not\ brew}, \mathit{not}\ (\mathit{lowtemp} \vee \mathit{lowyeast})
\end{aligned}
$$

https://doi.org/10.1017/S147106841300063X Published online by Cambridge University Press

$$\leftarrow ruin$$
$$\neg noseal \leftarrow$$

The above program does not use the standard ASP syntax since we allow for disjunction in the body. Furthermore, the disjunction used in the head and the body is weak disjunction. The only information that we can therefore deduce from e.g. the first rule is ($lowyeast \vee lowtemp \vee noseal \vee malfunction$). At first, this new form of disjunction may indeed appear weaker that strong disjunction since it does not induce a choice. Still, even without inducing a choice, conclusions obtained from other rules may allow us to refine our knowledge. In particular, note that from $lowyeast \vee lowtemp \vee noseal \vee malfunction$ together with $\neg malfunction$ and $\neg noseal$ we can entail $lowyeast \vee lowtemp$. Similarly, conclusions can also have prerequisites that are disjunctions. For example, we can no longer deduce *brew* since $lowyeast \vee lowtemp$ entails $lowyeast \vee lowtemp \vee done$. From $maintenance \leftarrow noseal \vee malfunction \vee notemp$ and *notemp* we can deduce that we should call maintenance. However, we do not yet have enough information to diagnose whether yeast should be added or the temperature should be raised. The unique answer set of this program, according to the semantics of weak disjunction which we present in Section 4, is given by

$$\{lowyeast \vee lowtemp, maintenance,$$
$$diagnostic, \neg malfunction, verifytemp, notemp, \neg noseal\}$$

The expressiveness of a weak disjunction becomes clear when we study its complexity. In particular, we show that while most complexity results coincide with the strong disjunctive semantics, the complexity of brave reasoning (deciding whether a literal 'l' is entailed by a consistent answer set of program P) in absence of negation-as-failure is lower for weak disjunction. Still, the expressiveness is higher than for normal programs. The complexity results are summarized in Table 1 in Section 5.

The remainder of this paper is organized as follows. In Section 2 we provide the reader with some important notions from ASP and possibilistic logic. In Section 3 we introduce new semantics for PASP which can furthermore be used to characterize normal ASP programs using possibility theory. In Section 4 we characterize disjunctive ASP in terms of constraints on possibility distributions, and we discuss the complexity results of the new semantics for PASP in detail in Section 5. Related work is discussed in Section 6, and we formulate our conclusions in Section 7.

This paper aggregates and extends parts of our work from Bauters *et al.* (2011) and substantially extends a previous conference paper (Bauters *et al.* 2010) which did not consider classical negation nor computational complexity. In addition, rather than limiting ourselves to atoms in this paper, we extend our work to cover the case of literals which offer interesting and unexpected results in the face of weak disjunction. Complexity results are added for all reasoning tasks and full proofs are provided in the appendix.

https://doi.org/10.1017/S147106841300063X Published online by Cambridge University Press

2 Background

We start by reviewing the definitions from both ASP and possibilistic logic that will be used in the remainder of the paper. We then review the semantics of PASP from Nicolas *et al.* (2006), a framework that combines possibilistic logic and ASP. Finally, we recall some notions from the complexity theory.

2.1 Answer Set Programming

To define ASP programs, we start from a finite set of atoms $\mathscr{A}$. A *literal* is defined as an atom a or its classical negation $\neg a$. For L a set of literals, 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 L is *consistent* if $L \cap \neg L = \emptyset$. We write the set of all literals as $\mathscr{L} = (\mathscr{A} \cup \neg\mathscr{A})$. A *naf-literal* is either a literal 'l' or a literal 'l' preceded by *not*, which we call the *negation-as-failure operator*. Intuitively, '*not* l' is true when we cannot prove 'l'. An expression of the form

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

with l_i a literal for every $0 \leqslant i \leqslant n$ is called a *disjunctive rule*. We call $l_0; ...; l_k$ the *head* of the rule (interpreted as a disjunction) and $l_{k+1}, ..., l_m, not\ l_{m+1}, ..., not\ l_n$ the *body* of the rule (interpreted as a conjunction). For a rule r we use $head(r)$ and $body(r)$ to denote the set of literals in the head, resp. the body. 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 $n = m$, we say that it 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* and is used as a shorthand for $(l_0; ...; l_k \leftarrow \top)$ with $\top$ a special language construct that denotes tautology. 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 the rule of the form $(\bot \leftarrow l_{k+1}, ..., l_m, not\ l_{m+1}, ..., not\ l_n)$ with $\bot$ a special language construct that denotes 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 with no negation-as-failure. A *definite rule* is a simple rule with no classical negation, i.e. in which all literals are atoms. A *normal* (resp. *simple*, *definite*) *program* P is a set of normal (resp. simple, definite) rules.

The *Herbrand base* $\mathscr{B}_P$ of a disjunctive program P is the set of atoms appearing in P. We define the set of literals that are relevant for a disjunctive program P as $Lit_P = (\mathscr{B}_P \cup \neg\mathscr{B}_P)$. An *interpretation* I of a disjunctive program P is any set of literals $I \subseteq Lit_P$. A *consistent interpretation* I is an interpretation I that does not contain both a and $\neg a$ for some $a \in I$.

A *consistent interpretation* I is said to be a *model* of a positive disjunctive rule r if $head(r) \cap I \neq \emptyset$ or $body(r) \nsubseteq I$, i.e. the body is false or the head is true. In particular, a consistent interpretation I is a *model* of a constraint rule r if $body(r) \nsubseteq I$. If for an interpretation I and a constraint rule r we have that $body(r) \subseteq I$, then we say that the interpretation I *violates* the constraint rule r. Note that for a fact rule we require

https://doi.org/10.1017/S147106841300063X Published online by Cambridge University Press

that $head(r) \cap I \neq \emptyset$, i.e. at least one of the literals in the head must be true. Indeed, otherwise I would not be a model of r. An interpretation I of a positive disjunctive program P is a model of P either if I is consistent and for every rule $r \in P$ we have that I is a model of r, or if $I = Lit_P$. It follows from this definition that Lit_P is always a model of P, and that all other models of P (if any) are consistent interpretations, which we will further on also refer to as *consistent models*. We say that I is an *answer set* of the positive disjunctive program P when I is a minimal model of P w.r.t. set inclusion.

The semantics of an ASP program with negation-as-failure is based on the idea of a stable model (Gelfond and Lifzchitz 1988). The reduct P^I of a disjunctive program P w.r.t. the interpretation I is defined as:

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

An interpretation 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. An 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 interpretations, we say that P is a *consistent program*. When P has the answer set Lit_P, then this is the unique (Baral 2003) inconsistent answer set and we say that P is an *inconsistent program*.

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 without constraint rules and w.r.t. an interpretation I as:

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

We use $P^\star$ 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 interpretation $P^\star$ is consistent, $P^\star$ is the (unique and consistent) answer set of the simple program P without constraint rules. When we allow constraint rules, an interpretation is a (consistent) answer set of $P = P' \cup C$ iff I is a (consistent) answer set of P and I is a 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).

2.2 *Possibilistic logic*

An interpretation in possibilistic logic corresponds with the notion of an interpretation in propositional logic. We represent such an interpretation as a set of atoms ω, where $\omega \models a$ if $a \in \omega$ and $\omega \models \neg a$ otherwise with $\models$ the satisfaction relation from classical logic. The set of all interpretations is defined as $\Omega = 2^{\mathscr{A}}$, with $\mathscr{A}$ a finite set of atoms. At the semantic level, possibilistic logic (Dubois *et al.* 1994) is defined in terms of a *possibility distribution* π on the universe of

https://doi.org/10.1017/S147106841300063X Published online by Cambridge University Press

interpretations. A possibility distribution, which is an $\Omega \rightarrow [0,1]$ mapping, 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. We say that a possibility distribution π is *vacuous* when $\forall \omega \in \Omega \cdot \pi(\omega) = 0$. 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 $\pi_1 \geqslant \pi_2$ when $\forall \omega \in \Omega \cdot \pi_1(\omega) \geqslant \pi_2(\omega)$ and we write $\pi_1 > \pi_2$ when $\pi_1 \geqslant \pi_2$ and $\pi_1 \neq \pi_2$.

A possibility distribution π induces two uncertainty measures that allow us to rank propositions. The *possibility measure* Π is defined by Dubois *et al.* (1994):

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

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 (Dubois *et al.* 1994). Note that we always have $N(\top) = 1$ for any possibility distribution, while $\Pi(\top) = 1$ (and, related, $N(\bot) = 0$) only holds when the possibility distribution is normalized (i.e. only normalized possibility distributions can express consistent beliefs) (Dubois *et al.* 1994). 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 the min-decomposability property 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 $N(p \vee q) \geqslant \max(N(p), N(q))$ holds. As possibility measures are the dual measures of necessity measures, they have the property of max-decomposability w.r.t. disjunction, whereas for the conjunction only the inequality $\Pi(p \wedge q) \leqslant \min(\Pi(p), \Pi(q))$ 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 $N(p) \geqslant c$, 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 *least specific possibility distribution*, which is the possibility distribution that makes minimal commitments, i.e. the greatest possibility distribution w.r.t. the ordering $>$ defined above. Such a least specific possibility distribution always exists and is unique (Dubois *et al.* 1994).

In Section 4 we will also consider constraints that deviate from the form of constraints we just discussed. As a result, there can be multiple minimally specific

possibility distributions rather than a unique least specific possibility distribution. To increase the uniformity throughout the paper we immediately start using the concept of a *minimally specific possibility distribution*, which is a maximal possibility distribution w.r.t. the ordering $>$, even though the distinction between the least specific possibility distribution and minimally specific possibility distributions only becomes relevant once we discuss the characterization of disjunctive programs.

2.3 Possibilistic Answer Set Programming

Possibilistic ASP (Nicolas *et al.* 2006) combines ASP and possibility theory by associating a weight with each rule, where the weight denotes 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.

Syntactically, a possibilistic disjunctive (resp. normal, simple, definite) program is a set of pairs $p = (r, \lambda)$ with r a disjunctive (resp. normal, simple, definite) rule and $\lambda \in]0, 1]$ a certainty associated with r. Possibilistic rules with $\lambda = 0$ are generally omitted as only trivial information can be derived from them. We will also write a possibilistic rule $p = (r, \lambda)$ 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:

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

For a possibilistic rule $p = (r, \lambda)$ 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 $cert(P) = \{\lambda \mid p = (r, \lambda) \in P\}$. We will also use the extended set of weights $cert^+(P)$, defined as $cert^+(P) = \{\lambda \mid \lambda \in cert(P)\} \cup \{1 - \lambda \mid \lambda \in cert(P)\} \cup \{0, \frac{1}{2}, 1\}$.

Semantically, PASP is based on a generalization of the concept of an interpretation. In classical ASP, an 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 $V(l) = \lambda$ is that the literal 'l' is true with certainty 'λ', which we will also write in set notation as $l^\lambda \in V$. As such, a valuation corresponds with the set of constraints $\{N(l) \geqslant \lambda \mid l^\lambda \in V\}$. Note that, like interpretations in ASP, these valuations are of an epistemic nature, i.e. they reflect what we know about the truth of atoms. For notational convenience, we often also use the set notation $V = \{l^\lambda, ...\}$. In accordance with this set notation, we write $V = \emptyset$ to denote the valuation in which each literal is mapped to 0. For $\lambda \in [0, 1]$, a certainty, and V, a valuation, we use V^λ to denote the classical projection $\{l \mid l \in Lit_P, V(l) \geqslant \lambda\}$. We also use $V^{\underline{\lambda}} = \{l \mid l \in Lit_P, V(l) > \lambda\}$, i.e. those literals that can be derived to be true with certainty strictly greater than 'λ'. 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 $N_V(l) = V(l)$.

https://doi.org/10.1017/S147106841300063X Published online by Cambridge University Press

We now present a straightforward extension of the semantics for PASP introduced in Nicolas *et al.* (2006). Let the λ-cut P_λ of a possibilistic program P, with $\lambda \in [0,1]$, be defined as:

$$P_\lambda = \{r \mid (r, \lambda') \in P \text{ and } \lambda' \geqslant \lambda\},$$

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

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

$$T_P(V)(l_0) = \max\left\{\lambda \in [0,1] \mid V^\lambda \models l_1, ..., l_m \text{ and } (l_0 \leftarrow l_1, ..., l_m) \in P_\lambda\right\}.$$

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 Nicolas *et al.* (2006), although we formulate it somewhat differently. Also, the work from Nicolas *et al.* (2006) only considered definite programs, even though adding classical negation does not impose any problems.

As before, we use $P^\star$ 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. Valuation V is said to be the answer set of a possibilistic simple program if $V = P^\star$ 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 (Nicolas *et al.* 2006):

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

A consistent valuation V is said to be a possibilistic answer set of the possibilistic normal program P iff $\left(P^{(V^0)}\right)^\star = V$, i.e. if V is the answer set of the reduct $P^{(V^0)}$.

Example 1
Consider the possibilistic normal program P from the Introduction:

$$\textbf{0.1: } invalid \leftarrow$$
$$\textbf{1: } airport \leftarrow not\ invalid$$

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

$$\textbf{0.1: } invalid \leftarrow$$

from which it trivially follows that $(P^{\{invalid\}})^\star = \{invalid^{0.1}\}$. The conclusion is thus that we do not need to go to the airport, which differs from our intuition of the problem. We will revisit this example in Example 4 in Section 3.2.

The semantics we presented allow for classical negation, even though this was not considered in Nicolas *et al.* (2006). However, adding classical negation does not

https://doi.org/10.1017/S147106841300063X Published online by Cambridge University Press

impose any problems and could, as an alternative, easily be simulated in ASP (Baral 2003).

2.4 Complexity theory

Finally, we recall some notions from the complexity theory. The complexity classes Σ_2^{P} and Π_2^{P} are defined as follows (Papadimitriou 1994):

$$\Sigma_0^{\mathsf{P}} = \Pi_0^{\mathsf{P}} = \mathsf{P}$$
$$\Sigma_1^{\mathsf{P}} = \mathsf{NP} \qquad \Sigma_2^{\mathsf{P}} = \mathsf{NP}^{\mathsf{NP}}$$
$$\Pi_1^{\mathsf{P}} = \mathsf{coNP} \qquad \Pi_2^{\mathsf{P}} = \mathsf{co}\Sigma_2^{\mathsf{P}}$$

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. We also consider the complexity class BH_2 (Cai *et al.* 1988), which is the class of all languages L such that $L = L_1 \cap L_2$, where L_1 is in NP and L_2 is in coNP. For a general complexity class C, a problem is C-hard if any problem in C can be polynomially reduced to this problem. 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 (QBF) $\phi = \exists X_1 \forall X_2 \cdot p(X_1, X_2)$ with $p(X_1, X_2)$ in disjunctive normal form (DNF) is the canonical Σ_2^P-complete problem. The decision problems we consider in this paper are brave reasoning (deciding whether a literal '*l*' (clause '*e*') is entailed by a consistent answer set of program P), cautious reasoning (deciding whether a literal '*l*' (clause '*e*') is entailed by every consistent answer set of a program P) and answer set existence (deciding whether a program P has a consistent answer set). Brave reasoning as well as answer set existence for simple, normal and disjunctive programs is P-complete, NP-complete and Σ_2^{P}-complete, respectively (Baral 2003). Cautious reasoning for simple, normal and disjunctive programs is P-complete, coNP-complete and Π_2^{P}-complete (Baral 2003).

3 Characterizing (P)ASP

ASP lends itself well to being characterized in terms of modalities. For instance, ASP can be characterized in autoepistemic logic by interpreting '*not a*' as the epistemic formula $\neg \mathsf{L}a$ ('a is not believed') (Gelfond 1987). In this paper, as an alternative, we show how ASP can be characterized within possibility theory. To arrive at this characterization, we first note that ASP is essentially a special case of PASP in which every rule is certain. As such, we will show how PASP can be characterized within possibility theory. This characterization does not coincide with the semantics proposed in Nicolas *et al.* (2006) for PASP, as the semantics from Nicolas *et al.* (2006) rely on the classical Gelfond–Lifschitz reduct. Rather, the semantics that we propose for PASP adhere to a different intuition of negation-as-failure. A characterization of ASP is then obtained from these new semantics by considering the special case in which all rules are entirely certain.

https://doi.org/10.1017/S147106841300063X Published online by Cambridge University Press

This characterization of ASP, while still in terms of modalities, stays close in spirit to the Gelfond–Lifschitz reduct. In contrast to the characterization in terms of autoepistemic logic, it does not require a special translation of literals to deal with classical negation and disjunction. The core idea of our characterization is to encode the meaning of each rule as a constraint on possibility distributions. Particular minimally specific possibility distributions that satisfy all the constraints imposed by the rules of a program will then correspond with the answer sets of that program.

In this section, we first limit our scope to possibilistic simple programs (Section 3.1). Afterwards we will broaden the scope and also consider possibilistic normal programs (Section 3.2). The most general case in which we also consider possibilistic disjunctive programs will be discussed in Section 4.

3.1 Characterizing possibilistic simple programs

When considering a fact, i.e. a rule of the form $r = (l_0 \leftarrow \top)$, we know by definition that this rule encodes that the literal in the head is necessarily true, i.e. $N(l_0) = 1$. If we attach a weight to a fact, then this expresses the knowledge that we are not entirely certain of the conclusion in the head, i.e. for a possibilistic rule $p = (r, \lambda)$ we have that $N(l_0) \geqslant N(\top)$. Note that the constraint uses $\geqslant$, as there may be other rules in the program that allow us to deduce l_0 with a greater certainty.

In a similar fashion we can characterize a rule of the form $(l_0 \leftarrow l_1, ..., l_m)$ as the constraint $N(l_0) \geqslant N(l_1 \wedge ... \wedge l_m)$ which is equivalent to the constraint $N(l_0) \geqslant \min(N(l_1), ..., N(l_m))$ due to the min-decomposability property of the necessity measure. Indeed, the intuition of such a rule is that the head is only necessarily true when every part of the body is true. When associating a weight with a rule, we obtain the constraint $N(l_0) \geqslant \min(N(l_1), ..., N(l_m), \lambda)$ for a possibilistic rule $p = (r, \lambda)$ with $r = (l_0 \leftarrow l_1, ..., l_m)$. Similarly, to characterize a constraint rule, i.e. a rule of the form $r = (\bot \leftarrow l_1, ..., l_m)$, we use the constraint $N(\bot) \geqslant \min(N(l_1), ..., N(l_m))$, or, in the possibilistic case with $p = (r, \lambda)$, the constraint $N(\bot) \geqslant \min(N(l_1), ..., N(l_m), \lambda)$.

Definitiona 2

Let P be a possibilistic simple program and $\pi : \Omega \rightarrow [0, 1]$ a possibility distribution. For every $p \in P$, the constraint $\gamma(p)$ imposed by $p = (r, \lambda)$ with $\lambda \in]0, 1]$, $r = (l_0 \leftarrow l_1, ..., l_m)$ and $m \geqslant 0$ is given by

$$N(l_0) \geqslant \min(N(l_1), ..., N(l_m), \lambda). \tag{1}$$

$C_P = \{\gamma(p) \mid p \in P\}$ is the set of constraints imposed by program P. If π satisfies the constraints in C_P, π is said to be a possibilistic model of C_P, written $\pi \models C_P$. A possibilistic model of C_P will also be called a possibilistic model of P. We write S_P for the set of all minimally specific possibilistic models of P.

Definitiona 3

Let P be a possibilistic simple program. Let π be a minimally specific model of P, i.e. $\pi \in S_P$. Then $V = \left\{l^{N(l)} \mid l \in Lit_P\right\}$ is called a *possibilistic answer set* of P.

https://doi.org/10.1017/S147106841300063X Published online by Cambridge University Press

Example 2
Consider the possibilistic simple program P with the rules:

$$\mathbf{0.8}: a \leftarrow \qquad \mathbf{0.6}: \neg b \leftarrow a$$
$$\mathbf{0.7}: c \leftarrow a, \neg b \qquad \mathbf{0.9}: d \leftarrow d.$$

The set C_P consists of the constraints:

$$N(a) \geqslant 0.8 \qquad N(\neg b) \geqslant \min(N(a), 0.6)$$
$$N(c) \geqslant \min(N(a), N(\neg b), 0.7) \qquad N(d) \geqslant \min(N(d), 0.9).$$

It is easy to see that the last constraint is trivial and can be omitted and that the other constraints can be simplified to $\Pi(\neg a) \leqslant 0.2$, $\Pi(b) \leqslant 0.4$ and $\Pi(\neg c) \leqslant 0.4$. The least specific possibility distribution that satisfies these constraints is given by

$$\begin{array}{rrrr}
\pi(\{a,b,c,d\}) = 0.4 & \pi(\{a,c,d\}) = 1 & \pi(\{b,c,d\}) = 0.2 & \pi(\{c,d\}) = 0.2 \\
\pi(\{a,b,c\}) = 0.4 & \pi(\{a,c\}) = 1 & \pi(\{b,c\}) = 0.2 & \pi(\{c\}) = 0.2 \\
\pi(\{a,b,d\}) = 0.4 & \pi(\{a,d\}) = 0.4 & \pi(\{b,d\}) = 0.2 & \pi(\{d\}) = 0.2 \\
\pi(\{a,b\}) = 0.4 & \pi(\{a\}) = 0.4 & \pi(\{b\}) = 0.2 & \pi(\{\}) = 0.2.
\end{array}$$

By definition, since the possibility distribution satisfies the given constraints, π is a possibilistic model. Furthermore, it is easy to see that π is the unique minimally specific possibilistic model (due to least specificity). We can verify that $N(\neg a) = N(b) = N(\neg c) = N(\neg d) = 0$ since we have that $\pi(\{a,c,d\}) = 1$ and $N(d) = 0$ since $\pi(\{a,c\}) = 1$. Furthermore, it is easy to verify that $N(a) = 0.8$, $N(\neg b) = 0.6$ and $N(c) = 0.6$. Hence, we find that $V = \{a^{0.8}, \neg b^{0.6}, c^{0.6}\}$ is a possibilistic answer set of P.

In particular, when we consider all the rules to be entirely certain, i.e. $\lambda = 1$, the results are compatible with the semantics of classical ASP.

Example 3
Consider the program $P = \{(b \leftarrow a), (\neg a \leftarrow)\}$. The set of constraints C_P is given by $N(b) \geqslant N(a)$ and $N(\neg a) \geqslant N(\top)$. The first constraint can be rewritten as $1 - \Pi(\neg b) \geqslant 1 - \Pi(\neg a)$, i.e. as $\Pi(\neg a) \geqslant \Pi(\neg b)$. The last constraint can be rewritten as $1 - \Pi(a) \geqslant 1$, i.e. as $\Pi(a) = \max\{\pi(\omega) \mid \omega \models a\} = 0$. Given these two constraints, we find that S_P contains exactly one element, which is defined by

$$\pi(\{a,b\}) = 0 \qquad \pi(\{a\}) = 0$$
$$\pi(\{b\}) = 1 \qquad \pi(\{\}) = 1.$$

Note how the first constraint turned out to be of no relevance for this particular example. Indeed, due to the principle of minimal specificity and since there is nothing that prevents $\Pi(\neg a) = 1$, we find that $N(a) = 1 - \Pi(\neg a) = 0$. Therefore, the first constraint simplifies to $N(b) \geqslant 0$. Once more, due to the principle of minimal specificity we thus find that $N(b) = 0$ as there is no information that prevents $\Pi(\neg b) = 1$. To find out whether a, b, $\neg a$ and $\neg b$ are necessarily true w.r.t. the least specific possibility distribution $\pi \in S_P$ arising from the program, we verify whether $N(a) = 1$, $N(b) = 1$, $N(\neg a) = 1$ and $N(\neg b) = 1$, respectively, with N the necessity

https://doi.org/10.1017/S147106841300063X Published online by Cambridge University Press

measure induced by the unique least specific possibility distribution $\pi \in S_P$. As desired, we find that $N(\neg a) = 1 - \Pi(a) = 1$ whereas $N(a) = N(b) = N(\neg b) = 0$. The unique possibilistic answer set is therefore $\{\neg a^1\}$. As we will see, it then follows from Proposition 1 that the unique classical answer set of P is $\{\neg a\}$.

In Propositions 1 and 2 below, we prove that this is indeed a correct characterization of simple programs. First, we present a technical lemma.

Lemma 1
Let L be a set of literals, $M \subseteq L$ a consistent set of literals and let the possibility distribution π be defined as $\pi(\omega) = 1$ if $\omega \models M$ and $\pi(\omega) = 0$ otherwise. Then $M = \{l \mid N(l) = 1, l \in L\}$.

The proof is given in the online appendix of the paper (pp. 1–2).

Proposition 1
Let P be a simple program. If $\pi \in S_P$ then either the unique consistent answer set of P is given by $M = \{l \mid N(l) = 1, l \in Lit_P\}$, or π is the vacuous distribution, in which case P does not have any consistent answer sets.

The proof is given in the online appendix of the paper (pp. 2–4) available at http://dx.doi.org/10.1017/S147106841300063X.

Proposition 2
Let P be a simple program. If M is an answer set of P, then the possibility distribution π defined by $\pi(\omega) = 1$ iff $\omega \models M$ and $\pi(\omega) = 0$ otherwise belongs to S_P.

The proof is given in the online appendix of the paper (p. 4).

3.2 Characterizing possibilistic normal programs

To deal with negation-as-failure, we rely on a reduct-style approach in which a valuation is guessed and it is verified whether this guess is indeed stable. The approach taken in Gelfond and Lifzchitz (1988) to deal with negation-as-failure is to guess an interpretation and verify whether this guess is stable. We propose to treat a rule of the form $r = (l_0 \leftarrow l_1, ..., l_m, not\ l_{m+1}, ..., not\ l_n)$ as the constraint

$$N(l_0) \geqslant \min(N(l_1), ..., N(l_m), 1 - V(l_{m+1}), ..., 1 - V(l_n)),$$

where V is the guess for the valuation, and where we assume $min(\{\}) = 1$. Or, when we consider a possibilistic rule $p = (r, \lambda)$, we treat it as the constraint

$$N(l_0) \geqslant \min(N(l_1), ..., N(l_m), 1 - V(l_{m+1}), ..., 1 - V(l_n), \lambda).$$

We like to make it clear to the reader that the characterization of normal programs in terms of constraints on possibility distributions in its basic form is little more than a reformulation of the Gelfond–Lifschitz approach. The key difference is that this characterization can be used to guess the certainty with which we can derive particular literals from the available rules, rather than guessing what may or may not be derived from it. Nevertheless, this difference plays a crucial role when dealing with uncertain rules. In particular, this characterization of PASP does not coincide with the semantics of Nicolas *et al.* (2006) and adheres to a different intuition for negation-as-failure.

Definitiona 4
Let P be a possibilistic normal program and let V be a valuation. For every $p \in P$, the constraint $\gamma_V(p)$ induced by $p = (r, \lambda)$ with $\lambda \in]0,1]$, $r = (l_0 \leftarrow l_1, ..., l_m, not\ l_{m+1}, ..., not\ l_n)$ and V is given by

$$N(l_0) \geqslant \min(N(l_1), ..., N(l_m), 1 - V(l_{m+1}), ..., 1 - V(l_n), \lambda). \quad (2)$$

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

Definitiona 5
Let P be a possibilistic normal 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 4
Consider the possibilistic normal program P from Example 1. The constraints C_P induced by P are:

$$N(invalid) \geqslant 0.1$$
$$N(airport) \geqslant \min(1 - V(invalid), 1)$$

From the first constraint it readily follows that we need to choose $V(invalid) = 0.1$ to comply with the principle of minimal specificity. The other constraint can then readily be simplified to:

$$N(airport) \geqslant 0.9$$

Hence, it follows that $V = \{invalid^{0.1}, airport^{0.9}\}$ is the unique possibilistic answer set of P.

It is easy to see that the proposed semantics remain closer to the intuition of the possibilistic normal program discussed in the Introduction. Indeed, we conclude with a high certainty that we need to go to the airport.

Still it is interesting to further investigate the particular relationship between the semantics for PASP as proposed in Nicolas *et al.* (2006) and the semantics presented in this section. Let the possibilistic rule r be of the form:

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

When we determine the reduct w.r.t. valuation V of the possibilistic program containing r, then the certainty of the rule in the reduct that corresponds with r can be verified to be:

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

with F_N being a fuzzy negator, i.e. where F_N is a decreasing function with $F_N(0) = 1$ and $F_N(1) = 0$. In particular, for the semantics of Nicolas *et al.* (2006) we have that F_N is the Gödel negator F_G, defined as $F_G(0) = 1$ and $F_G(c) = 0$ with $0 < c \leqslant 1$.

https://doi.org/10.1017/S147106841300063X Published online by Cambridge University Press

In the semantics for PASP presented in this section, F_N is the Łukasiewicz negator $F_{\text{Ł}}(c) = 1 - c$ with $0 \leqslant c \leqslant 1$. Thus, for a rule such as:

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

and valuation $V = \{a^{0.2}\}$ we obtain under the approach from Nicolas *et al.* (2006) the reduct (**0:** $b \leftarrow$), whereas under our approach we obtain the constraint $N(b) \geqslant \min(0.9, 1 - 0.2)$, which can be encoded by the rule (**0.8:** $b \leftarrow$). Essentially, the difference between both semantics can thus be reduced to a difference in the choice of negator. However, even though the semantics share similarities, there is a notable difference in the underlying intuition of both approaches. Specifically, in the semantics presented in this paper, we have that '*not l*' is understood as '*the degree to which "¬l" is possible*', or, equivalently, '*the degree to which it is not the case that we can derive "l" with certainty*'. This contrasts with the intuition of '*not l*' in Nicolas *et al.* (2006) as a Boolean condition and understood as '*we cannot derive "l" with a strictly positive certainty*'.

Interestingly, we find that the complexity of the main reasoning tasks for possibilistic normal programs remains at the same level of the polynomial hierarchy as the corresponding normal ASP programs.

While we will see in Section 5 that the complexity of possibilistic normal programs remains unchanged compared with classical normal programs, it is important to note that under the semantics proposed in this section there is no longer a 1-on-1 mapping between the classical answer sets of a normal program and the possibilistic answer sets. Indeed, if we consider a possibilistic normal program constructed from a classical normal program where we attach certainty $\lambda = 1$ to each rule, then we can sometimes obtain additional intermediary answer sets. Consider the next example.

Example 5

Consider the normal program with the single rule $a \leftarrow not\ a$. This program has no classical answer sets. Now consider the possibilistic normal program P with the rule

$$\textbf{1:}\ a \leftarrow not\ a.$$

The set of constraints $C_{(P,V)}$ is given by

$$N(a) \geqslant \min(1 - V(a), 1).$$

This constraint can be rewritten as

$$\begin{aligned} & N(a) \geqslant \min(1 - V(a), 1) \\ & \equiv N(a) \geqslant 1 - V(a) \\ & \equiv 1 - \Pi(\neg a) \geqslant 1 - V(a) \\ & \equiv \Pi(\neg a) \leqslant V(a). \end{aligned}$$

We thus find that the set $S_{(P,V)}$ is a singleton with $\pi \in S_{(P,V)}$ defined by $\pi(\{a\}) = 1$ and $\pi(\{\}) = V(a)$. We can now establish for which choices of $V(a)$ it holds that

https://doi.org/10.1017/S147106841300063X Published online by Cambridge University Press

$V(a) = N(a)$:

$$\begin{aligned} V(a) &= N(a) \\ \Pi(\neg a) &= 1 - \Pi(\neg a) \\ 2 \cdot \Pi(\neg a) &= 1 \end{aligned}$$

and thus, since $\Pi(\neg a) \leqslant V(a)$, we have $\pi(\{\}) = 0.5$. The unique possibilistic answer set of P is therefore $\{a^{0.5}\}$. In the same way, one may verify that the program

$$\textbf{1:}\ a \leftarrow not\ b \qquad\qquad \textbf{1:}\ b \leftarrow not\ a$$

has an infinite number of possibilistic answer sets, i.e. $\{a^c, b^{1-c}\}$ for every $c \in [0,1]$. For practical purposes, however, this behaviour has a limited impact as we only need to consider a finite number of certainty levels to perform brave/cautious reasoning. Indeed, we only need to consider the certainties used in the program, their complement to account for negation-as-failure and $\frac{1}{2}$ to account for the intermediary value as in Example 5. Thus, for the main reasoning tasks it suffices to limit our attention to the certainties from the set $cert^+(P)$.

We now show that when we consider rules with an absolute certainty, i.e. classical normal programs, we obtain a correct characterization of classical ASP, provided that we restrict ourselves to absolutely certain conclusions, i.e. valuations V for which it holds that $\forall l \cdot V(l) \in \{0,1\}$.

Example 6
Consider the program P with the rules

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

The set of constraints $C_{(P,V)}$ is then given by

$$N(a) \geqslant 1 \qquad\qquad N(b) \geqslant N(b) \qquad\qquad N(c) \geqslant \min\left(N(a), 1 - V(b)\right).$$

We can rewrite the first constraint as $1 - \Pi(\neg a) \geqslant 1$ and thus $\Pi(\neg a) = 0$. The second constraint is trivially satisfied and, since it does not entail any new information, can be dropped. The last constraint can be rewritten as $\Pi(\neg c) \leqslant 1 - \min(1 - \Pi(\neg a), 1 - V(b))$, which imposes an upper bound on the value that $\Pi(\neg c)$ can assume. Since we already know that $\Pi(\neg a) = 0$, we can further simplify this inequality to $\Pi(\neg c) \leqslant 1 - \min(1 - 0, 1 - V(b)) = 1 - (1 - V(b)) = V(b)$. In conclusion, the program imposes the constraints

$$\Pi(\neg a) = 0 \qquad\qquad \Pi(\neg c) \leqslant V(b).$$

The set $S_{(P,V)}$ then contains exactly one element, which is defined by

$$\begin{aligned} \pi(\{a,b,c\}) &= 1 & \pi(\{b,c\}) &= 0 \\ \pi(\{a,b\}) &= V(b) & \pi(\{b\}) &= 0 \\ \pi(\{a,c\}) &= 1 & \pi(\{c\}) &= 0 \\ \pi(\{a\}) &= V(b) & \pi(\{\}) &= 0. \end{aligned}$$

https://doi.org/10.1017/S147106841300063X Published online by Cambridge University Press

Note that this possibility distribution is independent of the choice for $V(a)$ and $V(c)$ since there are no occurrences of '*not a*' and '*not c*' in P. It remains then to determine for which choices of $V(b)$ it holds that $V(b) = N(b)$, i.e. for which the guess $V(b)$ is stable. We have:

$$V(b) = N(b) = 1 - \Pi(\neg b) = 1 - \max\{\pi(\omega) \mid \omega \models \neg b\} = 0$$

and thus we find that $\pi(\{a, b\}) = \pi(\{a\}) = 0$. We have $N(a) = 1 - \Pi(\neg a) = 1$, $N(c) = 1 - \Pi(\neg c) = 1$ and $N(b) = 1 - \Pi(\neg b) = 0$. As we will see in the next propositions, the unique answer set of P is therefore $\{a, c\}$.

Proposition 3
Let P be a normal program and V be a valuation. Let $\pi \in S_{(P,V)}$ such that

$$\forall l \in Lit_P \cdot V(l) = N(l) \text{ ; and} \tag{3}$$

$$\forall l \in Lit_P \cdot N(l) \in \{0, 1\} \tag{4}$$

then $M = \{l \mid N(l) = 1, l \in Lit_P\}$ is an answer set of the normal program P.

Proof
This proposition is a special case of Proposition 5 presented below. □

Note that the requirement stated in (4) cannot be omitted. Let us consider Example 5, in which we considered the normal program $P = \{a \leftarrow not\ a\}$. This normal program P has no classical answer sets. The constraint that corresponds with the rule $(a \leftarrow not\ a)$ is $N(a) \geqslant 1 - V(a)$. For a choice of $V = \{a^{0.5}\}$, however, we would find that $V(a) = N(a)$ and thus that V is an answer set of P if we were to omit this requirement.

Proposition 4
Let P be a normal program. If M is an answer set of P, there is a valuation V, defined by $V(l) = 1$ if $l \in M$ and $V(l) = 0$ otherwise, and a possibility distribution $\pi \in S_{(P,V)}$ such that for every $l \in Lit_P$ we have $V(l) = N(l)$ (i.e. $N(l) = 1$ if $l \in M$ and $N(l) = 0$ otherwise).

Proof
This proposition is a special case of Proposition 6 presented below. □

We like to point out to the reader that we could try to encode the information in a rule in such a way that we interpret '*not a*' as $\Pi(\neg a)$, which closely corresponds to the intuition of negation-as-failure. Indeed, when it is completely possible to assume that '$\neg a$' is true, then surely '*not a*' is true. Under this encoding, however, we run into a significant problem. Consider the rules $(b \leftarrow not\ c)$ and $(c \leftarrow not\ b)$. These rules would then correspond with the constraints $N(b) \geqslant \Pi(\neg c)$ and $N(c) \geqslant \Pi(\neg b)$, respectively. Note though that both constraints can be rewritten as the constraint $1 - \Pi(\neg b) \geqslant \Pi(\neg c)$. This would imply that both rules are semantically equivalent in ASP, which is clearly not the case. Hence, we cannot directly encode '*not a*' as $\Pi(\neg a)$ and guessing a valuation is indeed necessary since without the guess V we would not be able to obtain a unique set of constraints. As we have shown, this only affects literals preceded by negation-as-failure and we can continue to interpret a literal 'b' as $N(b)$.

https://doi.org/10.1017/S147106841300063X Published online by Cambridge University Press

4 Possibilistic semantics of disjunctive ASP programs

We now turn our attention to how we can characterize disjunctive rules. We found in Section 3 that we can characterize a rule of the form $r = (head \leftarrow body)$ as the constraint $N(head) \geqslant N(body)$, or, similarly, that we can characterize a possibilistic rule $p = (r, \lambda)$ as the constraint $N(head) \geqslant \min(N(body), \lambda)$. Such a characterization works particularly well due the min-decomposability w.r.t. conjunction. Indeed, since the body of e.g. a simple rule $r = (l_0 \leftarrow l_1, ..., l_m)$ is a conjunction of literals, we can write $body = l_1 \wedge ... \wedge l_m$. Then $N(body)$ can be rewritten as $\min(N(l_1), ..., N(l_m))$, which allows for a straightforward simplification. In a similar fashion, for a positive disjunctive rule $r = (l_0; ...; l_k \leftarrow l_{k+1}, ..., l_m)$ we can readily write $N(body)$ as $\min(N(l_{k+1}), ..., N(l_m))$. We would furthermore like to simplify $N(head)$ with $head = l_0 \vee ... \vee l_k$. However, we do not have that $N(head) = \max(N(l_0), ..., N(l_k))$. Indeed, in general we only have that $N(head) \geqslant \max(N(l_0), ..., N(l_k))$. This means that we can either choose to interpret the head as $\max(N(l_0), ..., N(l_k))$ or $N(l_0 \vee ... \vee l_k)$. In particular, a *possibilistic disjunctive rule* $p = (r, \lambda)$ with

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

can either be interpreted as the constraint

$$\max(N(l_0), ..., N(l_k)) \geqslant \min(N(l_{k+1}), ..., N(l_m), 1 - V(l_{m+1}), ..., 1 - V(l_n), \lambda), \quad (5)$$

which we will call the *strong* interpretation of disjunction, or as the constraint

$$N(l_0 \vee ... \vee l_k) \geqslant \min(N(l_{k+1}), ..., N(l_m), 1 - V(l_{m+1}), ..., 1 - V(l_n), \lambda), \quad (6)$$

which we will call the *weak* interpretation of disjunction. In the remainder of this paper, we syntactically differentiate between both approaches by using the notation $l_0; ...; l_k$ and $l_0 \vee ... \vee l_k$ to denote the strong interpretation and the weak interpretation of disjunction, respectively.

The choice of how to treat disjunction is an important one that crucially impacts the nature of the resulting answer sets. For example, the non-deterministic nature of strong disjunction provides a useful way to generate different (candidate) solutions, whereas weak disjunction is oftentimes better suited when we are interested in modelling the epistemic state of an agent, since it amounts to accepting the disjunction as being true rather than making a choice of which disjunct to accept. In this section we consider both characterizations: the characterization of disjunction as (5) is discussed in Section 4.1, and in Section 4.2 we discuss the characterization of disjunction as (6). In particular, we will show that the first characterization of disjunction corresponds to the semantics of disjunction found in ASP, whereas the Boolean counterpart of the second characterization has, to the best of our knowledge, not yet been studied in the literature.

4.1 Strong possibilistic semantics of disjunctive rules

We first consider the characterization of disjunction in which we treat a disjunction of the form '$l_0; ...; l_k$' as $\max(N(l_0), \ldots, N(l_k))$. As it turns out, under these strong possibilistic semantics the disjunction behaves as in classical ASP.

https://doi.org/10.1017/S147106841300063X Published online by Cambridge University Press

Definitiona 6
Let P be a possibilistic disjunctive program and let V be a valuation. For every possibilistic disjunctive rule $p = (r, \lambda)$ with $\lambda \in]0, 1]$ and $r = (l_0; ...; l_k \leftarrow l_{k+1}, ..., l_m, not\ l_{m+1}, ..., not\ l_n)$ the constraint $\gamma_V^s(p)$ induced by p and V is given by

$$\max(N(l_0), ..., N(l_k)) \geqslant \min(N(l_{k+1}), ..., N(l_m), 1 - V(l_{m+1}), ..., 1 - V(l_n), \lambda) \quad (7)$$

$C^s_{(P,V)} = \{\gamma_V^s(p) \mid p \in P\}$ is the set of constraints imposed by program P and V, and $S^s_{(P,V)}$ is the set of all minimally specific possibilistic models of $C^s_{(P,V)}$.[1]

Whenever P is a positive disjunctive program, i.e. whenever P is a disjunctive program without negation-as-failure, (7) is independent of V and we simplify the notation to γ^s, C^s_P and S^s_P.

Note that, unlike in possibilistic logic where a unique least specific possibility distribution exists because of the specific form of the considered constraints, the constraint of the form (7) can give rise to multiple minimally specific possibility distributions, of which some will correspond with answer sets. Indeed, the program $P = \{a; b \leftarrow\}$ induces the constraint $\max(N(a), N(b)) \geqslant 1$, which has two minimally specific possibility distributions, yet no least specific possibility distribution. Indeed, we have the minimally specific possibility distributions π_1, π_2 defined by

$$\begin{array}{llll} \pi_1(\{a,b\}) = 1 & \pi_1(\{b\}) = 0 & \pi_2(\{a,b\}) = 1 & \pi_2(\{b\}) = 1 \\ \pi_1(\{a\}) = 1 & \pi_1(\{\}) = 0 & \pi_2(\{a\}) = 0 & \pi_2(\{\}) = 0. \end{array}$$

Definitiona 7
Let P be a possibilistic disjunctive program and let V be a valuation. Let $\pi \in S^s_{(P,V)}$ 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.

We now further illustrate the semantics and the underlying intuition by considering a possibilistic disjunctive program in detail.

Example 7
Consider the possibilistic (positive) disjunctive program P with the following rules:

0.8: $a; b \leftarrow$
0.6: $c \leftarrow a$
0.4: $c \leftarrow b$.

The constraints C^s_P induced by this program are:

$$\begin{aligned} \max(N(a), N(b)) &\geqslant 0.8 \\ N(c) &\geqslant \min(N(a), 0.6) \\ N(c) &\geqslant \min(N(b), 0.4). \end{aligned}$$

[1] We use the superscript 's' to highlight that we employ the semantics of *s*trong disjunction.

https://doi.org/10.1017/S147106841300063X Published online by Cambridge University Press

From the first constraint it follows that we either need to choose $V(a) = 0.8$ or $V(b) = 0.8$ in accordance with the principal of minimal specificity. Hence, we either obtain $V(c) = 0.6$ or $V(c) = 0.4$. As such we find that the two unique possibilistic answer sets of P are $\{a^{0.8}, c^{0.6}\}$ and $\{b^{0.8}, c^{0.4}\}$.

As before, if we restrict ourselves to rules that are entirely certain, we obtain a characterization of disjunctive programs in classical ASP.

Example 8
Consider the program P with the rules

$$a; b \leftarrow \qquad\qquad a \leftarrow b$$

The set of constraints C_P^{s} is given by

$$\max(N(a), N(b)) \geqslant N(\top) = 1 \qquad\qquad N(a) \geqslant N(b).$$

Intuitively, the first constraint induces a choice. To satisfy this constraint, we need to take either $N(a) = 1$ or $N(b) = 1$. Depending on our choice, we can consider two possibility distributions. The possibility distribution π_1 is the least specific possibility distribution that satisfies the constraints $N(a) = 1$ and $N(a) \geqslant N(b)$, whereas π_2 is the least specific possibility distribution satisfying the constraints $N(b) = 1$ and $N(a) \geqslant N(b)$:

$$\pi_1(\{a, b\}) = 1 \qquad\qquad \pi_1(\{b\}) = 0$$
$$\pi_1(\{a\}) = 1 \qquad\qquad \pi_1(\{\}) = 0$$

and

$$\pi_2(\{a, b\}) = 1 \qquad\qquad \pi_2(\{b\}) = 0$$
$$\pi_2(\{a\}) = 0 \qquad\qquad \pi_2(\{\}) = 0.$$

It is clear that the possibility distribution π_2 cannot be minimally specific w.r.t. the constraints $\max(N(a), N(b)) = 1$ and $N(a) \geqslant N(b)$ since $\pi_1(\{a\}) > \pi_2(\{a\})$ and $\pi_1(\omega) \geqslant \pi_2(\omega)$ for all other interpretations ω. We thus have that S_P^{s} only contains a single element, namely π_1. With N the necessity measure induced by π_1 we obtain $N(a) = 1$ and $N(b) = 0$. As will follow from Propositions 5 and 6, the unique answer set of P is therefore $\{a\}$.
Let us now add the rule $(b \leftarrow \mathit{not}\ b)$ to P. Note that in classical ASP this extended program has no answer sets. The set of constraints $C_{(P,V)}^{\mathrm{s}}$ is given by:

$$C_P^{\mathrm{s}} \cup \{N(b) \geqslant 1 - V(b)\}.$$

This new constraint, intuitively, tells us that 'b' must necessarily be true, since we force it to be true whenever it is not true. Note, however, that the act of making 'b' true effectively removes the motivation for making it true in the first place. As expected, we cannot find any minimally specific possibilistic model that agrees with the constraints imposed by P and V such that $\forall l \in Lit_P \cdot N(l) \in \{0, 1\}$. The problem has to do with our choice of $V(b)$. If we take $V(b) = 1$ then the constraint imposed by the first rule still forces us to choose either $N(a) = 1$ or $N(b) = N(a) = 1$

https://doi.org/10.1017/S147106841300063X Published online by Cambridge University Press

due to the interplay with the constraint imposed by the second rule. However, $S^{s}_{(P,V)}$ contains only one minimally specific possibility distribution, namely the one with $N(a) = 1$. Hence, $N(b) = 0 \neq V(b)$. If we take $V(b) = 0$, then the last rule forces $N(b) = 1$. Hence, $V(b) = 0 \neq 1 = N(b)$.

Now that we have clarified the intuition, we can formalize the connection between the strong possibilistic semantics and classical disjunctive ASP.

Proposition 5
Let P be a disjunctive program, V be a valuation and let $\pi \in S^{s}_{(P,V)}$ such that

$$\forall l \in Lit_P \cdot V(l) = N(l) \text{ ; and} \tag{8}$$

$$\forall l \in Lit_P \cdot N(l) \in \{0, 1\} \tag{9}$$

then $M = \{l \mid N(l) = 1, l \in Lit_P\}$ is an answer set of the disjunctive program P.

The proof is given in the online appendix of the paper (pp. 4–5).

Proposition 6
Let P be a disjunctive program. If M is an answer set of P, there is a valuation V, defined as $V(l) = 1$ if $l \in M$ and $V(l) = 0$ otherwise, and a possibility distribution π, defined as $\pi(\omega) = 1$ if $\omega \models M$ and $\pi(\omega) = 0$ otherwise, such that $\pi \in S^{s}_{(P,V)}$ and for every $l \in Lit_P$ we have $V(l) = N(l)$.

The proof is given in the online appendix of the paper (pp. 5–6)

4.2 Weak possibilistic semantics of disjunctive rules

Under the strong possibilistic semantics of disjunction we consider all the disjuncts of a satisfied rule separately. Under this non-deterministic view the rule $(a; b \leftarrow)$ means that 'a' is believed to be true or 'b' is believed to be true. When looking at answer sets as epistemic states, it becomes apparent that there is also another choice in how we can treat disjunction in the head. Indeed, we can look at the disjunction as a whole to hold, without making any explicit choices as to which of the disjuncts holds. When trying to reason about one's knowledge, there are indeed situations in which we do not want, or simply cannot make, a choice as to which of the disjuncts are true. This implies that we need to look at an answer set as a set of clauses, rather than as a set of literals.

An elaborate example using weak disjunction and uncertainty has been given in Section 1. In this section we consider the semantics of such programs. For starters, we will extend the PASP semantics with the notion of clauses, rather than literals, and define an applicable immediate consequence operator for programs comprising clauses. We then prove some important properties, such as the monotonicity of the immediate consequence operator. For the classical case (i.e. when omitting weights), we furthermore characterize the complexity of clausal programs, both with and without negation-as-failure in Section 5. In particular, we show how the complexity is critically determined by whether we restrict ourselves to atoms and highlight, as shown by the higher complexity of some of the reasoning tasks, that weak disjunction is a non-trivial extension of ASP.

https://doi.org/10.1017/S147106841300063X Published online by Cambridge University Press

We start by formally defining possibilistic clausal programs, i.e. possibilistic programs with a syntax that allows for disjunction in the body. We then define the weak possibilistic semantics of such clausal programs in terms of constraints on possibility distributions. We also introduce an equivalent characterization based on an immediate consequence operator and a reduct, which is more in line with the usual treatment of ASP programs. When all the rules are entirely certain, we obtain the classical counterpart, which we name clausal programs.

4.2.1 Semantical characterization

We rely on the notion of a *clause*, i.e. a finite disjunction of literals. Consistency and entailment for sets of clauses are defined as in propositional logic. As such, we can derive from the information '$a \vee b \vee c$' and '$\neg b$' that '$a \vee c$' is true.

Definitiona 8
A *clausal rule* is an expression of the form $(e_0 \leftarrow e_1, ..., e_m, not\ e_{m+1}, ..., not\ e_n)$ with e_i being a clause for every $0 \leqslant i \leqslant n$. A *positive clausal rule* is an expression of the form $(e_0 \leftarrow e_1, ..., e_m)$, i.e. a clausal rule without negation-as-failure. A *(positive) clausal program* is a finite set of (positive) clausal rules.

For a clausal rule, which is of the form $r = (e_0 \leftarrow e_1, ..., e_m, not\ e_{m+1}, ..., not\ e_n)$, we say that e_0 is the *head* and that $e_1, ..., e_m, not\ e_{m+1}, ..., not\ e_n$ is the *body* of the clausal rule. We use the notation $head(r)$ and $body(r)$ to denote the clause in the head, resp. the set of clauses in the body. The Herbrand base $\mathscr{B}_P$ of a clausal program P is still defined as the set of atoms appearing in P. As such, possibility distributions are defined in the usual way as $\pi : 2^{\mathscr{B}_P} \rightarrow [0, 1]$ mappings.

Until now, we were able to define the possibility distributions that satisfied the constraints imposed by the rules in a program in terms of a valuation V, i.e. a $V : Lit_P \rightarrow [0, 1]$ mapping. This need no longer be the case. Specifically, note that we will now impose constraints of the form $N(l_0 \vee ... \vee l_k) \geqslant \lambda$. Assume that we have a possibility distribution π defined as

$$\pi(\{a, b, c\}) = 0 \qquad \pi(\{a, b\}) = 0 \qquad \pi(\{a, c\}) = 1 \qquad \pi(\{a\}) = 1$$
$$\pi(\{b, c\}) = 0 \qquad \pi(\{b\}) = 0 \qquad \pi(\{c\}) = 1 \qquad \pi(\{\}) = 0.$$

This possibility distribution is the least specific possibility distribution that satisfies the constraints $N(a \vee b \vee c) = 1$ and $N(\neg b) = 1$. However, it can be verified that this possibility distribution cannot be defined in terms of a mapping $V : Lit_P \rightarrow [0, 1]$.

Instead, we define the set of clauses appearing in the head of the rules of a clausal program P as $Clause_P = \{head(r) \mid r \in P\}$. Given a clausal program, it is clear that the only information that can be derived from the program are those clauses that are in the head of a rule. To compactly describe the possibility distribution imposed by clausal programs, we will thus, for the remainder of this section and for Section 5, take a valuation V to be a $Clause_P \rightarrow [0, 1]$ mapping. As before, valuation V corresponds with the set of constraints $\{N(e) \geqslant \lambda \mid e^{\lambda} \in V\}$. The set notation for valuations and the notations V^{λ} and $V^{\underline{\lambda}}$ are extended as usual. Entailment for

https://doi.org/10.1017/S147106841300063X Published online by Cambridge University Press

valuations is defined as in possibilistic logic, i.e. if we consider the least specific possibility distribution π_V satisfying the constraints $\{N_V(e) \geqslant \lambda \mid e^\lambda \in V\}$, then $V \models p^\lambda$ with 'p' a proposition iff $N_V(p) \geqslant \lambda$. In particular, recall from possibilistic logic the inference rules or graded modus ponens (GMP), i.e. we can infer from $N(\alpha) \geqslant \lambda$ and $N(\alpha \rightarrow \beta) \geqslant \lambda'$ that $N(\beta) \geqslant \min(\lambda, \lambda')$. In addition, recall the inference rule (S), i.e. we can infer from $N(\alpha) \geqslant \lambda$ that $N(\alpha) \geqslant \lambda'$ with $\lambda \geqslant \lambda'$.

Definitiona 9
A *possibilistic (positive) clausal program* is a set of possibilistic (positive) clausal rules, which are pairs $p = (r, \lambda)$ with r a (positive) clausal rule and $\lambda \in]0, 1]$ a certainty associated with r.

We define P^* and the λ-cut P_λ as usual.

We are now almost able to define the semantics of weak disjunction. In previous sections we guessed a valuation and used this valuation to deal with negation-as-failure. However, for clausal programs, a new problem arises. Note that the least specific possibility distribution that satisfies the constraints $N(a \vee b \vee c) = 1$ and $N(\neg b) = 1$ is also the least specific possibility distribution that satisfies the constraints $N(a \vee c)$ and $N(\neg b)$. As such, if $Clause_P = \{(a \vee b \vee c), (\neg b), (a \vee c)\}$, there would not be a unique valuation that can be used to define this least specific possibility distribution. Indeed, a valuation uniquely defines a possibility distribution, but not *vice versa*. To avoid such ambiguity, we will instead immediately guess a possibility distribution π_V and use this to deal with negation-as-failure in a clausal program.

Definitiona 10
Let P be a possibilistic clausal program and let π_V be a possibility distribution. For every $p \in P$, the constraint $\gamma^{\mathrm{w}}_{\pi_V}(p)$ induced by $p = (r, \lambda)$ with $\lambda \in]0, 1]$, $r = (e_0 \leftarrow e_1, ..., e_m, not\ e_{m+1}, ..., not\ e_n)$ and π_V under the weak possibilistic semantics is given by

$$N(e_0) \geqslant \min(N(e_1), ..., N(e_m), 1 - N_V(e_{m+1}), ..., 1 - N_V(e_n), \lambda). \tag{10}$$

$C^{\mathrm{w}}_{(P,\pi_V)} = \{\gamma^{\mathrm{w}}_{\pi_V}(p) \mid p \in P\}$ is the set of constraints imposed by program P and π_V, and $S^{\mathrm{w}}_{(P,\pi_V)}$ is the set of all minimally specific possibilistic models of $C^{\mathrm{w}}_{(P,\pi_V)}$.

Whenever P is a possibilistic (positive) clausal program, i.e. whenever P is a possibilistic clausal program without negation-as-failure, (10) is independent of π_V and we simplify the notation to $\gamma^{\mathrm{w}}, C^{\mathrm{w}}_P$ and S^{w}_P.

Definitiona 11
Let P be a possibilistic clausal program. Let π_V be a possibility distribution such that $\pi_V \in S^{\mathrm{w}}_{(P,\pi_V)}$. We then say that π_V is a possibilistic answer set of P.

As already indicated, we can also use valuation V to concisely describe π_V. When we say that V is a possibilistic answer set of the clausal program P, we are, more precisely, stating that the possibility distribution induced by V is a possibilistic answer set of the clausal program P.

https://doi.org/10.1017/S147106841300063X Published online by Cambridge University Press

Lemma 2
Let P be a possibilistic positive clausal program. Then $S^{\mathrm{w}}_{(P,\pi_V)}$ is a singleton, i.e. $\pi \in S^{\mathrm{w}}_{(P,\pi_V)}$ is the least specific possibility distribution.

Proof
This readily follows from the form of the constraints imposed by the rules $p \in P$ and since a possibilistic positive clausal program is free of negation-as-failure. □

Example 9
Consider the possibilistic clausal program P with the rules:

$$\mathbf{1} : a \vee c \vee d \leftarrow$$
$$\mathbf{0.4} : \neg d \leftarrow$$
$$\mathbf{0.8} : e \leftarrow \mathit{not}\ (a \vee b \vee c).$$

We have that $C^{\mathrm{w}}_{(P,\pi_V)}$ is the set of constraints:

$$N(a \vee c \vee d) \geqslant 1$$
$$N(\neg d) \geqslant 0.4$$
$$N(e) \geqslant \min(1 - N_V(a \vee b \vee c), 0.8).$$

We can rewrite the first constraint as $N(\neg d \rightarrow a \vee c) \geqslant 1$. Given the second constraint $N(\neg d) \geqslant 0.4$, we can apply the inference rule (GMP) to conclude that $N(a \vee c) \geqslant 0.4$. From propositional logic we know that $(a \vee c) \rightarrow (a \vee b \vee c)$, i.e. we also have $N(a \vee b \vee c) \geqslant 0.4$.

For π_V to be an answer set of P we know from Definition 11 that we must have that $\pi \in S^{\mathrm{w}}_{(P,\pi_V)}$ with $\pi = \pi_V$. In other words, we must have that $N_V(a \vee b \vee c) = N(a \vee b \vee c) \geqslant 0.4$. Due to the principle of least specificity, which implies that $N(a \vee b \vee c) = 0.4$, the last constraints can be simplified to $N(e) \geqslant \min(1 - 0.4, 0.8)$ or $N(e) \geqslant 0.6$. As such, the least specific possibility distribution defined by the constraints $N(e) \geqslant 0.6$, $N(a \vee c \vee d) \geqslant 1$ and $N(\neg d) \geqslant 0.4$ is a possibilistic answer set of P.

Note that we implicitly defined the possibilistic answer set of the previous example as a valuation, i.e. in terms of clauses that appear in the head. Alternatively, we could thus write that $V = \{e^{0.6}, a \vee b \vee d^1, \neg b^{0.4}\}$ defines the possibilistic answer set of P. This idea will be further developed in Section 4.2.2 to avoid the need to explicitly define a possibility distribution (which would require an exponential amount of space) and instead rely on an encoding of a possibility distribution by a (polynomial) set of weighted clauses.

For the crisp case, we only want clauses that are either entirely certain or completely uncertain, i.e. true or false. To this end, we add the constraint (11), which is similar to (4) from Proposition 3.

Definitiona 12
Let P be a clausal program and $\pi_V \in S^{\mathrm{w}}_{(P,\pi_V)}$ a possibility distribution such that

$$\forall \omega \in \Omega \cdot \pi_V(\omega) \in \{0, 1\} \tag{11}$$

then π_V is called an answer set of P.

https://doi.org/10.1017/S147106841300063X Published online by Cambridge University Press

4.2.2 Syntactic characterization

We now introduce a syntactic counterpart of the semantics for weak disjunction by defining an immediate consequence and reduct operator. As such, it is more in line with the classical Gelfond–Lifschitz approach. In addition, the syntactic approach only needs polynomial size (as we will only consider clauses appearing in the head of the clausal rules). Indeed, what we will do is formalise the idea of using a valuation to determine the possibilistic answer sets of a clausal program rather than relying on an exponential possibility distribution.

Definitiona 13
Let P be a possibilistic positive clausal program. We define the immediate consequence operator T_P^{w} as:

$$T_P^{\mathrm{w}}(V)(e_0) = \max\left\{\lambda \in [0,1] \mid (e_0 \leftarrow e_1, ..., e_m) \in P_\lambda \text{ and } \forall i \in \{1, ..., m\} \cdot V^\lambda \models e_i\right\}.$$

We use $P_{\mathrm{w}}^\star$ to denote the fixpoint obtained by repeatedly applying T_P^{w} starting from the minimal clausal valuation $V = \emptyset$, i.e. the least fixpoint of T_P^{w} w.r.t. set inclusion. When P is a positive clausal program, we take $\lambda \in \{0, 1\}$.

Example 10
Consider the clausal program P with the clausal rules

$$\mathbf{1}: a \vee b \vee c \leftarrow$$
$$\mathbf{0.4}: \neg b \leftarrow$$
$$\mathbf{0.8}: e \leftarrow (a \vee c \vee d).$$

We can easily verify that, starting from $V = \emptyset$, we obtain

$$T_P^{\mathrm{w}}(V)(a \vee b \vee c) = 1 \text{ and}$$
$$T_P^{\mathrm{w}}(V)(\neg b) = 0.4.$$

In the next iteration we furthermore find that

$$T_P^{\mathrm{w}}(T_P^{\mathrm{w}}(V))(e) = 0.4$$

since (**0.8:** $e \leftarrow (a \vee c \vee d)) \in P_{0.4}$ and since $(T_P^{\mathrm{w}}(V))^{0.4} \models a \vee c \vee d$. In addition, this is the least fixpoint, i.e. we have $P_{\mathrm{w}}^\star = \left\{(a \vee b \vee c)^1, \neg b^{0.4}, e^{0.4}\right\}$.

Note that this definition of the immediate consequence operator is a generalization of the immediate consequence operator for possibilistic simple programs (see Definition 1). Indeed, for a possibilistic positive clausal program where all clauses contain only a single literal, i.e. a possibilistic simple program, we have that $P^\star = P_{\mathrm{w}}^\star$. In addition, when all clauses contain only a single literal, we can simplify the immediate consequence operator and simply write $e_i \in V^\lambda$ instead of $V^\lambda \models e_i$.
We now show that the fixpoint obtained from the immediate consequence operator T_P^{w} is indeed the answer set of P.

Proposition 7
Let P be a possibilistic positive clausal program without possibilistic constraint rules. Then $P_{\mathrm{w}}^\star$ is a possibilistic answer set of P.

https://doi.org/10.1017/S147106841300063X Published online by Cambridge University Press

The proof is given in the online appendix of the paper (pp. 6–7).

Thus far we have only considered possibilistic positive clausal programs. If we allow for negation-as-failure, we will also need to generalize the notion of a reduct. As usual, in the classical case we want that an expression of the form '*not e*' is true when '*e*' cannot be entailed. Furthermore, since we are working in the possibilistic case, we want to take the degrees into account when determining the reduct.

Definitiona 14

Given a possibilistic clausal program P and a valuation V, the reduct P^V of P w.r.t. V is defined as:

$$\begin{aligned} P^V = \{\ & ((e_0 \leftarrow e_1, ..., e_m), \min(\lambda_{rule}, \lambda_{body})) \ \mid \ \min(\lambda_{rule}, \lambda_{body}) > 0 \\ & \wedge \lambda_{body} = \max\left\{\lambda \mid \forall i \in \{m+1, ..., n\} \cdot V^{\underline{1-\lambda}} \not\models e_i, \lambda \in [0,1]\right\} \\ & \wedge ((e_0 \leftarrow e_1, ..., e_m, not\ e_{m+1}, ..., not\ e_n), \lambda_{rule}) \in P\} \end{aligned}$$

This definition corresponds with the Gelfond–Lifschitz reduct when we consider crisp clausal programs where each clause consists of exactly one literal. Indeed, if we consider clauses with exactly one literal, we could simplify $\forall i \in \{m+1, ..., n\} \cdot V^{\underline{1-\lambda}} \not\models e_i$ to $\{e_{m+1}, ..., e_n\} \cap V^{\underline{1-\lambda}} = \emptyset$. This new reduct generalises the Gelfond–Lifschitz reduct in two ways. Firstly, we now have clauses, i.e. we now need to verify whether the negative body is not entailed by our guess. Secondly, we need to take the weights attached to the rules, which we interpret as certainties, into account. In particular, the certainty of the reduct of a rule is limited by the certainty of the negative body of the rule and the certainty of the rule itself. In the crisp case these certainty degrees would become trivial.

Proposition 8

Valuation E is a possibilistic answer set of the possibilistic clausal program P without possibilistic constraint rules iff E is a possibilistic answer set of P^E.

The proof is given in the online appendix of the paper (p. 7).

Before we discuss the complexity results, we look at an example to further uncover the intuition of clausal programs.

Example 11

Consider the possibilistic clausal program P with the following rules:

$$\mathbf{0.7}: a \vee b \vee c \leftarrow \qquad \mathbf{0.2}: \neg b \leftarrow \qquad \mathbf{1}: d \leftarrow not\ (a \vee c \vee f) \qquad \mathbf{1}: e \leftarrow not\ c.$$

The reduct P^V with $V = \left\{(a \vee b \vee c)^{0.7}, (\neg b)^{0.2}, d^{0.8}, e^1\right\}$ is then:

$$\mathbf{0.7}: a \vee b \vee c \leftarrow \qquad \mathbf{0.2}: \neg b \leftarrow \qquad \mathbf{0.8}: d \leftarrow \qquad\qquad\qquad \mathbf{1}: e \leftarrow$$

since $V^{1-0.8} \models a \vee c$ but $V^{\underline{1-0.8}} \not\models a \vee c$ and $V^{\underline{1-1}} \not\models c$. We then have that $(P^V)^{\star}_{\mathrm{w}} = \left\{(a \vee b \vee c)^{0.7}, (\neg b)^{0.2}, d^{0.8}, e^1\right\}$, hence V is indeed an answer set of P.

https://doi.org/10.1017/S147106841300063X Published online by Cambridge University Press

5 Complexity results

Before we discuss the complexity results of the weak possibilistic semantics for disjunctive rules (Section 4.2), we first look at the complexity results of both possibilistic normal programs (Section 3.2) and the strong possibilistic semantics for disjunctive rules (Section 4.1). As such, for Propositions 9–12 we once again consider a valuation V for a possibilistic normal/disjunctive program P as a $V : Lit_P \rightarrow [0,1]$ mapping. We find that for possibilistic normal programs the addition of weights does not affect the complexity compared with classical normal programs.

Proposition 9 (*possibilistic normal program; brave reasoning*)
Let P be a possibilistic normal program. The problem of deciding whether there exists a possibilistic answer set V of P such that $V(l) \geqslant \lambda$ is NP-complete.

The proof is given in the online appendix of the paper (p. 8).

Proposition 10 (*possibilistic normal program; cautious reasoning*)
Let P be a possibilistic normal program. The problem of deciding whether for all possibilistic answer sets V of P we have that $V(l) \geqslant \lambda$ is coNP-complete.

The proof is given in the online appendix of the paper (p. 9).

Similarly, we find for possibilistic disjunctive programs under strong disjunctive semantics that the addition of weights does not affect the complexity compared with classical disjunctive programs.

Proposition 11 (*possibilistic disjunctive program; brave reasoning*)
Let P be a possibilistic disjunctive program. The problem of deciding whether there is a possibilistic answer set V such that $V(l) \geqslant \lambda$ is a Σ_2^{P}-complete problem.

The proof is given in the online appendix of the paper (pp. 9–10).

Proposition 12 (*possibilistic disjunctive program; cautious reasoning*)
Let P be a possibilistic disjunctive program. The problem of deciding whether for all possibilistic answer sets V we have that $V(l) \geqslant \lambda$ is a Π_2^{P}-complete problem.

The proof is given in the online appendix of the paper (pp. 10–11).

We now look at the complexity of the weak possibilistic semantics for disjunctive rules for a variety of decision problems and under a variety of restrictions. In particular, throughout this section we look at the complexity of weak disjunction in the crips case that allows us to compare these results against the complexity of the related decision problems in classical ASP and other epistemic extensions of ASP, e.g. Truszczyński (2011) and Vlaeminck *et al.* (2012). As we will see, for certain classes of clausal programs, decision problems exist where weak disjunction is computationally less complex than disjunctive programs while remaining more complex than normal programs.

An overview of the complexity results available in the literature for disjunctive programs as well as the new results for weak disjunction (in the crisp case) which we discuss in the remainder of this section can be found in Table 1.

https://doi.org/10.1017/S147106841300063X Published online by Cambridge University Press

Table 1. *Completeness results for the main reasoning tasks with references*

No NAF, no ¬	Existence	Brave reasoning	Cautious reasoning
Strong disjunction	NP (1)	Σ_2^P (1)	coNP (1)
Weak disjunction	P (6)	P (6)	P (6)
No NAF, ¬	**Existence**	**Brave reasoning**	**Cautious reasoning**
Strong disjunction	NP (1)	Σ_2^P (1)	coNP (1)
Weak disjunction	NP (4)	BH_2 (3)	coNP (5)
NAF, ¬	**Existence**	**Brave reasoning**	**Cautious reasoning**
Strong disjunction	Σ_2^P (2)	Σ_2^P (2)	Π_2^P (2)
Weak disjunction	Σ_2^P (8)	Σ_2^P (7)	Π_2^P (9)

'No NAF' (resp. 'no ¬') indicates results for programs without negation-as-failure (resp. classical negation)

(1) Eiter and Gottlob (1993)
(2) Baral (2003)
(3) Propositions 13 and 14
(4) Corollary 2
(5) Corollary 3
(6) Proposition 15
(7) Propositions 16 and 17
(8) Corollary 5
(9) Corollary 6

Proposition 13 (*weak disjunction, positive clausal program; brave reasoning*)
Let P be a positive clausal program. The problem of deciding whether a clause 'e' is entailed by a consistent answer set E of P is BH_2-hard.

The proof is given in the online appendix of the paper (pp. 11–12).

Proposition 14 (*weak disjunction, positive clausal program; brave reasoning*)
Let P be a positive clausal program. The problem of deciding whether a clause 'e' is entailed by a consistent answer set M of P is in BH_2.

The proof is given in the online appendix of the paper (pp. 12–13).

Corollary 1
Let P be a positive clausal program. The problem of deciding whether a clause 'e' is entailed by a consistent answer set E of P is BH_2-complete.

Corollary 2 (*weak disjunction, positive clausal program; answer set existence*)
Determining whether a positive clausal program P has a consistent answer set is an NP-complete problem.

The proof is given in the online appendix of the paper (p. 14).

https://doi.org/10.1017/S147106841300063X Published online by Cambridge University Press

Corollary 3 (*weak disjunction, positive clausal program; cautious reasoning*)

Cautious reasoning, i.e. determining whether a clause 'e' is entailed by every answer set M of a positive clausal program P is coNP-complete.

The proof is given in the online appendix of the paper (p. 14).

Surprisingly, the expressivity of positive clausal programs under the weak interpretation of disjunction is directly tied to the ability to use classical negation in clauses. If we limit ourselves to positive clausal programs without classical negation, we find that the expressiveness is restricted to P.

In order to see this, let us take a closer look at the immediate consequence operator for clausal programs as defined in Definition 13. When there are no occurrences of classical negation we can simplify this immediate consequence operator to

$$T_P^{\mathrm{w}}(E) = \{e_0 \mid e_0 \leftarrow e_1, ..., e_m \in P \wedge \forall i \in \{1, ..., m\} \cdot \exists e \in E \cdot e \subseteq e_i\}$$

where $e \subseteq e_i$ is defined as the subset relation where we interpret e and e_i as sets of literals, i.e. $e = (l_1 \vee ... \vee l_n)$ is interpreted as $\{l_1, ..., l_n\}$.

Proposition 15

Let P be a positive clausal program without classical negation. We can find the unique answer set of P in polynomial time.

The proof is given in the online appendix of the paper (p. 14).

We now examine the complexity of general clausal programs. We will do this by showing that 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 can be reduced to the problem of determining whether a clause 'e' is entailed by a consistent answer set M of the clausal program P. We start with the definition of our reduction.

Definitiona 15

Let $\phi = \exists X_1 \forall X_2 \cdot p(X_1, X_2)$ be a QBF with $p(X_1, X_2) = \theta_1 \vee ... \vee \theta_n$ a formula in disjunctive normal form with X_i sets of variables. We define the clausal program P_ϕ corresponding to ϕ as

$$P_\phi = \{x \leftarrow not\ \neg x \mid x \in X_1\} \cup \{\neg x \leftarrow not\ x \mid x \in X_1\} \tag{12}$$

$$\cup \{\neg\theta_t \vee sat \leftarrow \mid 1 \leqslant t \leqslant n\} \tag{13}$$

$$\cup \{\leftarrow not\ sat\} \tag{14}$$

with $\neg\theta_t$ the clausal representation of the negation of the formula θ_t, e.g. when $\theta_t = x_1 \wedge \neg x_2 \wedge ... \wedge \neg x_k$ then $\neg\theta_t = \neg x_1 \vee x_2 \vee ... \vee x_k$.

https://doi.org/10.1017/S147106841300063X Published online by Cambridge University Press

Example 12
Given the QBF $\phi = \exists p_1, p_2 \forall q_1, q_2 \cdot (p_1 \wedge q_1) \vee (p_2 \wedge q_2) \vee (\neg q_1 \wedge \neg q_2)$ the clausal program P_ϕ is

$$\begin{aligned} p_1 &\leftarrow not\ \neg p_1 \\ \neg p_1 &\leftarrow not\ p_1 \\ p_2 &\leftarrow not\ \neg p_2 \\ \neg p_2 &\leftarrow not\ p_2 \\ \neg p_1 \vee \neg q_1 \vee sat &\leftarrow \\ \neg p_2 \vee \neg q_2 \vee sat &\leftarrow \\ q_1 \vee q_2 \vee sat &\leftarrow \\ &\leftarrow not\ sat. \end{aligned}$$

Note how $M = \{p_1, p_2, \neg p_1 \vee \neg q_1 \vee sat, \neg p_2 \vee \neg q_2 \vee sat, q_1 \vee q_2 \vee sat\}$ is an answer set of P_ϕ and that $M \models sat$. Accordingly, we find that the QBF is satisfied.

If we take the QBF $\phi' = \exists p_1, p_2 \forall q_1, q_2 \cdot (p_1 \wedge q_1) \vee (p_2 \wedge q_2)$ then the clausal program $P_{\phi'}$ corresponding to ϕ' is the program P_ϕ in which the penultimate rule has been removed. Note how $P_{\phi'}$ has no answer sets, because we are not able to entail '*sat*' from any of the answer sets of $P_{\phi'}$. Indeed, the QBF ϕ' is not satisfiable.

Proposition 16 (*weak disjunction; brave reasoning*)
Let P be a clausal program. The problem of deciding whether a clause 'e' is entailed by a consistent answer set M of P is Σ_2^{P}-hard.

The proof is given in the online appendix of the paper (pp. 14–15).

Proposition 17 (*weak disjunction; brave reasoning*)
Let P be a clausal program. The problem of deciding whether a clause 'e' is entailed by a consistent answer set M of P is in Σ_2^{P}.

The proof is given in the online appendix of the paper (p. 15).

Corollary 4
Let P be a clausal program. The problem of deciding whether a clause 'e' is entailed by a consistent answer set E of P is Σ_2^{P}-complete.

Corollary 5 (*weak disjunction; answer set existence*)
Determining whether a clausal program P has a consistent answer set is an Σ_2^{P}-complete problem.

The proof is given in the online appendix of the paper (p. 15).

Corollary 6 (*weak disjunction; cautious reasoning*)
Cautious reasoning, i.e. determining whether a clause 'e' is entailed by every answer set M of a clausal program P, is Π_2^{P}-complete.

Proof
This problem is complementary to brave reasoning, i.e. we verify that there does not exist an answer set M' of P such that '$\neg e$' is entailed by M'. □

https://doi.org/10.1017/S147106841300063X Published online by Cambridge University Press

6 Related work

The work presented in this paper touches on various topics that have been the subject of previous research. In this section we structure our discussion of related existing work along three main lines. Previous work on the semantics of disjunctive programs is discussed in Section 6.1. In Section 6.2 we look at how ASP and possibility theory have been used in the literature for epistemic reasoning. Finally, in Section 6.3, we look at prior work on characterizing rules with possibility theory and fuzzy logic.

6.1 Semantics of disjunctive programs

Many characterizations of stable models have been proposed in the literature. We refer the reader to Lifschitz (2010) for a concise overview of 13 such definitions. One of the earliest characterizations of stable models was in terms of autoepistemic logic (Moore 1985). Formulas in autoepistemic logic are constructed using atoms and propositional connectives, as well as the modal operator L, which intuitively stands for '*it is believed*'. The characterization of stable models proposed in Gelfond and Lifschitz (1991) based on autoepistemic logic is to look at '*not a*' as the expression '$\neg\mathsf{L}a$', a choice which clearly stands out for its simplicity and intuitiveness. For example, to explain the semantics of the rule $a_0 \leftarrow a_1, ..., a_m, not\ a_{m+1}, ..., not\ a_n$ one would consider the formula $a_1 \wedge ... \wedge a_m \wedge \neg\mathsf{L}a_{m+1} \wedge ... \wedge \neg\mathsf{L}a_n \rightarrow a_0$. Yet this characterization does have some problems. Indeed, it was soon afterwards realized that this correspondence does not hold for programs with classical negation or disjunction in the head. A more involved characterization based on autoepistemic logic that does work for classical negation and disjunction has been proposed in Lifschitz and Schwarz (1993). The idea is to look at literals 'l' that are not preceded by negation as failure as the formula $(l \wedge \mathsf{L}l)$, while one still looks at a literal of the form '*not l*' as the formula $\neg\mathsf{L}l$. In our approach, an expression of the form '*not l*' is essentially identified with $\Pi(\neg l)$, which clearly resembles the first characterization in terms of autoepistemic logic. By staying closer to the Gelfond–Lifschitz reduct, our approach is more elegant in that we do not require a special translation of literals to be able to deal with classical negation and disjunction.

Several authors have already proposed alternatives and extensions to the semantics of disjunctive programs. Ordered disjunction (Brewka 2002) falls in the latter category and allows to use the head of the rule to formulate alternative solutions in their preferred order. For example, a rule such as $l_1 \times ... \times l_k \leftarrow$ represents the knowledge that l_1 is preferred over l_2, which is preferred over l_3 ..., but that at the very least we want l_k to be true. As such it allows for an easy way to express context-dependent preferences. The semantics of ordered disjunction allow certain non-minimal models to be answer sets, hence, unlike the work in this paper, it does not adhere to the standard semantics of disjunctive rules in ASP.

Annotated disjunctions are another example of a framework that changes the semantics of disjunctive programs (Vennekens *et al.* 2004). It is based on the idea that every disjunct in the head of a rule is annotated with a probability. Interestingly, both

https://doi.org/10.1017/S147106841300063X Published online by Cambridge University Press

ordered and annotated disjunction rely on split programs, as found in the possible model semantics (Sakama and Inoue 1994). These semantics provide an alternative to the minimal model semantics. The idea is to split a disjunctive program into a number of normal programs, one for each possible choice of disjuncts in the head, of which the minimal Herbrand models are then the possible models of disjunctive programs. Intuitively this means that a possible model represents a set of atoms for which a possible justification is present in the program. In line with our conclusions for weak disjunction, using the possible model semantics also leads to a lower computational complexity.

Not all existing extensions of disjunction allow non-minimal models. For example, in Buccafurri *et al.* (2002) an extension of disjunctive logic programs is presented which adds the idea of inheritance. Conflicts between rules are then resolved in favour of more specific rules. Such an approach allows for an intuitive way to deal with default reasoning and exceptions. In particular, the semantics allow for rules to be marked as being defeasible and allows to specify an order or inheritance tree among (sets of) rules. Interestingly, the complexity of the resulting system is not affected and coincides with the complexity of ordinary disjunctive programs.

6.2 Epistemic reasoning with ASP and possibility theory

It was argued in Gelfond (1991) that classical ASP, while later proven to have strong epistemic foundations (Loyer and Straccia 2006), is not well suited for epistemic reasoning. Specifically, ASP lacks mechanisms for introspection and can thus not be used to e.g. reason based on cautiously deducible information. At the same time, however, it was shown that extensions of ASP could be devised that do allow for a natural form of epistemic reasoning. The language ASP^K proposed in Gelfond (1991) allows for modal atoms, e.g. $\mathsf{K}a$, where K is a modal operator that can intuitively be read as '*it is known that [a is true]*'. These new modal atoms can in turn be used in the body of rules. The semantics of ASP^K were originally based on a three-valued interpretation (to allow for the additional truth value '*uncertain*'), but later, in Truszczyński (2011), it was shown that this is not essential and that a more classical two-valued possible world structure can also be considered. In addition, further extensions are discussed that allow for epistemic reasoning over arbitrary theories, where it is shown that ASP^K can be encoded within these extensions. The complexity is studied for these extensions and is shown to be brought up on level w.r.t. ASP, e.g. to Σ_3^{P} for disjunctive epistemic programs.

Alternatively, existing extensions of ASP can be used to implement some epistemic reasoning tasks, such as reasoning based on brave/cautious conclusions. This idea is proposed in Faber and Woltran (2009) to overcome the need for an intermediary step to compute the desired consequences of the ASP program P_1 before being fed into P_2. Rather, they propose a translation to manifold answer set programs, which exploit the concept of weak constraints (Buccafurri *et al.* 2000) to allow for such programs to access all desired consequences of P_1 within a single answer set. As such, for problems that can be cast into this particular form, only a single ASP program needs to be evaluated and the intermediary step is made obsolete.

https://doi.org/10.1017/S147106841300063X Published online by Cambridge University Press

As we have mentioned in Section 6.1, the semantics of ASP can also be expressed in terms of autoepistemic logic (Marek and Truszczyński 1991). These semantics have the benefit of making the modal operator explicit, allowing for an extension of ASP that incorporates such explicit modalities to better express exactly which form of knowledge is required. However, since autoepistemic logic treats negation-as-failure as a modality, it is quite hard to extend to the uncertain case. Furthermore, as already discussed, it is shown in Lifschitz and Schwarz (1993) that this characterization does not allow us to treat classical negation or disjunctive rules in a natural way, which weakens its position as a candidate for generalizing ASP from normal programs to e.g. disjunctive programs.

Possibility theory, which can e.g. be used for belief revision, has a strong epistemic notion and shares a lot of commonalities with epistemic entrenchments (Dubois and Prade 1991). Furthermore, in Dubois *et al.* (2012) a generalization of possibilistic logic is studied, which corresponds to a weighted version of a fragment of the modal logic KD. In this logic, epistemic states are represented as possibility distributions, and logical formulas are used to express constraints on possible epistemic states. In this paper we similarly interpret rules in ASP as constraints on possibility distributions, which furthermore allow us to unearth the semantics of weak disjunction.

6.3 Characterization of rules using possibility theory and fuzzy logic

A large amount of research has focused on how possibility distributions can be used to assign a meaning to rules. For example, possibility theory has been used to model default rules (Benferhat *et al.* 1992; Benferhat *et al.* 1997). Specifically, a default rule 'if a then b' is interpreted as $\Pi(a \wedge b) > \Pi(a \wedge \neg b)$, which captures the intuition that when a is known to hold, b is more plausible than $\neg b$ if all that is known is that a holds. In this approach entailment is defined by looking at the least specific possibility distributions which are similar in spirit to our approach for characterizing ASP rules (although the notion of the least specific possibility distribution is defined, in this context, w.r.t. the plausibility ordering on interpretations induced by the possibility degrees).

The work on possibilistic logic (Dubois *et al.* 1994) forms the basis of possibilistic logic programming (Dubois *et al.* 1991). The idea of possibilistic logic programming is to start from a necessity-valued knowledge base, which is a finite set of pairs $(\phi \ \alpha)$, called necessity-valued formulas, with ϕ being a closed first-order formula and $\alpha \in [0, 1]$. Semantically, a necessity-valued formula expresses a constraint of the form $N(\phi) \geqslant \alpha$ on the set of possibility distributions. A possibilistic logic program is then a set of necessity-valued implications. As rules are essentially modelled using material implication, however, the stable model semantics cannot straightforwardly be characterized using possibilistic logic programming. For example, the knowledge base $\{(a \rightarrow b \ \ 1), (\neg b \ \ 1)\}$, which represents the program $\{b \leftarrow a, \neg b \leftarrow\}$, induces that $N(\neg a) = 1$. Indeed, the semantics of this knowledge base indicate that $\Pi(a \wedge \neg b) = 0$ and $\Pi(b) = 0$, i.e. we find that $\Pi(a) = 0$. In other words: a direct encoding using

https://doi.org/10.1017/S147106841300063X Published online by Cambridge University Press

possibilistic logic programming allows for contraposition, which is not in accordance with the stable model semantics.

Rules in logic can also be interpreted as statements of conditional probability (Jaynes 2003). In the possibilistic setting this notion has been adapted to the notion of conditional necessity measures. Rules can then also be be modelled in terms of conditional necessity measures (Benferhat *et al.* 1997; Dubois and Prade 1997; Benferhat *et al.* 2002). The conditional possibility measure $\Pi\left(\psi \mid \phi\right)$ is defined as the greatest solution to the equation $\Pi(\phi \wedge \psi) = \min(\Pi\left(\psi \mid \phi\right), \Pi(\phi))$ in accordance with the principle of least specificity. It can be derived mathematically and this gives us $\Pi\left(\psi \mid \phi\right) = 1$ if $\Pi(\psi \wedge \phi) = \Pi(\phi)$ and $\Pi\left(\psi \mid \phi\right) = \Pi(\psi \wedge \phi)$ otherwise whenever $\Pi(\phi) > 0$. When $\Pi(\phi) = 0$, then by convention $\Pi\left(\psi \mid \phi\right) = 1$ for every $\psi \neq \bot$ and $\Pi\left(\bot \mid \phi\right) = 0$ otherwise. The conditional necessity measure is defined as $N\left(\psi \mid \phi\right) = 1 - \Pi\left(\neg\psi \mid \phi\right)$. However, there does not seem to be a straightforward way to capture the stable model semantics using conditional necessity measures, especially when classical negation is allowed. Indeed, if we represent the semantics of the program $\{b \leftarrow a, \neg b \leftarrow\}$ as the constraints $N\left(b \mid a\right) \geqslant 1$ and $N\left(\neg a \mid \top\right) \geqslant 1$. Using the definition of the conditional necessity measure, the first constraint is equivalent to $1 - \Pi\left(\neg b \mid a\right) \geqslant 1$, i.e. $\Pi\left(\neg b \mid a\right) = 0$. The second constraint simplifies to $\Pi(a) = 0$, which, using the convention stated above gives rise to $\Pi\left(\neg b \mid a\right) = 1$, clearly a contradictory result to the earlier conclusion that $\Pi\left(\neg b \mid a\right) = 0$.

The work in Nicolas *et al.* (2006) was one of the first papers to explore the idea of combining possibility theory with ASP. Rather than defining the semantics of ASP in terms of constraints on possibility distributions as we did in this paper, the goal was to allow one to reason with possibilities in ASP programs. In this way one can associate certainties with the information encoded in an ASP program. The approach from Nicolas *et al.* (2006) upholds the 1-on-1 relationship between the classical answer sets of a normal program and the possibilistic answer sets, which brings with it some advantages. One of those advantages is that it allows us to deal with possibilistic nested programs (Nieves and Lindgren 2012). The work from Nicolas *et al.* (2006) was later extended to also cover the case of disjunctive ASP in Nieves *et al.* (2013). The latter approach allows us to e.g. capture qualitative information by considering partially ordered sets, which would not be straightforward to accomplish in our work. However, the approaches from Nicolas *et al.* (2006) and Nieves *et al.* (2013) work by taking a PASP program and reducing it – by ignoring the certainty values – to a PASP program without negation-as-failure. As such, both approaches loose the certainty encoded through negation-as-failure, since the certainty values are not taken into account.

Possibility theory has also been used to define various semantics of fuzzy if–then rules (Zadeh 1992). Rather than working with literals, fuzzy if–then rules consider fuzzy predicates, each of which has its own universe of discourse. To draw conclusions from a set of fuzzy if–then rules, mechanisms are needed that can produce an (intuitively acceptable) conclusion from a set of such rules.

Finally, a formal connection also exists between the approach from Section 3 and the work on residuated logic programs (Damásio and Pereira 2001) under the Gödel semantics. Both approaches are different in spirit, however, in the same way

https://doi.org/10.1017/S147106841300063X Published online by Cambridge University Press

that possibilistic logic (which deals with uncertainty or priority) is different from the Gödel logic (which deals with graded truth). The formal connection is due to the fact that necessity measures are min-decomposable and disappear as soon as classical negation or disjunction is considered.

7 Conclusions

In this paper we defined semantics for PASP, a framework that combines possibility theory and ASP to allow for reasoning under (qualitative) uncertainty. These semantics are based on the interpretation of possibilistic rules as constraints on possibility distributions. We showed how our semantics for PASP differ from the existing semantics for PASP. Specifically, they adhere to a different intuition for negation-as-failure. As such, they can be used to arrive at acceptable results for problems where the possibilistic answer sets according to the existing semantics for PASP do not necessarily agree with our intuition of the problem. In addition, we showed how our semantics for PASP allowed for a new characterization of ASP. When looking at ASP as a special case of PASP, we naturally recover the intuition of a rule that the head is certain whenever we are certain that the body holds. The resulting characterization stays close to the intuition of the stable model semantics, yet also shares the explicit reference to modalities with autoepistemic logic. We showed that this characterization not only naturally characterizes normal programs, i.e. programs with negation-a-failure, but can also naturally characterize disjunctive programs and programs with classical negation.

Owing to our explicit reference to modalities in the semantics, we are furthermore able to characterize alternative semantics for disjunction in the head of the rule that has a more epistemic flavour than the standard treatment of disjunction in ASP, i.e. given a rule of the form $(a \vee b \leftarrow)$ we do not obtain two answer sets, but rather we have '$a \vee b$' as-is in the answer set. While such a characterization might seem weak, we showed that the interplay with literals significantly affects the expressiveness. Indeed, we found that the problem of brave reasoning/cautious reasoning under these weak semantics for disjunction for a program without negation-as-failure, but with classical negation, is BH_2-complete and coNP-complete, respectively. This highlights that weak disjunction is not merely syntactic sugar, i.e. it cannot simply be simulated in normal ASP without causing an exponential blow-up. For strong disjunction, on the other hand, we have obtained that brave and cautious reasoning without negation-as-failure are Σ_2^{P}-complete and coNP-complete, respectively. As such, the weak semantics for disjunction detailed in this paper allow us to work with disjunction in a less complex way that still remains non-trivial. If, however, we restrict ourselves to atoms, then brave reasoning under the weak semantics for disjunction is P-complete.

References

Banerjee, M. and Dubois, D. 2009. A simple modal logic for reasoning about revealed beliefs. In *Proceedings of the 10th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU'09)*, 805–816.

https://doi.org/10.1017/S147106841300063X Published online by Cambridge University Press

Baral, C. 2003. *Knowledge, Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, Cambridge, UK.

Bauters, K., Schockaert, S., De Cock, M. and Vermeir, D. 2010. Possibilistic answer set programming revisited. In *Proceedings of the 26th Conference on Uncertainty in Artificial Intelligence (UAI)*.

Bauters, K., Schockaert, S., De Cock, M. and Vermeir, D. 2011. Weak and strong disjunction in possibilistic ASP. In *Proceedings of the 11th International Conference on Scalable Uncertainty Management (SUM)*.

Benferhat, S., Dubois, D., Garcia, L. and Prade, H. 2002. On the transformation between possibilistic logic bases and possibilistic causal networks. *International Journal of Approximate Reasoning 29,* 2, 135–173.

Benferhat, S., Dubois, D. and Prade, H. 1992. Representing default rules in possibilistic logic. In *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning (KR)*, 673–684.

Benferhat, S., Dubois, D. and Prade, H. 1997. Nonmonotonic reasoning, conditional objects and possibility theory. *Artificial Intelligence 92,* 1–2, 259–276.

Brewka, G. 2002. Logic programming with ordered disjunction. In *Proceedings of the 18th National Conference on Artificial Intelligence (AAAI)*, 100–105.

Buccafurri, F., Faber, W. and Leone, N. 2002. Disjunctive logic programs with inheritance. *Theory and Practice of Logic Programming 2,* 3, 293–321.

Buccafurri, F., Leone, N. and Rullo, P. 2000. Enhancing disjunctive datalog by constraints. *IEEE Transactions on Knowledge and Data Engineering 12,* 5, 845–860.

Cai, J.-Y., Gundermann, T., Hartmanis, J., Hemachandra, L., Sewelson, V., Wagner, K. and Wechsung, G. 1988. The Boolean hierarchy I: Structural properties. *SIAM Journal on Computing 17,* 6, 1232–1252.

Damásio, C. V. and Pereira, L. M. 2001. Monotonic and residuated logic programs. In *Proceedings of the 6th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU '01)*, 748–759.

Dubois, D., Lang, J. and Prade, H. 1991. Towards possibilistic logic programming. In *Proceedings of the 8th International Conference on Logic Programming (ICLP)*, 581–595.

Dubois, D., Lang, J. and Prade, H. 1994. Possibilistic logic. In *Handbook of Logic for Artificial Intelligence and Logic Programming*, D. M. Gabbay, C. J. Hogger and J. A. Robinson, Eds. Vol. 3. Oxford University Press, Oxford, UK, 439–513.

Dubois, D. and Prade, H. 1991. Epistemic entrenchment and possibilistic logic. *Artificial Intelligence 50,* 2, 223–239.

Dubois, D. and Prade, H. 1997. A synthetic view of belief revision with uncertain inputs in the framework of possibility theory. *International Journal of Approximate Reasoning 17,* 2–3, 295–324.

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

Eiter, T. and Gottlob, G. 1993. Complexity results for disjunctive logic programming and application to non-monotonic logics. In *Proceedings of the 1993 International Logic Programming Symposium (ILPS)*, 266–278.

Faber, W. and Woltran, S. 2009. Manifold answer-set programs for meta-reasoning. In *LPNMR*, Lecture Notes in Computer Science, Vol. 5753, Springer, New York, NY, 115–128.

Gelfond, M. 1987. On stratified autoepistemic theories. In *Proceedings of the 6th National Conference on Artificial Intelligence (AAAI)*, 207–211.

Gelfond, M. 1991. Strong introspection. In *Proceedings of the 9th National Conference on Artificial Intelligence (AAAI'91)*, 386–391.

Gelfond, M. and Lifschitz, V. 1991. Classical negation in logic programs and disjunctive databases. *New Generation Computing 9*, 365–385.

Gelfond, M. and Lifzchitz, V. 1988. The stable model semantics for logic programming. In *Proceedings of the 5th International Conference on Logic Programming (ICLP)*, 1081–1086.

Huth, M. and Ryan, M. 2004. *Logic in Computer Science: Modelling and Reasoning About Systems*. Cambridge University Press, Cambridge, UK.

Jaynes, E. 2003. *Probability Theory: The Logic of Science*. Cambridge University Press, Cambridge, UK.

Lifschitz, V. 2010. Thirteen definitions of a stable model. In *Fields of Logic and Computation*, Lecture Notes in Computer Science, Vol. 6300, 488–503.

Lifschitz, V. and Schwarz, G. 1993. Extended logic programs as autoepistemic theories. In *Proceedings of the 2nd International Workshop on Logic Programming and Nonmonotonic Reasoning*, 101–114.

Loyer, Y. and Straccia, U. 2006. Epistemic foundation of stable model semantics. *Theory and Practice of Logic Programming 6,* 4, 355–393.

Marek, W. and Truszczyński, M. 1991. Autoepistemic logic. *Journal of the ACM 38*, 587–618.

Moore, R. 1985. Semantical considerations on non-monotonic logic. *Artificial Intelligence 29,* 1, 75–94.

Nguyen, L. A. 2005. On the complexity of fragments of modal logics. In *Proceedings of the 5th International Conference on Advances in Modal Logic (AiML'05)*, 249–268.

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

Nieves, J. C. and Lindgren, H. 2012. Possibilistic nested logic programs. In *Technical Communications of the 28th International Conference on Logic Programming (ICLP'12)*, 267–276.

Nieves, J. C., Osorio, M. and Cortés, U. 2013. Semantics for possibilistic disjunctive programs. *Theory and Practice of Logic Programming 13,* 1, 33–70.

Papadimitriou, C. 1994. *Computational Complexity*. Addison-Wesley, Boston, MA.

Pearce, D. 1997. A new logical characterization of stable models and answer sets. In *Proceedings of the 2nd International Workshop on Non-Monotonic Extensions of Logic Programming (NMELP)*, Lecture Notes in Artificial Intelligence, Vol. 1216, 57–70.

Sakama, C. and Inoue, K. 1994. An alternative approach to the semantics of disjunctive logic programs and deductive databases. *Journal of Automated Reasoning 13,* 1, 145–172.

Truszczyński, M. 2011. Revisiting epistemic specifications. In *Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning*, Lecture Notes in Computer Science, Vol. 6565. Springer, Berlin, Germany, 315–333.

Vennekens, J., Verbaeten, S. and Bruynooghe, M. 2004. Logic programs with annotated disjunctions. In *Proceedings of the 20th International Conference on Logic Programming (ICLP)*, Lecture Notes in Computer Science, Vol. 3132, 431–445.

Vlaeminck, H., Vennekens, J., Bruynooghe, M. and Denecker, M. 2012. Ordered epistemic logic: Semantics, complexity and applications. In *Proceedings of the 13th International Conference on Principles of Knowledge Representation and Reasoning (KR'12)*.

Zadeh, L. 1992. Fuzzy logic and the calculus of fuzzy if-then rules. In *Proceedings of the 22nd IEEE International Symposium on Multiple-Valued Logic (ISMVL)*, 480–480.