Noname manuscript No.
(will be inserted by the editor)

Aggregated Fuzzy Answer Set Programming

Jeroen Janssen · Steven Schockaert · Dirk Vermeir · Martine De Cock

Received: date / Accepted: date

Abstract Fuzzy Answer Set programming (FASP) is an extension of answer set programming (ASP), based on fuzzy logic. It allows to encode continuous optimization problems in the same concise manner as ASP allows to model combinatorial problems. As a result of its inherent continuity, rules in FASP may be satisfied or violated to certain degrees. Rather than insisting that all rules are fully satisfied, we may only require that they are satisfied partially, to the best extent possible. However, most approaches that feature partial rule satisfaction limit themselves to attaching predefined weights to rules, which is not sufficiently flexible for most real-life applications. In this paper, we develop an alternative, based on aggregator functions that specify which (combination of) rules are most important to satisfy. We extend upon previous work by allowing aggregator expressions to define partially ordered preferences, and by the use of a fixpoint semantics.

Keywords Answer Set Programming · Fuzzy Logic

1 Introduction

Answer Set Programming (ASP) is a declarative programming language that is especially suitable for modeling combinatorial problems. In ASP, a programmer writes rules of the form $a \leftarrow \alpha$, meaning the atom a should hold whenever the conjunction of the atoms in the set α holds. The semantics of a set of rules, called a program, are determined by certain minimal models, named *answer sets*. For example, a program P_{gc} solving the problem of graph coloring with two colors, viz. coloring the nodes of

J. Janssen · D. Vermeir
Dept. of Computer Science
Vrije Universiteit Brussel, VUB
Pleinlaan 2, B-1050 Brussel, Belgium
E-mail: {jejansse,dvermeir}@vub.ac.be

S. Schockaert · M. De Cock
Dept. of Applied Mathematics and Computer Science
Universiteit Gent, UGent
Krijgslaan 281, B-9000 Gent, Belgium
E-mail: {steven.schockaert,martine.decock}@ugent.be

a graph in such a way that adjacent nodes have different colors, can be written as follows:

$$\begin{aligned}
gen_1 &: \ white(X) \leftarrow not\ black(X)\\
gen_2 &: \ black(X) \leftarrow not\ white(X)\\
sim_1 &: sim(X,Y) \leftarrow white(X), white(Y)\\
sim_2 &: sim(X,Y) \leftarrow black(X), black(Y)\\
constr &: \qquad \bot \leftarrow edge(X,Y), sim(X,Y)
\end{aligned}$$

This program is written in the generate-define-test style of ASP programs. Rules gen_1 and gen_2 are the *generate* rules, which generate a certain graph coloring by stating that in every possible solution a node is either white or black. Rules sim_1 and sim_2 are the *defining* rules, which define when two nodes have a similar color. Rule *constr* is the test part that eliminates solutions in which two adjacent nodes have the same color. To encode a specific problem instance, rules of the form $fact_{a,b} : edge(a,b) \leftarrow$, denoting that there is an edge between node a and b, are added to the above program. The resulting rules are then grounded, which means that a rule such as gen_1 is replaced by a set of rules of the form $\{gen_{1\,a} : white(a) \leftarrow not\ black(a) \mid a \in Nodes\}$, where *Nodes* is the set of nodes of the problem instance. Note that the specific substitution of the variables is denoted using a subscript. The grounded program is then handed off to an answer set solver such as Smodels [71] or DLV [46], which generates the answer sets of the program, if they exist.

For example, consider the graph depicted in Figure 1a. The answer set solver will generate two answer sets, viz. $A_1 = \{edge(a,b), edge(b,a), sim(a,a), sim(b,b), white(a), black(b)\}$ and $A_2 = \{edge(a,b), edge(b,a), sim(a,a), sim(b,b), black(a), white(b)\}$, that correspond to the solutions of the problem instance. Note that for overconstrained problems with no solutions, it is possible that no answer sets exist. For example, for the graph depicted in Figure 1b, no 2-coloring of the graph exists, which is reflected by the non-existence of answer sets for the corresponding program.

In recent years ASP has been extended to handle problems with imperfect information. Most notable are the probabilistic [12, 28, 50, 51, 61, 62, 74] and possibilistic [2, 5, 63, 64] extensions to handle uncertainty, the fuzzy extensions [8, 33, 52–56, 67, 74, 80–82] which allow to encode the intensity to which the predicates are satisfied, and, more generally, many-valued extensions [10, 11, 13–15, 24, 26, 38–41, 43–45, 47, 48, 60, 70, 72, 73, 75, 76].

In this paper we focus on fuzzy answer set programming (FASP). This extension allows to encode continuous optimization problems in a concise manner, similar to how ASP is able to encode discrete optimization problems. In FASP, rules take on the form $a \leftarrow f(b_1, \ldots, b_n; c_1, \ldots, c_m)$, where a,b_i (for $1 \leq i \leq n$), and c_j (for $1 \leq j \leq m$) are atoms and f is a $[0,1]^{n+m} \rightarrow [0,1]$ function that is monotonically increasing in its n first and monotonically decreasing in its m last arguments. Such a rule is *satisfied* when the truth value in $[0,1]$ that is attached to $f(b_1, \ldots, b_n; c_1, \ldots, c_m)$ is lower than or equal to the truth value attached to a. Answer sets are certain fuzzy sets of atoms, i.e. mappings from atoms to $[0,1]$. For example, consider a continuous weighted graph coloring problem, where colors are grey values, and nodes that are adjacent need to be as different in color as possible. In FASP we can model this problem using the following program P_{fgc}:

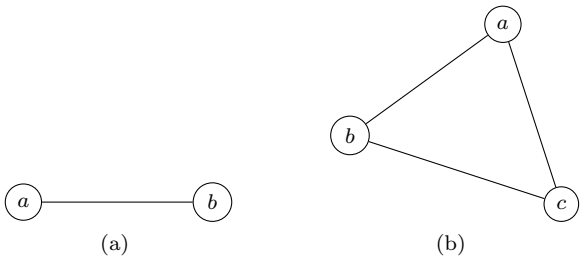

Fig. 1: Example instances for the graph coloring problem.

$$
\begin{aligned}
gen_1 &: \ white(X) \leftarrow \sim_l black(X)\\
gen_2 &: \ black(X) \leftarrow \sim_l white(X)\\
sim_1 &: sim(X,Y) \leftarrow (white(X) \leftrightarrow white(Y))\\
sim_2 &: sim(X,Y) \leftarrow (black(X) \leftrightarrow black(Y))\\
constr &: \quad 0 \leftarrow (edge(X,Y) \wedge_{constr} sim(X,Y))
\end{aligned}
$$

In this program we represent a grey value as a value from $[0,1]$. Hence, $white(X)$, $black(X)$ and $sim(X,Y)$ will be mapped to a value in $[0,1]$, but $edge(X,Y)$ is mapped to a value in $\{0,1\}$. Rules gen_1 and gen_2 are again *generate* rules, which generate a certain grey scale coloring. The function $\sim_l$ is the fuzzy negator function $\sim_l x = 1-x$, i.e. a generalization of negation in classical logic. Rules sim_1 and sim_2 are the *defining* part of the program, which defines the similarity between the colors of adjacent nodes. The function $\leftrightarrow$ is a similarity relation defined as $x \leftrightarrow y = (x \rightarrow y) \wedge (y \rightarrow x)$, where $\wedge$ is a t-norm (a generalization of classical conjunction) and $\rightarrow$ is an implicator (a generalization of classical implication). The *constr* rule is a *constraint* which ensures $edge(X,Y) \wedge_{constr} sim(X,Y) \leq 0$. The function $\wedge_{constr}$ is a t-norm. Furthermore, as $edge(X,Y)$ will be attached a value in $\{0,1\}$, this constraint ensures that any solution satisfies $sim(X,Y) \leq 0$ if there is an edge between X and Y, i.e. the colors must be as dissimilar as possible. Coloring the graph depicted in Figure 1a with $x \leftrightarrow y = \min(x \rightarrow y, y \rightarrow x)$ and $\rightarrow$ an arbitrary (residual) implicator, leads to two fuzzy answer sets, viz. $M_1 = \{edge(a,b)^1,\ edge(b,a)^1,\ sim(a,a)^1,\ sim(b,b)^1,\ white(a)^1,\ black(b)^1\}$ and $M_2 = \{edge(a,b)^1,\ edge(b,a)^1,\ sim(a,a)^1,\ sim(b,b)^1,\ black(a)^1,\ white(b)^1\}$, where $M = \{a_1^{k_1}, \ldots, a_n^{k_n}\}$, with $k_i \in [0,1]$, means $M(a_i) = k_i$ for $i \in 1 \ldots n$, and $M(a) = 0$ otherwise. Note that these correspond to the answer sets of P_{gc}.

Similar to the ASP program, the FASP program corresponding to the graph depicted in Figure 1b has no answer set. This is not ideal, however, as a coloring that colors node a white, node b black, and node c grey may be better than having no solution at all. The inadmissibility is due to the constraint *constr*, which removes solutions where nodes have a similarity which is strictly greater than 0. A possible alternative is to allow solutions in which the similarity degree of two adjacent nodes may be greater than zero. Of course, solutions in which this degree is as small as possible are still preferred. This idea can be implemented by allowing that the last rule, *constr*, should not always be completely satisfied. Many of the current approaches (for example [10, 11, 24, 44, 45, 47, 48, 52, 53, 55, 56, 70]) that allow *partial rule satisfaction* do so by coupling a weight to each rule, thus predefining to what degree each of the rules should be satisfied.

Attaching weights to rules is not an entirely satisfactory solution, however. First of all, having weights puts an additional burden on the programmer, who, moreover, may not always be aware of which weights are suitable. Second, we are not only interested in finding any solution: if multiple solutions can be found, we are especially interested in the solution modeling the rules best. Hence, based on the degree to which the rules of the program are satisfied, it is of interest to define an ordering on the solutions, which cannot be meaningfully done using weights. In [80] the proposed solution is to attach an aggregator expression to a program. This aggregator expression maps each prospective solution to a score, based on the satisfaction degrees of the rules. In this paper, we further develop this approach. In particular, the main contribution of the paper is two-fold. First, we decouple the order structure used by the aggregator expression from the lattice underlying the truth values. This ensures we can define preference orderings on answer sets which may not correspond to complete lattices. Second, our approach is based on a fixpoint semantics, rather than unfounded sets. As we show below, the approach from [80] does not correctly generalize to arbitrary truth lattices, an issue which is solved by our proposed fixpoint semantics. In addition, the fixpoint semantics are also more general, as it is not restricted to formulas that are built from t-norms. Last, the fixpoint semantics (more clearly) reveal the link between aggregated FASP and FASP approaches with weighted rules.

The structure of the paper is as follows. In Section 2 we recall the basic definitions from fuzzy set theory and fuzzy answer set programming. In section 3 we develop a fixpoint theory for fuzzy answer set programming with aggregators and investigate the main properties of this theory. While many of the properties we find are unsurprising, in the sense that they have a direct counterpart in fixpoint theory for (F)ASP, there are some notable differences as well. For instance, while one of the central properties of (F)ASP is that programs without negation have unique answer sets, it turns out that such programs can have several non-trivial answer sets in our setting, depending on the aggregator that is chosen. We apply AFASP on the reviewer assignment problem in Section 4, followed by a detailed overview of the relationship between AFASP and existing approaches in Section 5. Last, we conclude in Section 6.

A preliminary version of this paper appeared in [37].

2 Preliminaries

2.1 Fuzzy Logic Connectives

Recall that a **preorder** is a tuple $\mathcal{P} = (P, \leq)$ such that $(\leq) \subseteq P \times P$ is reflexive and transitive. For a specific preorder $\mathcal{P}$, we denote its ordering as $\leq_{\mathcal{P}}$. A **partial order** is a preorder that is anti-symmetric. A **lattice** $\mathcal{L} = (L, \leq)$ is a partially ordered set in which each tuple $(l, l') \in L^2$ has an infimum and supremum, denoted as $l \sqcap l'$, respectively $l \sqcup l'$. A lattice $\mathcal{L} = (L, \leq)$ is called **bounded** if L has a least and greatest element, denoted as $0_{\mathcal{L}}$, respectively $1_{\mathcal{L}}$. If the lattice that is used is clear from the context we simply write 0 and 1. A **complete lattice** is a lattice $\mathcal{L} = (L, \leq)$ in which each non-empty subset of L has an infimum and supremum. A **residuated lattice** $\mathcal{L} = (L, \leq, \wedge, \rightarrow)$ is a four-tuple where $(L, \leq)$ is a bounded lattice, $(L, \wedge, 1_{\mathcal{L}})$ is a monoid (not necessarily commutative) and the two $\mathcal{L}^2 \to \mathcal{L}$ operators $\wedge$ and $\rightarrow$ satisfy the **residuation principle**, i.e. for all $x, y, z \in \mathcal{L}$ we have $x \wedge z \leq y$ iff $z \leq x \rightarrow y$. The tuple $(\wedge, \rightarrow)$ is called a **residual pair**.

t-norm	t-conorm
$x \wedge_m y = \min(x, y)$	$x \vee_m y = \max(x, y)$
$x \wedge_l y = \max(0, x + y - 1)$	$x \vee_l y = \min(x + y, 1)$
$x \wedge_p y = x \cdot y$	$x \vee_p y = x + y - x \cdot y$

Table 1: Common fuzzy t-norms and t-conorms over $([0, 1], \leq)$

t-norm	residual implicator	induced negator
$\wedge_m$	$x \to_m y = \begin{cases} y & \text{if } x > y \\ 1 & \text{otherwise} \end{cases}$	$\sim_m x = \begin{cases} 0 & \text{if } x > 0 \\ 1 & \text{otherwise} \end{cases}$
$\wedge_l$	$x \to_l y = \min(1, 1 - x + y)$	$\sim_l x = 1 - x$
$\wedge_p$	$x \to_p y = \begin{cases} y/x & \text{if } x > y \\ 1 & \text{otherwise} \end{cases}$	$\sim_p x = \sim_m x$

Table 2: Common residual pairs and induced negators over $([0, 1], \leq)$

In general, fuzzy logics are logics whose semantics are defined in terms of variables that can take a truth value from some complete lattice $\mathcal{L}$ (called a truth lattice) instead of only the values true and false. Different ways exist to extend the classical logic connectives, leading to many logics with differing tautologies and axiomatizations [32, 66]. We briefly recall the most important concepts related to fuzzy logic connectives.

A **negator** is a decreasing $\mathcal{L} \to \mathcal{L}$ mapping $\sim$ satisfying $\sim 0 = 1$ and $\sim 1 = 0$. A negator is called **involutive** iff for each $x \in \mathcal{L}$ we have $\sim (\sim x) = x$. A **triangular norm** (t-norm) is an increasing, commutative and associative $\mathcal{L}^2 \to \mathcal{L}$ operator $\wedge$ satisfying for each $x \in \mathcal{L}$ the equation $1 \wedge x = x$. Intuitively, this operator corresponds to logical conjunction. A **triangular co-norm** (t-conorm) is an increasing, commutative and associative $\mathcal{L}^2 \to \mathcal{L}$ operator $\vee$ satisfying for each $x \in \mathcal{L}$ the equation $0 \vee x = x$. Intuitively, this operator corresponds to logical disjunction. An **implicator** $\to$ is a $\mathcal{L}^2 \to \mathcal{L}$ operator that is decreasing in its first and increasing in its second argument, satisfies $0 \to 0 = 1$ and for all $x \in \mathcal{L}$ satisfies $1 \to x = x$. Every t-norm $\wedge$ whose partial mappings are sup-morphisms (i.e. for which $\sup_i(x_i \wedge y) = (\sup_i x_i) \wedge y$ for any family $(x_i)_{i \in I}$) induces a **residual implicator** defined by $x \to y = \sup\{\lambda \in \mathcal{L} \mid x \wedge \lambda \leq y\}$ (see [16]). Furthermore, any such t-norm $\wedge$ and its residual implicator $\to$ satisfy the **residuation principle**. For a given implicator $\to$ its induced negator is the operator $\sim$ defined by $\sim x = x \to 0$. We summarized some common t-norms, t-conorms, residual implicators, and induced negators over the well-known complete lattice $([0, 1], \leq)$ in Tables 1 and 2.

An **$\mathcal{L}$-fuzzy set** A in a universe X is an $X \to \mathcal{L}$ mapping. When $\mathcal{L} = ([0, 1], \leq)$, the mapping is simply called a fuzzy set. For $x \in X$ we call $A(x)$ the **membership degree** of x in A. For convenience we denote with $A = \{a_1^{k_1}, \ldots, a_n^{k_n}\}$ the fuzzy set A satisfying $A(a_i) = k_i$ for $1 \leq i \leq n$ and $A(a) = 0$ for $a \notin \{a_1, \ldots, a_n\}$. We use $\mathcal{F}_{\mathcal{L}}(X)$ to denote the universe of all $\mathcal{L}$-fuzzy sets in X. The **support** of a fuzzy set A is defined by $supp(A) = \{x \in X \mid A(x) > 0\}$. Inclusion of fuzzy sets in the sense of Zadeh is defined as $A \subseteq B$ iff $\forall x \in X \cdot A(x) \leq B(x)$.

2.2 Fuzzy Answer Set Programming

Definition 1 Consider a complete lattice $\mathcal{L}$. A **rule** over $\mathcal{L}$ is an object of the form $r : a \leftarrow \alpha$ where r is the **label** of the rule, $\leftarrow$ corresponds to a residual implicator, a is either an atom or a value from $\mathcal{L}$ and α is of the form $f(b_1, \ldots, b_n; c_1, \ldots, c_m)$, with b_i and c_j (for $1 \leq i \leq n$ and $1 \leq j \leq m$) either atoms or values from $\mathcal{L}$, and where f is a (total and finite-time computable) $\mathcal{L}^{n+m}$ function that is increasing in its first n arguments and decreasing in its m last arguments. The atom a is called the **head** of the rule and α is called the **body**. For a given rule r we denote the head with r_h, the body with r_b, and the t-norm that induces the residual implicator $\leftarrow$ with $\wedge_r$. Furthermore we define the **Herbrand Base** of a rule r, denoted $\mathcal{B}_r$, as the set of atoms that occur in it. A rule with a value from $\mathcal{L}$ in its head is called a **constraint**, whereas a rule with a value from $\mathcal{L}$ as its body is called a **fact**.

When there is no cause for confusion, we will often use the label of the rule to denote the rule itself.

Definition 2 Consider a complete lattice $\mathcal{L}$. A **FASP program** is a finite set of rules over $\mathcal{L}$. For a given program P the **Herbrand Base** is defined by $\mathcal{B}_P = \bigcup_{r \in P} \mathcal{B}_r$. Furthermore, we denote the lattice over which the rules in P are defined with $\mathcal{L}_P$. An **interpretation** I of P is a $\mathcal{B}_P \to \mathcal{L}_P$ mapping. It is extended to values from $\mathcal{L}_P$ by $I(l) = l$ for any $l \in \mathcal{L}_P$, to expressions $f(b_1, \ldots, b_n; c_1, \ldots, c_m)$ (with b_i and c_j in $\mathcal{B}_P \cup \mathcal{L}_P$) by

$$I(f(b_1, \ldots, b_n; c_1, \ldots, c_m)) = f(I(b_1), \ldots, I(b_n); I(c_1), \ldots, I(c_m))$$

and to rules $r : a \leftarrow \alpha \in P$ by

$$I(r) = I(a \leftarrow \alpha) = I(a) \leftarrow I(\alpha)$$

We say an interpretation I satisfies the rule $r \in P$ to degree $k \in \mathcal{L}_P$ iff $I(r) \geq k$.

Note that for any residual pair $(\wedge, \rightarrow)$ it holds that

$$I(a \leftarrow \alpha) \geq k \text{ iff } I(a) \geq (k \wedge I(\alpha)) \tag{1}$$

due to the residuation principle. In the special case where the rule is completely satisfied, we thus obtain

$$I(a \leftarrow \alpha) = 1 \text{ iff } I(a) \geq I(\alpha) \tag{2}$$

Hence, residual implicators implement the intuitive requirement that, in order for a rule to be satisfied, the higher the truth degree of its body is, the higher the truth degree of its head should be.

3 Aggregated FASP

3.1 Models

Normally, given a FASP program P, one is interested in the interpretations I satisfying for each $r \in P$ that $I(r) = 1$, which are usually called **models**. In the present framework, however, we recognize that rules cannot always be completely fulfilled. This has

two main advantages: first, we can tackle problems lacking a "perfect" solution (i.e. a solution satisfying all the rules to a degree of 1) and second, we can find satisfactory solutions faster if we do not need a "perfect" solution.

The first situation can occur when problems are *overconstrained*. For example, consider the graph coloring problem introduced in Section 1. For the graph depicted in Figure 1b, we mentioned in Section 1 that no coloring exists that will satisfy the constraint rule. Hence the problem is overconstrained and we must make some compromises, adhering stricter to rules that are considered more important. The second situation can occur when the time it takes to compute a solution is more important than finding a solution satisfying all rules.

The behavior of rules that are only partially satisfied depends crucially on the choice of the residual implicator. For example, rule $a \leftarrow \alpha$ with $k = I(a) < I(\alpha)$ is satisfied to degree k, $k \cdot I(\alpha)$ or $I(\alpha) - (1 - k)$ depending on whether the Gödel resp. Goguen or Łukasiewicz implicator is used. Thus, depending on which implicator is used, the degree to which the head should be satisfied, depends 1) not at all on $I(\alpha)$; 2) proportionally on $I(\alpha)$; or 3) linearly on $I(\alpha)$. Depending on the context, each of these three situations may be required.

To create our framework, we first define **rule interpretations**, which are functions that map rules to a truth value (i.e. a value from the considered truth lattice).

Definition 3 Given a FASP program P over the lattice $\mathcal{L}$, a **rule interpretation** of P is a $P \to \mathcal{L}$ mapping. Any interpretation I of P induces a rule interpretation ρ_I defined as $\forall(r : a \leftarrow \alpha) \in P \cdot (\rho_I(r) = I(a \leftarrow \alpha))$.

Hence, the difference between the interpretation of a program and a rule interpretation is that the former maps propositional symbols to truth values, whereas the latter maps the rules themselves to a truth value. We extend the lattice ordering used to construct the rules of a program P to the rule interpretations in a pointwise manner, i.e. for ρ_1 and ρ_2 rule interpretations, we define $\rho_1 \leq \rho_2$ iff $\forall r \in P \cdot \rho_1(r) \leq \rho_2(r)$. The relative importance of rules is encoded in an **aggregator** function.

Definition 4 An **aggregator** over a program P and preorder $\mathcal{P}$ is an order-preserving $(P \to \mathcal{L}_P) \to \mathcal{P}$ function.

Hence, an aggregator maps rule interpretations to preference scores from some preorder. As it is order-preserving, we guarantee that rule interpretations that map the rules to a higher degree receive a higher score. The aggregator typically encodes which rules are deemed more important by the designer, who may be more reluctant to accept solutions that do poorly on some rules while not caring much about failing to fully satisfy others.

Example 1 Consider the fuzzy graph coloring program P_{fgc} and Figure 1b introduced in Section 1 and suppose edge (a, c) is more important than edge (b, c), which is more important than edge (a, b). To represent this we can use an aggregator over the preorder $\mathcal{P} = (\mathbb{R}, \leq)$:

$$\mathcal{A}(\rho) = \beta(\rho) \times (0.7 \cdot \rho(\mathit{constr}_{(a,c)}) + 0.7 \cdot \rho(\mathit{constr}_{(c,a)}) + 0.5 \cdot \rho(\mathit{constr}_{(b,c)}) + 0.5 \cdot \rho(\mathit{constr}_{(c,b)}) + 0.4 \cdot \rho(\mathit{constr}_{(a,b)}) + 0.4 \cdot \rho(\mathit{constr}_{(b,a)}))$$

where β is a function mapping rule interpretations to a value in $\mathbb{R}$ that is defined as $\beta(\rho) = ((\prod\{\rho((\mathit{gen}_1)_a), \rho((\mathit{gen}_2)_a) \mid a \in \mathit{Nodes}\}) \times (\prod\{\rho((\mathit{sim}_1)_{a,b}), \rho((\mathit{sim}_2)_{a,b}) \mid$

$a, b \in \mathit{Nodes}\})) \geq 1$). Hence, if the generate rules or similarity rules are not satisfied to degree 1, the score of an interpretation of the rules will always be 0; otherwise it is the weighted sum of the constraint rules. Now consider two rule interpretations ρ_1 and ρ_2 of this program such that $\beta(\rho_1) = \beta(\rho_2) = 1$ and with the following values for the constraints:

	$\mathit{constr}_{(a,b)}$	$\mathit{constr}_{(b,a)}$	$\mathit{constr}_{(a,c)}$	$\mathit{constr}_{(c,a)}$	$\mathit{constr}_{(b,c)}$	$\mathit{constr}_{(c,b)}$
ρ_1	0.3	0.3	0.7	0.7	0.7	0.7
ρ_2	1	1	0.5	0.5	0.5	0.5

Computing $\mathcal{A}(\rho_1)$ and $\mathcal{A}(\rho_2)$ we obtain $\mathcal{A}(\rho_1) = 1.92$ and $\mathcal{A}(\rho_2) = 2$. Hence according to this aggregator, a solution satisfying the rules to the degrees specified by rule interpretation ρ_2 is better than a solution satisfying the rules to the degrees specified by rule interpretation ρ_1.

Depending on the application, this might not be what we want. For example, the fact that edge (a, c) is more important can also mean that ρ_1 should be more preferred than ρ_2, since it satisfies this rule better. This can be represented using an aggregator over the partial order $\mathcal{P}' = ([0, 1]^3, \leq_{lex})$ where $(a_1, a_2, a_3) \leq_{lex} (a'_1, a'_2, a'_3)$ is defined as:

$$\begin{aligned}(a_1, a_2, a_3) \leq_{lex} (a'_1, a'_2, a'_3) \equiv &(\forall i \in 1..3 \cdot a_i \leq a'_i) \\ &\vee (\exists i \in 1 \ldots 3 \cdot (a_i <_{\mathcal{L}} a'_i) \wedge (\forall j < i \cdot a_j = a'_j))\end{aligned}$$

The aggregator is

$$\mathcal{A}'(\rho) = \begin{cases} (0, 0, 0) & \text{if } \beta(\rho) = 0 \\ (\rho(\mathit{constr}_{(a,c)}), \rho(\mathit{constr}_{(b,c)}), \rho(\mathit{constr}_{(a,b)})) & \text{otherwise} \end{cases} \tag{3}$$

Note that since $\mathit{constr}_{(x,y)} = \mathit{constr}_{(y,x)}$, for any $x, y \in \mathit{Nodes}$, we do not need to take the constraints $\mathit{constr}_{(b,a)}$, $\mathit{constr}_{(c,b)}$ and $\mathit{constr}_{(c,a)}$ into account. Using $\mathcal{A}'$, we obtain $\mathcal{A}'(\rho_1) = (0.7, 0.7, 0.3)$ and $\mathcal{A}'(\rho_2) = (0.5, 0.5, 1)$, from which we obtain that $\mathcal{A}'(\rho_2) \leq_{lex} \mathcal{A}'(\rho_1)$, i.e. ρ_1 will indeed be strictly preferred over ρ_2.

Many aggregation strategies have been proposed over the years (for an overview see [18]). Formally, an aggregation operator $\mathcal{A}$ is defined as a function mapping vectors over $\mathcal{L}^n$, with $\mathcal{L}$ a complete lattice, to a preorder $\mathcal{P}$. It can easily be seen that the aggregator defined in Definition 4 above fits this definition as rule interpretations correspond to vectors in $\mathcal{L}^n$, with n the number of rules in a program. In the following, let $u = (u_1, \ldots, u_n)$ and $v = (v_1, \ldots, v_n)$ be two vectors in $\mathcal{L}^n$. The main task of an aggregation operator is to define an ordering over the vectors in $\mathcal{L}^n$, which we call the **induced ordering** of an aggregation operator $\mathcal{A}$. Formally for an aggregation operator $\mathcal{A}$ it is defined as

$$u \leq_{\mathcal{A}} v \equiv \mathcal{A}(u) \leq_{\mathcal{P}} \mathcal{A}(v)$$

In some cases the aggregator is just the identity function, and thus $u \leq_{\mathcal{A}} v \equiv u \leq_{\mathcal{P}} v$, with $\leq_{\mathcal{P}}$ an ordering over vectors in $\mathcal{L}^n$. This is for example the case with the **Pareto aggregator** and **lexicographical aggregator**. Formally the former maps a vector to a preorder $(\mathcal{L}^n, \leq_{par})$ with $\leq_{par}$ defined as

$$u \leq_{par} v \equiv \forall i \in 1 \ldots n \cdot u_i \leq_{\mathcal{L}} v_i$$

The latter maps a vector to a preorder $(\mathcal{L}^n, \leq_{lex})$ defined as

$$u \leq_{lex} v \equiv (u \leq_{par} v) \vee (\exists i \in 1 \ldots n \cdot (u_i <_{\mathcal{L}} v_i) \wedge (\forall j < i \cdot u_j = v_j))$$

In other cases the aggregator maps vectors to a single value in $(\mathcal{L}, \leq_{\mathcal{L}})$ or $(\mathbb{R}, \leq)$. Well-known aggregation operators of this form are the minimum, maximum, median, product and sum. Though these operators are useful, sometimes, we may consider the satisfaction of some rules more important, e.g. expressed using priority levels for each rule. To cope with such priority levels, weighted versions of the basic operators have been proposed, such as the weighted sum used in Example 1. For weighted minimum and maximum we refer the reader to [22, 25, 83]. A particularly interesting class of operators with weights are the Ordered Weighted Average (OWA) operators (see e.g. [84, 85]), which encompass a wide range of aggregation operators over vectors in $[0, 1]^n$, including the minimum, maximum and median. Formally, given a vector $u = (u_1, \ldots, u_n) \in [0, 1]^n$, a collection of weights $(w_1, \ldots, w_n) \in [0, 1]^n$ such that $\sum_i w_i = 1$ and a permutation σ of u such that $u_{\sigma(1)} \leq \ldots \leq u_{\sigma(n)}$, an OWA operator is defined by

$$OWA(u) = \sum_{j=1}^{n} w_j x_{\sigma(j)} \tag{4}$$

By manipulating the weights of the OWA operator, particular aggregation operators are obtained, with the minimum and the maximum as extreme cases, corresponding to the weight vectors $(1, 0, \ldots, 0)$ and $(0, \ldots, 0, 1)$ respectively. Hence, the aggregated value will always be in between the minimum and the maximum of their arguments. Interestingly, the weights of an OWA operator are not associated to a source, but to an ordered position. This makes it ideal for applications where certain outliers should not be taken into account, such as for example in the judging of olympic sports, where the most extreme scores do not count for the final score. Furthermore OWA's can be used to model the demand that most of the rules should be fulfilled, or at least a few rules should be fulfilled.

Two other families of aggregation operators allow to model interaction between values, viz. the discrete **Sugeno** integral [77] and the **Choquet** integral [9]. The difference between these aggregators is that the Sugeno integral is more suitable for ordinal aggregation (where only the order of elements is important) while the Choquet integral is suitable for cardinal aggregation (where the distance between the numbers has a meaning) [18]. Interestingly, the Sugeno integral generalizes the weighted minimum and the weighted maximum, and the Choquet integral generalizes the weighted mean and the OWA operator. The downside of these operators is the high number of weights that need to be provided by the user. To aggregate n values, in principle, the user needs to supply 2^n weights, which clearly is rather cumbersome. However, in some cases one can reduce the number of required weights, see for example [1, 31].

One can also use **t-norms** and **t-conorms** for aggregation. However, the aggregated value of these operators is not in between the minimum and the maximum, which is useful in certain applications. Two classes of operators that do have this feature can be constructed based on t-norms and t-conorms, however, viz. the **exponential compensatory operators** [79] and the **convex-linear compensatory operators** [49,79]. Another related class of operators are **uninorms** [27], which generalize t-norms and t-conorms. Contrary to the two aforementioned compensatory operators, uninorms satisfy the full reinforcement property, i.e. the tendency that when we collect a number of high scores, the aggregated value will be greater than the maximum of

these scores, and similarly when we collect a number of low scores, the aggregated value will be lower than the minimum of these scores. In some cases this follows the human aggregation process more closely.

Last, although the min and max operators are useful because they work in a strictly ordinal manner, the ordering induced by these operators can sometimes be too coarse. For example, using the minimum, the vectors $(0.1, 0.5)$ and $(0.1, 0.1)$ would be equally preferred as $\min(0.1, 0.5) = \min(0.1, 0.1)$. However, the first vector clearly has a better score for the second value to be aggregated. To cope with this, refinements of the induced ordering have been proposed, namely the **discrimin** and the **leximin** (see e.g. [23]). Formally the discrimin aggregator maps a vector to the corresponding element in the structure $(\mathcal{L}^n, \leq_{disc})$ defined as

$$u \leq_{disc} v \equiv \min\{u_i \mid i \in \mathcal{D}(u, v)\} \leq \min\{v_i \mid i \in \mathcal{D}(u, v)\} \tag{5}$$

where $\mathcal{D}(u, v) = \{i \mid i \in \mathbb{N}, u_i \neq v_i\}$. Intuitively, this ordering is based on the idea that the values in which the two vectors agree are of no importance when comparing them. Decisions are thus based on the least satisfied discriminating value. The idea of the leximin aggregator is to represent vectors of satisfaction levels by ranked multi-sets of satisfaction degrees. Formally it maps a vector in $\mathcal{L}^n$ to the corresponding element in the structure $(\mathcal{L}^n, \leq_{lexi})$ defined as

$$u \leq_{lexi} v \equiv \exists k \leq n \cdot \forall i < k \cdot (u_{\sigma(i)} = v_{\sigma(i)}) \wedge (u_{\sigma(k)} \leq v_{\sigma(k)}) \tag{6}$$

where for a given vector u, σ is a permutation of u such that $u_{\sigma(1)} \leq \ldots \leq u_{\sigma(n)}$. Hence, two vectors are indifferent if the corresponding reordered vectors are the same. The leximin ordering is a refinement of both the minimum and the discrimin [19]. Similarly two refinements of the maximum, called **discrimax** and **leximax**, can be defined.

An **aggregated FASP program** then consists of a FASP program and an aggregator function over this program.

Definition 5 An **aggregated FASP program** (short: **AFASP program**) P is a tuple $\langle \mathcal{R}, \mathcal{A} \rangle$, where $\mathcal{R}$ is a FASP program over a lattice $\mathcal{L}$, called the **rule base**, and $\mathcal{A}$ is an aggregator function over $\mathcal{R}$, and an arbitrary preorder $\mathcal{P}$. Given an AFASP program P we denote its rule base as $\mathcal{R}_P$, its aggregator as $\mathcal{A}_P$, the lattice over which $\mathcal{R}_P$ is defined as $\mathcal{L}_P$ and the preorder used for the aggregator as $\mathcal{P}_P$. AFASP programs whose rule bodies are of the form $f(b_1, \ldots, b_n;)$, i.e. only contain increasing functions, are called **positive** AFASP programs. An AFASP program whose rule base contains no constraints is called **constraint-free**; a positive constraint-free program is called **simple**. The set $\mathcal{B}_P = \mathcal{B}_{\mathcal{R}_P}$ is called the **Herbrand base** of P. Furthermore, we define the set P_a, for $a \in \mathcal{B}_P$, as $P_a = \{r \mid r \in \mathcal{R}_P \wedge (r_h = a)\}$. Last, a **rule interpretation of an AFASP program** is a rule interpretation of $\mathcal{R}_P$.

In the remainder of the paper the term **program** always refers to an AFASP program. Furthermore, we will use the term **interpretation of a program** for the interpretation of the rule base of the program.

Finally, we introduce two types of **approximate models**: ρ-**rule models**, which link rule interpretations to interpretations of the aggregator expression, and k-**models**, which are interpretations that induce rule interpretations whose score is at least k.

Definition 6 Let $P = \langle \mathcal{R}_P, \mathcal{A}_P \rangle$ be an AFASP program. A **ρ-rule model**, with ρ a rule interpretation of P, is an interpretation I of $\mathcal{R}_P$ such that $\rho_I \geq \rho$. A **k-model**, $k \in \mathcal{P}_P$, of P is any interpretation I satisfying $\mathcal{A}_P(\rho_I) \geq k$. Lastly, we define the values $\min(P) = \mathcal{A}_P(\rho_\bot)$ and $\max(P) = \mathcal{A}_P(\rho_\top)$, where $\forall r \in \mathcal{R}_P \cdot \rho_\bot(r) = 0$ and $\forall r \in \mathcal{R}_P \cdot \rho_\top(r) = 1$. Intuitively these correspond to the minimal, resp. maximal value the aggregator expression can attain.

Obviously, any ρ_1-rule model M, with ρ_1 some rule interpretation, is also a ρ_2-rule model when $\rho_2 \leq \rho_1$. Similarly any k_1-model M of a program P, with $k_1 \in \mathcal{P}_P$, will also be a k_2-model when $k_2 \leq_{\mathcal{P}_P} k_1$.

Example 2 Consider program P_{fgc} and the graph depicted in Figure 1b from Section 1, together with rule interpretations ρ_1 and ρ_2 from Example 1. Furthermore for this example we define $x \leftrightarrow y = (x \rightarrow_m y) \wedge_m (y \rightarrow_m x)$, $\wedge_{constr} = \wedge_l$ and interpret the rules using the Łukasiewicz implication. To have a ρ_1-model, we need an interpretation I such that $\rho_I \geq \rho_1$. Now consider

$$\begin{aligned} I_1 \ = \{&edge(a,b)^1, edge(b,a)^1, edge(a,c)^1, edge(c,a)^1, edge(b,c)^1, edge(c,b)^1, white(a)^1, \\ &white(b)^{0.7}, black(b)^{0.3}, white(c)^{0.3}, black(c)^{0.7}, sim(a,b)^{0.7}, sim(b,a)^{0.7}, sim(a,c)^{0.3}, \\ &sim(c,a)^{0.3}, sim(b,c)^{0.3}, sim(c,b)^{0.3}, sim(a,a)^1, sim(b,b)^1, sim(c,c)^1\} \end{aligned}$$

Clearly $I_1(r) = 1$ for every rule not in $\{constr_{(x,y)} \mid x, y \in Nodes\}$, hence $I_1(r) \geq \rho_1(r)$. For the constraint rules we obtain:

$$I_1(constr_{(a,b)}) = (1 \wedge_l 0.7) \rightarrow_l 0 = 0.3 = \rho_1(constr_{(a,b)})$$

Likewise for the other constraint rules we obtain that $I_1(constr_{(x,y)}) = \rho_1(constr_{(x,y)})$ for any $x, y \in Nodes$. Hence I_1 is a ρ_1-rule model of P_{fgc}. Analogously we can verify that I_1 is not a ρ_2-rule model of P_{fgc} as $I_1(constr_{(a,b)}) = 0.3 < \rho_2(constr_{(b,c)})$. Consider now

$$\begin{aligned} I_2 \ = \{&edge(a,b)^1, edge(b,a)^1, edge(a,c)^1, edge(c,a)^1, edge(b,c)^1, edge(c,b)^1, white(a)^1, \\ &black(b)^1, white(c)^{0.5}, black(c)^{0.5}, sim(a,c)^{0.5}, sim(c,a)^{0.5}, sim(b,c)^{0.5}, sim(c,b)^{0.5}, \\ &sim(a,a)^1, sim(b,b)^1, sim(c,c)^1\} \end{aligned}$$

Again one can easily verify that $I_2(r) = 1$ for any rule not in $\{constr_{(x,y)} \mid x, y \in Nodes\}$. For the constraint rules we obtain:

$$I_2(constr_{(a,b)}) = (1 \wedge_l 0 \rightarrow_l 0) = 1 \geq \rho_2(constr_{(a,b)})$$

Likewise for every other constraint rule $constr_{(x,y)}$ with $x, y \in Nodes$ we obtain $I_2(constr_{(x,y)}) \geq \rho_2(constr_{(x,y)})$. Hence I_2 is a ρ_2-rule model of P_{fgc}.

Now consider the aggregators $\mathcal{A}$ and $\mathcal{A}'$ from Example 1. Using $\mathcal{A}$ we obtain that I_1 is an 1.92-model and I_2 is a 2-model of P_{fgc}, hence I_2 is preferred over I_1. However, with $\mathcal{A}'$ we obtain that I_1 is an $(0.7, 0.7, 0.3)$-model of P_{fgc}, whereas I_2 is an $(0.5, 0.5, 1)$-model of P_{fgc}. This means that $I_2 \leq_{lex} I_1$, and thus with this aggregator interpretation I_1 is preferred over I_2 since it satisfies the more important rules to a better degree.

3.2 Answer Sets

Rules in ASP implement the intuition of (non-deterministic) forward chaining. In practice, we are therefore interested in those models that are in accordance with this intuition. Effectively, there are two types of models we wish to exclude from our solution.

The first problem is the occurrence of atoms with a value above the one warranted by the rules. Consider the rule $r : a \leftarrow \alpha$. This rule will be fully satisfied by an interpretation I (i.e. have a value of 1) whenever $I(a) \geq I(\alpha)$. Examples are the two 1-models M and M', satisfying $M(a) = 1$, $M(\alpha) = 0.5 = M'(\alpha)$ and $M'(a) = 0.5$. However, the first model attaches a higher value to a than what the rule actually supports (viz. 0.5) and is therefore unwanted. In other words, we do not want to conclude anything more than what is needed to make the rule completely satisfied.

The second problem arises when atoms are "self-motivating", i.e. their truth value is supported by some rule, but that support is ultimately based on the value of the atom itself. An illustration of this can be seen in the following two-rule program P with $\mathcal{A}_P(\rho) = \inf\{\rho(r) \mid r \in \mathcal{R}_P\}$:

$$r_1 : a \leftarrow b$$
$$r_2 : b \leftarrow a$$

Both the models $M = \{a^1, b^1\}$ and $M' = \{a^0, b^0\}$ are free from the first problem we mentioned, but the support given to the value of b is derived from the support for the value of a, which is itself derived from the value of b. Hence, only model M' is free from knowledge not supported by the program.

The models that do not suffer from these defects will be called **answer sets**, and correspond to particular minimal models, as we will show later on. They are defined formally in the next sections. As the definition of answer sets for non-positive programs is an extension of the one for positive programs, we introduce them separately.

3.2.1 Positive Programs

To define the answer sets of positive programs we need to introduce new concepts and extend concepts from FASP to deal with the partial rule satisfaction. First we introduce the support of a rule w.r.t. some value in the lattice over which the rule ranges, which captures the idea of partial rule application. Using the support, we then extend the immediate consequence operator which allows us to derive knowledge from a program in a forward chaining manner, while still allowing for partial rule satisfaction. Then we define answer sets of positive programs and introduce some motivating propositions.

Definition 7 ([80]) Let $r : a \leftarrow \alpha$ be a rule defined over the lattice $\mathcal{L}$ and let I be an interpretation of r. The **support** of this rule w.r.t. some $c \in \mathcal{L}$ is denoted as $I_s(r, c)$ and is defined by

$$I_s(r, c) = \inf\{y \in \mathcal{L} \mid I(\alpha) \rightarrow y \geq c\}$$

It turns out that a characterization of this operator is easy to find[1].

Proposition 1 *Let $r : a \leftarrow \alpha$ be a rule defined over the lattice $\mathcal{L}$, I an interpretation of r, and c a value in $\mathcal{L}$. Then $I_s(r, c) = I(\alpha) \wedge_r c$.*

[1] All proofs can be found in Appendix A.

Example 3 Consider the rule $r : a \leftarrow_m \alpha$ with interpretations I and I' satisfying $I(\alpha) = I'(\alpha) = 0.5$, $I(a) = 1$ and $I'(a) = 0.5$. The support of this rule w.r.t. $\rho_\top(r)$ is given by $I_s(r, \rho_\top(r)) = 0.5 \wedge_m 1 = 0.5$. Likewise we can compute that $I'_s(r, \rho_\top(r)) = 0.5 \wedge_m 1 = 0.5$. Hence, $I(a) > I_s(r, \rho_\top(r))$ and $I'(a) = I_s(r, \rho_\top(r))$. This means that the interpretation of a by I' is consistent with the support provided by the rule, whereas the interpretation by I is strictly greater. Hence, rule r cannot be used to justify the value of a under interpretation I. Likewise we can see that $I_s(r, 0.9) < I(r)$, meaning I attaches a value to a that is higher than what is needed for satisfying this rule to a degree of 0.9.

For simple AFASP programs, answer set semantics are determined using a "forward chaining" approach, captured in the definition of the **immediate consequence operator**. This operator ensures that the support of a rule is propagated to its head, which means that we derive exactly the maximal amount of knowledge contained in the program.

Definition 8 Let $\mathcal{R}$ be a set of rules over a lattice $\mathcal{L}$ with ρ a corresponding rule interpretation. The immediate consequence operator $\Pi_{\mathcal{R},\rho}$ derived from $\mathcal{R}$ and ρ is a mapping from $(\mathcal{B}_\mathcal{R} \to \mathcal{L})$ to $(\mathcal{B}_\mathcal{R} \to \mathcal{L})$ defined for $I \in (\mathcal{B}_\mathcal{R} \to \mathcal{L})$ and $a \in \mathcal{B}_\mathcal{R}$ as:

$$\Pi_{\mathcal{R},\rho}(I)(a) = \sup\{I_s(r, \rho(r)) \mid r \in \mathcal{R}, (r_h = a)\}$$

For an AFASP program P, we usually write $\Pi_{P,\rho}$ for $\Pi_{\mathcal{R}_P \setminus \mathcal{C}_P,\rho}$, where $\mathcal{C}_P$ is the set of constraint rules of P.

The following example illustrates the use of this operator.

Example 4 Let P be an AFASP program with the following rule base $\mathcal{R}_P$:

$$r_1 : a \leftarrow_m 0.8$$
$$r_2 : c \leftarrow_m 0.5$$
$$r_3 : b \leftarrow_m a \wedge_m c$$
$$r_4 : b \leftarrow_m 0.2$$

and aggregator function $\mathcal{A}_P(\rho) = \inf\{\rho(r) \mid r \in \mathcal{R}_P\}$. Consider now the interpretation $\emptyset$, which attaches to each atom the value $0_\mathcal{L}$. When using the rule interpretation $\rho_\top$ we can compute $\Pi_{P,\rho_\top}(\emptyset)$ as follows (note that we are using Proposition 1 in the computation):

1. For a: $\Pi_{P,\rho_\top}(\emptyset)(a) = \sup\{\emptyset_s(r, \rho_\top(r)) \mid r \in P_a\} = \emptyset_s(r_1, \rho_\top(r_1)) = 0.8 \wedge_m 1 = 0.8$
2. For b: $\Pi_{P,\rho_\top}(\emptyset)(b) = \sup\{\emptyset_s(r, \rho_\top(r)) \mid r \in P_b\} = \sup\{\emptyset_s(r_3, \rho_\top(r_3)), \emptyset_s(r_4, \rho_\top(r_4))\}$ $= \sup\{\emptyset((a \wedge_m c) \wedge_l 1), (0.2 \wedge_m 1)\} = \sup\{0, 0.2\} = 0.2$
3. For c: $\Pi_{P,\rho_\top}(\emptyset)(c) = \sup\{\emptyset_s(r, \rho_\top(r)) \mid r \in P_c\} = 0.5 \wedge_m 1 = 0.5$

Hence $\Pi_{P,\rho_\top}(\emptyset) = \{a^{0.8}, b^{0.2}, c^{0.5}\}$. For rule interpretation $\rho = \{r_1^{0.5}, r_2^{0.3}, r_3^1, r_4^1\}$ the computation of $\Pi_{P,\rho}(\emptyset)$ is as follows:

1. For a: $\Pi_{P,\rho}(\emptyset)(a) = \emptyset_s(r_1, \rho(r_1)) = 0.8 \wedge_m 0.5 = 0.5$
2. For b: $\Pi_{P,\rho}(\emptyset)(b) = \sup\{\emptyset_s(r_3, \rho(r_3)), \emptyset_s(r_4, \rho(r_4)\} = \sup\{\emptyset((a \wedge_m c) \wedge_l 1), (0.2 \wedge_m 1)\} = \sup\{0, 0.2\} = 0.2$
3. For c: $\Pi_{P,\rho}(\emptyset)(c) = \emptyset_s(r_2, \rho(r_2)) = 0.5 \wedge_m 0.3 = 0.3$

Hence $\Pi_{P,\rho}(\emptyset) = \{a^{0.5}, b^{0.2}, c^{0.3}\}$.

This operator is similar to the one proposed by Damásio et al. in [10]. The difference is that we add the weights of the program as a parameter of the operator, where in [10] the weights of the program are fixed in the program itself. Once we have chosen a particular rule interpretation, however, the two operators are equivalent. Hence, as in [10], our operator is monotonic for simple programs:

Proposition 2 ([10]) *Let P be a positive AFASP program and ρ a rule interpretation of this program. The immediate consequence operator $\Pi_{P,\rho}$ is monotonically increasing, i.e. for every two interpretations I_1 and I_2 it holds that*

$$I_1 \leq I_2 \Rightarrow \Pi_{P,\rho}(I_1) \leq \Pi_{P,\rho}(I_2)$$

The following proposition shows that our operator is also monotonic in the rule interpretations, something that is also illustrated in Example 4.

Proposition 3 *Let P be a positive AFASP program. The immediate consequence operator is monotonically increasing in the rule interpretations, i.e. for any two rule interpretations ρ_1 and ρ_2 and interpretation I of P it holds that*

$$\rho_1 \leq \rho_2 \Rightarrow \Pi_{P,\rho_1}(I) \leq \Pi_{P,\rho_2}(I)$$

Due to a result from Tarski [78], which states that the fixpoints of any order preserving function over a complete lattice must form a complete lattice, it follows that our consequence operator has a least fixpoint $\Pi^*_{P,\rho}$ for any positive AFASP program P and rule interpretation[2] ρ. This least fixpoint can be computed using an "iterated fixpoint procedure", i.e. by applying $\Pi_{P,\rho}$ repeatedly, starting from the minimal interpretation $\emptyset$, until a fixpoint is found.

Definition 9 Let P be a positive AFASP program, and let ρ be a rule interpretation of P. Formally, we define the sequence $\mathcal{S}\langle P, \rho\rangle = \langle J_i \mid i \text{ an ordinal}\rangle$ by

$$J_i = \begin{cases} \emptyset & \text{if } i = 0 \\ \Pi_{P,\rho}(J_{i-1}) & \text{if } i \text{ is a successor ordinal} \\ \bigcup_{j<i}(J_j) & \text{if } i \text{ is a limit ordinal} \end{cases} \tag{7}$$

where $\bigcup_{i\in I}(J_i) = \sup_{i\in I} J_i$.

The least fixpoint of $\Pi_{P,\rho}$ is the first element J_i in the sequence $\mathcal{S}\langle P, \rho\rangle$ for which $\Pi_{P,\rho}(J_i) = J_i$.

Example 5 Consider program P from Example 4. If we apply $\Pi_{P,\rho_\top}$ to the interpretation $J_1 = \Pi_{P,\rho_\top}(\emptyset) = \{a^{0.8}, b^{0.2}, c^{0.5}\}$ of the sequence $\mathcal{S}\langle P, \rho_\top\rangle$ we obtain J_2:

1. For a: $J_2(a) = \Pi_{P,\rho_\top}(J_1)(a) = \sup\{(J_1)_s(r, \rho_\top(r)) \mid r \in P_a)\} = (J_1)_s(r_1, \rho_\top(r_1))$ $= 0.8 \wedge_m 1 = 0.8 = J_1(a)$
2. For b: $J_2(b) = \Pi_{P,\rho_\top}(J_1)(b) = \sup\{(J_1)_s(r, \rho_\top(r)) \mid r \in P_b\} = \sup\{(J_1)_s(r_3),$ $(J_1)_s(r_4)\} = \sup\{J_1((a \wedge_m c) \wedge_l 1), (0.2 \wedge_m 1)\} = \sup\{0.5, 0.2\} = 0.5$

[2] Note that this is only the case when the lattice is non-empty. We implicitly assume in this paper that this is always the case.

3. For c: $J_2(c) = \Pi_{P,\rho_\top}(J_1)(c) = \sup\{(J_2)_s(r, \rho_\top(r)) \mid r \in P_c\} = (J_1)_s(r_2, \rho_\top(r_2)) = 0.5 \wedge_m 1 = 0.5$

Hence $J_2 = \{a^{0.8}, b^{0.5}, c^{0.5}\}$. One can readily verify that $J_3 = \Pi_{P,\rho_\top}(J_2) = J_2$. Hence J_2 is a fixpoint of $\Pi_{P,\rho_\top}$ and, as it is the first fixpoint of the sequence $\mathcal{S}\langle P, \rho_\top\rangle$ it is the least fixpoint of $\Pi_{P,\rho_\top}$. In other words $J_2 = \Pi^*_{P,\rho_\top}$.

Example 6 Consider program P from Example 4 with the rule interpretation $\rho = \{r_1^{0.5}, r_2^{0.3}, r_3^1, r_4^1\}$. If we apply $\Pi_{P,\rho}$ to the interpretation $J_1 = \Pi_{P,\rho}(\emptyset) = \{a^{0.5}, b^{0.2}, c^{0.3}\}$ of the sequence $\mathcal{S}\langle P, \rho\rangle$ we obtain J_2:

1. For a: $J_2(a) = \Pi_{P,\rho}(J_1)(a) = \sup\{(J_1)_s(r, \rho(r)) \mid r \in P_a\} = (J_1)_s(r_1, \rho(r_1)) = 0.8 \wedge_m 0.5 = 0.5$
2. For b: $J_2(b) = \Pi_{P,\rho}(J_1)(b) = \sup\{(J_1)_s(r, \rho(r)) \mid r \in P_b\} = \sup\{(J_1)_s(r_3, \rho(r_3)), (J_1)_s(r_4, \rho(r_4))\} = \sup\{J_1((a \wedge_m c) \wedge_l 1), (0.2 \wedge_m 1)\} = \sup\{0.3, 0.2\} = 0.3$
3. For c: $J_2(c) = \Pi_{P,\rho}(J_1)(c) = \sup\{(J_1)_s(r, \rho(r)) \mid r \in P_c\} = (J_1)_s(r_2, \rho(r_2)) = 0.5 \wedge_m 0.3 = 0.3$

Again one can readily verify that $\Pi_{P,\rho}(J_2) = J_2$ and hence J_2 is the least fixpoint of $\Pi_{P,\rho}$, i.e. $J_2 = \Pi^*_{P,\rho}$. Note that $\rho \leq \rho_\top$ and $\Pi^*_{P,\rho} \leq \Pi^*_{P,\rho_\top}$, i.e. rule interpretations that put stricter requirements on the satisfaction of rules will lead to greater fixpoints.

Note that this least fixpoint may not necessarily be found in a finite number of steps, as illustrated in the following example (due to [73]).

Example 7 Consider the program *Inf*, with $\mathcal{R}_{Inf}$:

$$r : a \leftarrow \delta(a)$$

Where $\delta(x) = x + (1 - x)/2$. Obviously, δ is increasing and, moreover, $\forall x \in [0, 1] \cdot 0 < \delta(x) \leq 1$. The first steps of the computation of the least fixpoint of $\Pi_{Inf,\rho_\top}$ are shown below:

$$\begin{aligned}
J_0 &= \{a^0\} \\
J_1 &= \Pi_{P,\rho}(J_0) = \{a^{0.5}\} \\
J_2 &= \Pi_{P,\rho}(J_1) = \{a^{0.75}\} \\
J_3 &= \Pi_{P,\rho}(J_2) = \{a^{0.875}\} \\
J_4 &= \Pi_{P,\rho}(J_3) = \{a^{0.9375}\} \\
J_5 &= \Pi_{P,\rho}(J_4) = \{a^{0.96875}\} \\
J_6 &= \Pi_{P,\rho}(J_5) = \{a^{0.984375}\} \\
... &= ...
\end{aligned}$$

Clearly, $\forall i \in \mathbb{N} \cdot J_i < \{a^1\}$, but $J_\omega = \bigcup_{i\in\mathbb{N}} J_i = \{a^1\}$, which is the least fixpoint $\Pi^*_{P,\rho_\top}$.

The following proposition shows us that smaller rule interpretations yield smaller (least) fixpoints. Hence if we tighten the lower bounds imposed on the rules of a program P, the resulting knowledge that we can derive from P using forward chaining increases. This corresponds to our intuition, as in general deriveable knowledge monotonically increases with tighter constraints, e.g. from inconsistent constraints one is able to derive anything.

Proposition 4 *Let $\rho_1 \leq \rho_2$ be rule interpretations of a positive AFASP program P. Then $\Pi^*_{P,\rho_1} \leq \Pi^*_{P,\rho_2}$.*

Consider program P from Example 4 and its interpretation J_2 from Example 6. Note that for J_2 we obtain that $J_2(r_1) = 0.5 \leftarrow_m 0.8 = 0.5$, that $J_2(r_2) = 0.3 \leftarrow_m 0.5 = 0.3$, that $J_2(r_3) = 0.3 \leftarrow_m (0.5 \wedge_m 0.3) = 1$, and that $J_2(r_4) = 0.3 \leftarrow_m 0.2 = 1$, and thus, according to Definition 6, J_2 is a ρ-rule model with ρ defined as in Example 6, i.e. $\rho = \{r_1^{0.5}, r_2^{0.3}, r_3^1, r_4^1\}$. It turns out that this is a general property, i.e. that fixpoints of the immediate consequence operator are ρ-rule models. Note that this property holds for all constraint-free programs and thus in particular also for non-positive programs.

Proposition 5 *Let P be a constraint-free AFASP program, ρ a rule interpretation of P and M a fixpoint of $\Pi_{P,\rho}$. Then M is a ρ-rule model of P.*

The converse is not true in general, as one can see from the following example.

Example 8 Consider the following simple AFASP program:

$$r : a \leftarrow 0.5$$

with rule interpretation $\rho_\top$ and interpretation $I = \{a^1\}$. As $I(r) = 1 = \rho_\top(r)$, I is a $\rho_\top$-rule model of P. But $\Pi_{P,\rho_\top}(I)(a) = 0.5 \wedge_m 1 = 0.5 < I(a)$ and thus I is not a fixpoint of $\Pi_{P,\rho_\top}$.

However, the converse of Proposition 5 turns out to hold for minimal (w.r.t. Zadeh inclusion) ρ-rule models and simple programs.

Proposition 6 *Let P be a simple AFASP program, ρ a rule interpretation of P and M a minimal ρ-rule model of P. Then M is a fixpoint of $\Pi_{P,\rho}$.*

Finally, we define k-**answer sets** of a program P as those least fixpoints of the consequence operator that are k-models of P.

Definition 10 Let P be a positive AFASP program. An interpretation M is a k-**answer set** ($k \in \mathcal{P}_P$) of P iff $M = \Pi^*_{P,\rho_M}$ and $\mathcal{A}_P(\rho_M) \geq k$.

Example 9 Consider program P from Example 4 and the least fixpoints $\Pi^*_{P,\rho_\top}$ and $\Pi^*_{P,\rho}$ (with ρ as in Example 6) computed in Example 5 and Example 6 respectively. For convenience we refer to $\Pi^*_{P,\rho_\top}$ as A_1 and to $\Pi^*_{P,\rho}$ as A_2. Then A_1 is a 1-answer set of P as $\mathcal{A}_P(\rho_{A_1}) = \inf\{r_1, r_2, r_3, r_4\}(\rho_{A_1}) = 1$ and A_2 is an 0.3-answer set as $\mathcal{A}_P(\rho_{A_2}) = \inf\{r_1, r_2, r_3, r_4\}(\rho_{A_2}) = 0.3$. Note that A_1 is also an 0.3-answer set, since according to Definition 6 it is an 0.3-model, i.e. $\rho_{A_1}(\mathcal{A}_P) \geq 0.3$.

The idea behind this definition is that answer sets represent the knowledge inferrable from a program P without resorting to external knowledge, i.e. knowledge not contained in the program. This is reflected in the definition since the least fixpoint of Π_{P,ρ_I} corresponds to the result of applying forward chaining on the minimal interpretation. Furthermore, the knowledge expressed by an answer set is also maximal as it is a fixpoint of the immediate consequence operator. Hence using forward chaining on this model will not yield new knowledge.

The k-prefix allows to distinguish between **approximate** answer sets, i.e. answer sets that do not fulfill the rules of the program completely. This allows us to handle

conflicting information, or to find approximate solutions to problems encoded as fuzzy answer set programs, when the computation of perfect solutions is too costly.

Note that answer sets for positive programs, contrary to the classical case and non-aggregated FASP approaches, are not necessarily unique. This is illustrated in the following example.

Example 10 Consider a program P with the following rules:

$$r_1 : a \leftarrow_m 1$$
$$r_2 : b \leftarrow_m 1$$

and aggregator function $\mathcal{A}_P(\rho) = \inf\{\rho(r) \mid r \in \mathcal{R}_P\}$. Consider now two interpretations of P, viz. $I_1 = \{a^{0.5}, b^1\}$ and $I_2 = \{a^1, b^{0.5}\}$. It is easy to see that both of them are least fixpoints of the immediate consequence operator with their induced rule interpretations. As $\rho_{I_1}(r_1) = 0.5 = \rho_{I_2}(r_2)$ and $\rho_{I_1}(r_2) = 1 = \rho_{I_2}(r_1)$, both of them are 0.5-answer sets.

However, since there is only one rule interpretation satisfying all the rules to degree 1, the optimal answer set will always be unique. There is a strong connection between minimal ρ-rule models and the answer sets we define here, as illustrated by the following proposition.

Proposition 7 *Let P be a simple AFASP program and ρ a rule interpretation of P. Then $\Pi^*_{P,\rho}$ is the unique minimal ρ-rule model of P.*

From this proposition we can show that our answer sets correspond to minimal rule models.

Corollary 1 *Let P be a simple AFASP program. Then A is a $\mathcal{A}_P(\rho_A)$-answer set of P iff A is the unique minimal ρ_A-rule model of P.*

One may wonder whether every rule interpretation ρ for a simple AFASP program P can be used to generate an answer set $M = \Pi^*_{P,\rho}$ such that $\rho_M = \rho$. The answer is negative, as one can see from the following example:

Example 11 Consider the program P with aggregator $\mathcal{A}_P(\rho) = \inf\{\rho(r) \mid r \in \mathcal{R}_P\}$ and the following rule base $\mathcal{R}_P$:

$$r_1 : a \leftarrow_m 0.2$$
$$r_2 : a \leftarrow_m (a > 0)$$

where $(x > 0) = 0$ if $x \leq 0$ and $(x > 0) = 1$ otherwise. Computing $\Pi^*_{P,\rho}$ for $\rho = \{r_1^{0.2}, r_2^1\}$ yields $\Pi^*_{P,\rho} = \{a^1\} = M$, which induces $\rho_M = \{r_1^1, r_2^1\} \neq \rho$. One can easily verify that $M = \Pi^*_{P,\rho_M}$ and $\mathcal{A}_P(\rho_M) \geq 1$, thus M is an 1-answer set of P.

As one can see, the least fixpoint of $\Pi_{P,\rho}$ from Example 11 turned out to be a 1-answer set of P, where $1 > \mathcal{A}_P(\rho)$, since $\mathcal{A}_P(\rho) = 0.2$. The following propositions show that in general the least fixpoint of $\Pi_{P,\rho}$ for some program P and an arbitrary rule interpretation ρ of this program will always be a k-answer set, for $k \geq \mathcal{A}_P(\rho)$. This means that to obtain a k-answer set of a positive program P for an arbitrary $k \in \mathcal{P}_P$, we only need to compute $\Pi^*_{P,\rho}$ for some rule interpretation ρ satisfying $\mathcal{A}_P(\rho) \geq k$.

Proposition 8 *Let P be a simple AFASP program, ρ a rule interpretation of P, and $M = \Pi^*_{P,\rho}$. Then $\Pi^*_{P,\rho_M} = M$.*

Proposition 9 *Let P be a simple AFASP program and ρ a rule interpretation of P. Then $M = \Pi^*_{P,\rho}$ is a $\mathcal{A}_P(\rho)$-answer set.*

We obtain two immediate corollaries, the first of which shows that a model of a simple program is a k-answer set iff it is produced by some ρ-rule interpretation with $\mathcal{A}_P(\rho) \geq k$.

Corollary 2 *M is a k-answer set of a simple AFASP program P iff there is some rule interpretation ρ for which $\mathcal{A}_P(\rho) \geq k$, such that $M = \Pi^*_{P,\rho}$.*

The following corollary shows that it is easy to obtain a suitable rule interpretation for simple AFASP programs.

Corollary 3 *Every simple AFASP program P has a $max(P)$-answer set, with $max(P)$ as defined in Definition 6.*

Hence, when constructing a k-answer set of a simple program P, with $k \in \mathcal{P}_P$ and $k \leq max(P)$, we can simply use $\rho_\top$ and compute $\Pi^*_{P,\rho_\top}$. As any k_1-answer set of P is a k_2-answer set of P for $k_1 \geq k_2$, $\Pi^*_{P,\rho_\top}$ will be a k-answer set for any $k \leq max(P)$. Hence we have arrived at a practical procedure for finding answer sets of simple programs.

Example 12 Consider an AFASP program P with rule base $\mathcal{R}_P$:

$$r_1 : a \leftarrow_m 0.8$$
$$r_2 : b \leftarrow_m 0.4$$
$$r_3 : c \leftarrow a \wedge_m b$$

The aggregator is $\mathcal{A}_P(\rho) = \rho(r_1) + \rho(r_2) + \rho(r_3)$, defined over the preorder $(\mathbb{R}, \leq)$. As $\mathcal{A}_P(\rho_\top) = 3$, we know that $\Pi^*_{P,\rho_\top} = \{a^{0.8}, b^{0.4}, c^{0.4}\}$ is the 3-answer set of P.

3.2.2 General programs

Definition 10 of answer sets is only applicable to positive programs. The reason for this limitation is that the definition depends crucially on the monotonicity of the fixpoint operator. In this section, we extend the definition of answer sets to cover arbitrary programs, similar to previous FASP proposals such as [53]. One might think that the semantics of non-positive programs could again be given by minimal models, but it turns out that minimal models are unsuitable. For example, consider the following program:

$$r_1 : 0 \leftarrow\sim_l a$$
$$r_2 : a \leftarrow a$$

with aggregator $\mathcal{A}_P(\rho) = \inf\{\rho(r) \mid r \in \mathcal{R}_P\}$. The minimal 1-model is $\{a^1\}$, but the motivation for a depends on a itself and thus $\{a^1\}$ is not acceptable. The underlying reason is that constraints should not be used as support of a atom, i.e. in (F)ASP, stating that a is true is not at all the same as stating that solutions in which a is false are not allowed.

To solve this problem, we reduce the semantics of such a program P to that of a simple reduct program P^M, which is called the reduct w.r.t. a candidate answer set M, similar to [53]. Note that this generalizes the well-known Gelfond-Lifschitz transformation from [30].

Definition 11 Let I be an interpretation of an AFASP program P. The **reduct** of a rule $r : a \leftarrow f(b_1, \ldots, b_n; c_1, \ldots, c_m)$, w.r.t. I, denoted as r^I, is defined by

$$r^I : a \leftarrow f(b_1, \ldots, b_n; I(c_1), \ldots, I(c_m)) \tag{8}$$

Similarly, the reduct w.r.t. I of an AFASP program P, denoted P^I, is obtained from P by replacing all rules r in $\mathcal{R}_P$ by their reduct r^I. We will also write $\mathcal{R}^I$, with $\mathcal{R}$ a set of rules, to denote $\{r^I \mid r \in \mathcal{R}\}$.

Example 13 Consider a program P with rule base

$$r_1 : a \leftarrow \sim_l b$$
$$r_2 : b \leftarrow 0.3$$

Then the reduct P^A for the candidate answer set $A = \{a^{0.7}, b^{0.3}\}$ contains the following rule base:

$$r_1^A : a \leftarrow 0.7$$
$$r_2^A : b \leftarrow 0.3$$

Thus, P^I is obtained by replacing each atom a in which the body function of a rule is decreasing by its valuation $I(a)$. Obviously, P^I is a positive AFASP program. To see that the above reduction generalizes the traditional Gelfond-Lifschitz (GL) transformation, it suffices to note that in traditional logic programming the only way to have a negative occurrence of a proposition a in a rule body is via a negation-as-failure literal *not a*. The GL transformation then essentially replaces such literals with their values in the intended stable model interpretation, yielding a positive program.

Answer sets of these programs are then defined similarly as in the crisp case.

Definition 12 Let P be an AFASP program. An interpretation M is a k**-answer set** of P ($k \in \mathcal{P}_P$) iff $M = \Pi^*_{P^M, \rho_M}$ and $\mathcal{A}_P(\rho_M) \geq k$.

Example 14 Consider program P_{fgc} from Section 1 together with interpretations I_1 and I_2 from Example 2. It turns out that both I_1 and I_2 are approximate answer sets of this program. Indeed, for I_1 the reduct $P_{fgc}^{I_1}$ becomes

$$(gen_1)_a : white(a) \leftarrow 1$$
$$(gen_1)_b : white(b) \leftarrow 0.7$$
$$(gen_1)_c : white(c) \leftarrow 0.3$$
$$(gen_2)_a : black(a) \leftarrow 0$$
$$(gen_2)_b : black(b) \leftarrow 0.3$$
$$(gen_2)_c : black(c) \leftarrow 0.7$$
$$(sim_1)_{(a,a)} : sim(a,a) \leftarrow white(a) \leftrightarrow_m white(a)$$
$$(sim_1)_{(a,b)} : sim(a,b) \leftarrow white(a) \leftrightarrow_m white(b)$$
$$(sim_1)_{(a,c)} : sim(a,c) \leftarrow white(a) \leftrightarrow_m white(c)$$

$$(sim_1)_{(b,a)} : sim(b,a) \leftarrow white(b) \leftrightarrow_m white(a)$$
$$(sim_1)_{(b,b)} : sim(b,b) \leftarrow white(b) \leftrightarrow_m white(b)$$
$$(sim_1)_{(b,c)} : sim(b,c) \leftarrow white(b) \leftrightarrow_m white(c)$$
$$(sim_1)_{(c,a)} : sim(c,a) \leftarrow white(c) \leftrightarrow_m white(a)$$
$$(sim_1)_{(c,b)} : sim(c,b) \leftarrow white(c) \leftrightarrow_m white(b)$$
$$(sim_1)_{(c,c)} : sim(c,c) \leftarrow white(c) \leftrightarrow_m white(c)$$
$$(sim_2)_{(a,a)} : sim(a,a) \leftarrow black(a) \leftrightarrow_m black(a)$$
$$(sim_2)_{(a,b)} : sim(a,b) \leftarrow black(a) \leftrightarrow_m black(b)$$
$$(sim_2)_{(a,c)} : sim(a,c) \leftarrow black(a) \leftrightarrow_m black(c)$$
$$(sim_2)_{(b,a)} : sim(b,a) \leftarrow black(b) \leftrightarrow_m black(a)$$
$$(sim_2)_{(b,b)} : sim(b,b) \leftarrow black(b) \leftrightarrow_m black(b)$$
$$(sim_2)_{(b,c)} : sim(b,c) \leftarrow black(b) \leftrightarrow_m black(c)$$
$$(sim_2)_{(c,a)} : sim(c,a) \leftarrow black(c) \leftrightarrow_m black(a)$$
$$(sim_2)_{(c,b)} : sim(c,b) \leftarrow black(c) \leftrightarrow_m black(b)$$
$$(sim_2)_{(c,c)} : sim(c,c) \leftarrow black(c) \leftrightarrow_m black(c)$$
$$constr_{(a,a)} : 0 \leftarrow edge(a,a) \wedge_m sim(a,a)$$
$$constr_{(a,b)} : 0 \leftarrow edge(a,b) \wedge_m sim(a,b)$$
$$constr_{(a,c)} : 0 \leftarrow edge(a,c) \wedge_m sim(a,c)$$
$$constr_{(b,a)} : 0 \leftarrow edge(b,a) \wedge_m sim(b,a)$$
$$constr_{(b,b)} : 0 \leftarrow edge(b,b) \wedge_m sim(b,b)$$
$$constr_{(b,c)} : 0 \leftarrow edge(b,c) \wedge_m sim(b,c)$$
$$constr_{(c,a)} : 0 \leftarrow edge(c,a) \wedge_m sim(c,a)$$
$$constr_{(c,b)} : 0 \leftarrow edge(c,b) \wedge_m sim(c,b)$$
$$constr_{(c,c)} : 0 \leftarrow edge(c,c) \wedge_m sim(c,c)$$

One can easily verify that $\Pi^*_{P^{I_1}_{fgc},\rho_{I_1}} = I_1$. Hence, combining this with the observation that I_1 is an 1.92-model of P_{fgc} using aggregator $\mathcal{A}$ from Example 2, we obtain that I_1 is a 1.92-answer set of P. With aggregator $\mathcal{A}'$ from Example 2 it would be a $(0.7, 0.7, 0.3)$-answer set. Likewise we obtain that $I_2 = \Pi^*_{P^{I_2}_{fgc},\rho_{I_2}}$, which means I_2 is a 2-answer set with aggregator $\mathcal{A}$ and a $(0.5, 0.5, 1)$-answer set with aggregator $\mathcal{A}'$.

Intuitively, answer sets need to be *self-producing*, i.e. by assuming the knowledge in the answer set and starting from the empty interpretation of a program, we should only be able to infer the same set. Any such set is guaranteed to contain a maximal amount of knowledge as it is a fixpoint of the immediate consequence operator and does not contain external knowledge due to the fixpoint computation starting from the empty interpretation and assumptions that originate from this same set. Note that Definition 12 supports programs with constraints, as these only influence the k-score obtained in the aggregator, and not the fixpoint computation of the reduct program;

therefore constraints can only restrict the results and cannot add atoms to solutions. A first proposition ensures that answer sets are models of a program, as we would expect.

Proposition 10 *Let M be a k-answer set of an AFASP program P. Then M is a k-model of P.*

A second proposition ensures that answer sets are minimal rule models of a program.

Proposition 11 *Let M be a k-answer set of an AFASP program P. Then M is a minimal ρ_M-model of P.*

As is the case for classical answer set programming, the reverse of this proposition does not hold:

Example 15 Consider the $\rho_\top$-rule model $M = \{a^0, b^1\}$ of the following program P:

$$r : a \leftarrow_m \sim_l b$$

with aggregator $\mathcal{A}_P(\rho) = \inf\{r \mid r \in P\}$. As any $M' \subset M$ must satisfy $M'(a) = 0$ and $M'(b) < 1$, we obtain that $M'(r) = M'(a) \leftarrow (1 - M'(b)) = 0 \leftarrow_m (1 - M'(b))$. Since $0 \leftarrow_m (1 - M'(b)) \geq 1$ only if $1 - M'(b) = 0$, or $M'(b) = 1$, we obtain $M'(r) < 1$. This means that M is a minimal $\rho_\top$-rule model of P. M is not a 1-answer set of P, however, as $\Pi^*_{P^M,\rho_\top} = \emptyset \neq M$.

It turns out that for answer sets of AFASP programs without constraints, there is a correspondence with minimal fixpoints of the non-monotonic function Π_{P,ρ_M}.

Proposition 12 *Let P be a constraint-free AFASP program with a k-answer set M. Then M is a minimal fixpoint of Π_{P,ρ_M}.*

The converse of Proposition 12 does not hold in general, however, as witnessed by the following example.

Example 16 Consider the program P with rule base over $([0,1], \leq)$:

$$r_1 : a \leftarrow a$$
$$r_2 : p \leftarrow (\sim_l p > 0) \wedge_m (\sim_l a > 0)$$

and the following aggregator over $([0,1], \leq)$: $\mathcal{A}_P = r_1 \wedge_m r_2$. It is easy to verify that $M = \{a^1, p^0\}$, with $\rho_M = \rho_\top$, is the only, and thus minimal, fixpoint of $\Pi_{P,\rho_\top}$. However, $\mathcal{R}_{P^M}$ is

$$r_1^M : a \leftarrow a$$
$$r_2^M : p \leftarrow (1 \wedge_m (0 > 0))$$

For P^M it thus holds that $\Pi^*_{P^M,\rho_\top} = \{a^0, p^0\} \neq M$. From this it follows that the minimal fixpoint of $\Pi_{P,\rho_\top}$ is not an answer set of P.

Note that the above example corresponds to our intuition about answer sets: the minimal interpretation M contains an atom a that is self-motivating and therefore unwanted. Hence, not every minimal fixpoint is intuitively suitable as an answer set.

Finally, we would like to point out that constraints can be simulated using non-monotonic functions, meaning the presented results are generally applicable. We show this simulation in an upcoming paper as it is quite lengthy and does not fit the scope of this paper.

4 Illustrative Example

In this section we illustrate how the features of the AFASP framework are useful for building real-life applications. The example we use is that of a "paper distribution" problem, a problem that has attracted quite some attention from the research community (see e.g. [29]). Specifically, we assume that there is a set of *papers* (named *Papers*) about a certain set of *topics* (named *Topics*) that need to be assigned to *reviewers* (the set of all reviewers is *Reviewers*) with a certain *expertise* on the aforementioned topics. When assigning these papers, care must be taken to ensure that there are no *conflicts* between reviewers and authors; furthermore, each paper should have enough reviewers and no reviewer should be burdened with a high review workload. We assume that the expertise of reviewers, the topics of papers and the affiliations of both authors and reviewers are known and thus need not be calculated with an AFASP program, but are given by a set of fact rules $\mathcal{F}$. For example to denote that reviewer r_1 is an expert of degree 0.4 on topic t_3 we add the fact $expert(r_1, t_3) \leftarrow 0.4$. The rule base $\mathcal{R}_{P_{paper}}$ of the program P_{paper} solving this problem is defined over the lattice $([0,1], \leq)$ and consists of the set $\mathcal{F}$ together with the following rules:

$$
\begin{aligned}
\mathit{confl}: \quad & \mathit{conflict}(R,P) \leftarrow_m \mathit{author}(R,P) \vee_m (\mathit{author}(R',P) \\
& \qquad \wedge_m \mathit{university}(R,U) \\
& \qquad \wedge_m \mathit{university}(R',U') \wedge_m \ \mathit{close}(U,U')) \\
\mathit{appr}: \quad & \mathit{appropriate}(R,P) \leftarrow_m (1 - \mathit{conflict}(R,P)) \\
& \qquad \wedge_m (\mathit{about}(P,T) \wedge_l \mathit{expert}(R,T)) \\
\mathit{inappr}: \quad & \mathit{inappropriate}(R,P) \leftarrow_l \ (1 - \mathit{appropriate}(R,P)) \\
\mathit{qualr}: \quad & \mathit{overworked}(R) \leftarrow_l \ f_1(\sum_{p \in Papers} \mathit{assign}(R,p)) \\
\mathit{enough}: \quad & \mathit{enough}(P) \leftarrow_m f_2(\sum_{r \in Reviewers} \mathit{assign}(r,P)) \\
\mathit{qualp}: \quad & 0 \leftarrow_l \ 1 - \mathit{enough}(P) \\
\mathit{assgn}: \quad & \mathit{assign}(R,P) \leftarrow_m ((1 - \mathit{inappropriate}(R,P)) \\
& \qquad \wedge_m (1 - \mathit{overworked}(R)) \geq 1)
\end{aligned}
$$

Where f_1 is:

$$
f_1(x) = \begin{cases} 0 & \text{if } x \leq 3 \\ \dfrac{x-3}{7} & \text{if } x \in [4,9] \\ 1 & \text{if } x \geq 10 \end{cases}
$$

and f_2 is:

$$
f_2(x) = \begin{cases} \dfrac{x}{4} & \text{if } x \leq 3 \\ 1 & \text{otherwise} \end{cases}
$$

Note that, due to the process of **grounding** (see e.g. [4]), a rule as *inappr* actually denotes the set of rules $\{\mathit{inappr}_{r,p} : \mathit{inappropriate}(r,p) \leftarrow_l (1 - \mathit{appropriate}(r,p)) \mid r \in \mathit{Reviewers}, p \in \mathit{Papers}\}$.

The motivation for the rules is as follows. The *confl* rules determine when there are potential conflicts with reviewer neutrality, i.e. $\mathit{conflict}(R,P)$ quantifies the degree of conflict that diminishes the suitability of person R reviewing paper P. To keep the discussion simple, we opted to only consider the degree to which universities are close (by which we mean both geographical proximity as affiliations between universities) to

determine these conflicts. The *appr* rules determine the degree to which an assignment is appropriate, based on the knowledge of the reviewer ($expert(R, T)$), the topic of the paper ($about(P, T)$) and potential conflicts. Note that, due to the grounding process, for a certain reviewer r and paper p there will be a set of rules determining the value of $appropriate(r, p)$, namely: $\{appr_{r,p,t} : appropriate(r, p) \leftarrow_m (1 - conflict(r, p)) \wedge_m (about(p, t) \wedge_l expert(r, t)) \mid t \in Topics\}$. Hence, since any answer set A is a fixpoint of Π_{P,ρ_A} and by Definition 8 (definition of $\Pi_{P,\rho}$ for some rule interpretation ρ), this means that in any answer set A it must hold that

$$A(appropriate(r, p)) = \sup\{A_s(appr, \rho_A(appr)) \mid appr \in P_{appropriate(r,p)}\}$$

We use the Łukasiewicz t-norm for combining the reviewer knowledge and paper topic to ensure that reviewers have enough knowledge about the paper content. The *inappr* rules determine the inappropriateness of a given paper assignment. The *qualr* rules determine when a reviewer is overworked, while the combination of the *enough* rules and *qualp* constraints are used to score an answer set based on the number of reviews papers have. Lastly, the *assgn* rules assign papers to reviewers based on the suitability of the match between reviewer and paper and bearing in mind the amount of work of reviewers.

Note that the *inappr*, *qualr*, and *qualp* rules are all evaluated with a Łukasiewicz implicator, while the other rules are evaluated with the Gödel implicator. The reason for this is that the former rules are allowed to be partially fulfilled and we want their fulfillment to change gradually, while the latter rules should be completely fulfilled, hence any residual implicator can be used for their evaluation. We specifically write the *inappr* rule to ensure that we can consider a lower inappropriateness score than the (fixed) appropriateness score when this leads to better reviewer assignments. For example, if a certain paper has a low number of reviewers, we can opt to also include a reviewer that is less familiar with the topics of the paper in this way. Furthermore, note that there is a strong interaction between the *assgn*, *inappr*, and *qualr* rules: if no reviewer is assigned a paper, the *assgn* rule is triggered for that reviewer and a paper is correspondingly assigned to him; this in turn leads to an increasing *overworked* score, leading to either less further assignments or a violation of the overworked constraint if this is needed to ensure that each paper has enough reviews for example.

The aggregator expression of P_{paper} then is

$$\mathcal{A}_{P_{paper}}(\rho) = E_1(\rho) \cdot E_2(\rho) \cdot (E_3(\rho) + 10 \cdot E_4(\rho) + 20 \cdot E_5(\rho)) \tag{9}$$

where

$$\begin{aligned}
E_1(\rho) &= \min\{\rho(f) \mid f \in \mathcal{F}\} \geq 1 \\
E_2(\rho) &= \min\{\rho(confl_{r,p}) \wedge_m \rho(appr_{r,p}) \wedge_m \rho(enough_p) \wedge_m \rho(assgn_{r,p}) \\
&\qquad \mid r \in Reviewers, p \in Papers\} \geq 1 \\
E_3(\rho) &= \sum\{\rho(inappr_{r,p}) \mid r \in Reviewers, p \in Papers\} \\
E_4(\rho) &= \sum\{\rho(qualp_p) \mid p \in Papers\} \\
E_5(\rho) &= \sum\{\rho(qualr_r) \mid r \in Reviewers\}
\end{aligned}$$

The preorder for the aggregator is $(\mathbb{R}, \leq)$. The aggregation expression ensures that the *confl*, *appr*, *enough*, and *assgn* rules are completely fulfilled and allows partial

fulfillment of the *inappr*, *qualr*, and *qualp* rules. The weights in the expression state that solutions in which reviewers are not overworked are better than solutions in which some reviewer is assigned a paper about a topic the reviewer is not familiar with or where papers do not have a lot of reviews. This can be seen from the fact that the *qualr* rules are satisfied to a lower degree when we underestimate $overworked(R)$, i.e. when we attach a lower value to $overworked(R)$ than the one warranted, when this allows us to find a solution where papers have an appropriate amount of reviewers. Thus the *qualr* rules are less fulfilled when reviewers are actually more overworked, meaning that giving the highest weight to these rules in the aggregator makes it more important to estimate the $overworked(R)$ values correctly.

As an example, suppose $Papers = \{p_1, \ldots, p_{10}\}$, $Authors = \{a_1, \ldots, a_{10}\}$ such that $author(a_i, p_i) = 1$ for any $i \in 1 \ldots 10$ and $Universities = \{u_1, \ldots, u_{10}\}$ such that $university(a_i, u_i) = 1$ for any $i \in 1 \ldots 10$. Furthermore, suppose $Reviewers = \{r_1, \ldots, r_5\}$, $Universities' = \{u'_1, \ldots, u'_5\}$ such that $university(r_i, u'_i) = 1$ for any $i \in 1 \ldots 5$. Tables 3, 4, and 5 (to be found at the end of the paper) then respectively give the *about*, *expert* and *close* scores. Note that for any answer set of this program, the value of these atoms must be as given in the aforementioned tables as these are added to the program as fact rules that must be completely satisfied, i.e. things that must be true in any answer set. In Tables 6 and 7 the corresponding conflict and appropriateness scores are shown. Note that, as the $confl$ and *appr* scores directly depend on the atoms given as facts and since the rules defining these atoms always need to be completely satisfied, these atoms must also have the same score in any answer set of P_{paper}. The difference between answer sets will be in the *overworked*, *inappropriate*, *enough* and *assign* scores.

Now, consider an approximate answer set A_1 with assignments as given in Table 8. The corresponding *inappropriate* scores can be found in Table 9. One can check that A_1 indeed is an answer set by checking whether $\Pi^*_{P^{A_1}, \rho_{A_1}} = A_1$. For example, for $inappropriate(r_1, p_1)$ we can see that there is only one rule with $inappropriate(r_1, p_1)$ in its head, viz. $inappr_{r_1,p_1}$. Now the reduct of this rule is

$$inappr^{A_1}_{r_1,p_1} \colon inappropriate(r_1, p_1) \leftarrow_l A_1(1 - appropriate(r_1, p_1))$$

which, by Table 7, is equivalent to

$$inappr^{A_1}_{r_1,p_1} \colon inappropriate(r_1, p_1) \leftarrow_l 0.5 \tag{10}$$

We can also compute $\rho_{A_1}(inappr_{r_1,p_1})$ as

$$\rho_{A_1}(inappr_{r_1,p_1}) = A_1(inappropriate(r_1, p_1) \leftarrow_l 1 - appropriate(r_1, p_1))$$

which, by Table 7 and Table 9, is equal to

$$\rho_{A_1}(inappr_{r_1,p_1}) = 0 \leftarrow_l 0.5 = 0.5 \tag{11}$$

Now using (10) and (11) we know that

$$\begin{aligned} \Pi^*_{P^{A_1}, \rho_{A_1}}(inappropriate(r_1, p_1)) &= A_1(0.5) \wedge_l \rho_{A_1}(inappr_{r_1,p_1}) \\ &= 0.5 \wedge_l 0.5 \\ &= 0 \\ &= A_1(inappropriate(r_1, p_1)) \end{aligned}$$

One can check that for all other atoms $l \in \mathcal{B}_{P_{paper}}$ we also obtain that $\Pi^*_{P^{A_1},\rho_{A_1}}(l) = A_1(l)$ and thus that A_1 is an answer set of P_{paper}.

Now, from the *inappropriate* scores in Table 9, we for example already know that $\rho_{A_1}(inappr_{r_1,p_1}) = 0.5$ by (11). Likewise we can see from Table 7 and Table 9 that

$$\begin{aligned} A_1(inappr_{r_1,p_4}) &= A_1(inappropriate(r_1,p_4) \leftarrow_l (1 - A_1(appropriate(r_1,p_4))) \\ &= 0.8 \leftarrow_l 0.8 = 1 \end{aligned}$$

The values of $inappr_{r,p}$ for any $r \in Reviewers$ and $p \in Papers$ follow in the same fashion and thus we can compute

$$E_3(\rho_{A_1}) = \sum\{A_1(inappr_{r,p}) \mid r \in Reviewers, p \in Papers\} = 385/10$$

As the *enough* rules should always be completely satisfied, we have that $enough(p)$ for a certain paper p always should be equal to $f_2(\sum\{assign(r,p) \mid r \in Reviewers\})$. Due to each paper having three reviewers in A_1 we then know that $A_1(enough(p))$ for any paper p will be equal to $f_2(3) = 3/4$. This means that for any $qualp_p$ rule we have that $A_1(qualp_p) = 3/4$. As there are 10 papers in total we can thus compute:

$$\begin{aligned} E_4(\rho_{A_1}) &= \sum\{A_1(qualp_p) \mid p \in Papers\} \\ &= \sum\{(3/4) \mid p \in Papers\} \\ &= 10 \cdot (3/4) \\ &= 15/2 \end{aligned}$$

Now, in this answer set we have that, even though reviewer r_1 has 6 reviews, $A_1($ $overworked(r)) = 0$ for each assigned reviewer r; thus for some reviewers we underestimate their degree of being overworked to ensure that papers have enough reviews. From the foregoing, we can compute

$$\begin{aligned} E_5(\rho_{A_1}) &= \sum\{A_1(qualr_r) \mid r \in Reviewers\} \\ &= \sum\{(f_1(\sum\{A_1(assign(r,p)) \mid p \in Papers\})) \rightarrow_l 0 \mid r \in Reviewers\} \\ &= \sum\{1 - (f_1(\sum\{A_1(assign(r,p)) \mid p \in Papers\})) \mid r \in Reviewers\} \\ &= 19/7 \end{aligned}$$

Hence we obtain that

$$\begin{aligned} \mathcal{A}_{P_{paper}}(\rho_{A_1}) &= E_1(\rho_{A_1}) \cdot E_2(\rho_{A_1}) \cdot (E_3(\rho_{A_1}) + 10 \cdot E_4(\rho_{A_1}) + 20 \cdot E_5(\rho_{A_1})) \\ &= 2349/14 \\ &\approx 167.8 \end{aligned}$$

There is room for improvement, however, as this answer set clearly contains a high work burden for some reviewers, while creating a minimal workload for others. Due to the nature of our aggregator expression, we can spread the papers among reviewers, potentially giving papers to reviewers with a lower knowledge of the domain, to obtain a better answer set. Answer set A_2, for which the assignments are given in Table 8, relieves the burden of reviewers r_3 and r_5 by assigning more reviews to reviewer r_4. Computing the value of the aggregator expression, we first obtain

from the *inappropriate* scores in Table 10 that $E_3(\rho_{A_2}) = \sum\{A_2(inappr_{r,p}) \mid r \in Reviewers, p \in Papers\} = 371/10$; furthermore we can compute in a similar fashion as for A_1 that $E_4(\rho_{A_2}) = \sum\{A_2(qualp_p) \mid p \in Papers\} = 15/2$ as every paper again has three reviewers assigned. Now, by calculating the assignments per reviewer we obtain $E_5(\rho_{A_2}) = \sum\{A_2(qualr_r) \mid r \in Reviewers\} = 20/7$. The foregoing shows $\mathcal{A}_{P_{paper}}(\rho_{A_2}) = 11847/70 \approx 169.2$, hence answer set A_2 is more suitable than A_1, as we expected.

The reader might wonder how the connectives and the input degrees in the reviewer assignment program can be determined. For the connectives an interesting approach is to first write a program that captures the overall solution and then use training data and e.g. a hill-climbing algorithm to fine-tune this program. This is similar to the techniques that are applied in rule-based systems for fuzzy control. To determine the input degrees automatic methods can be used. For example, in the field of information retrieval there are approaches that determine the expert degree of a person by means of language models (see e.g. [3]). The inputs of the reviewer assigment program can thus be seen as the outputs of such analyses.

5 Relationship to Existing Approaches

The combination of answer set programming and logic programming with uncertainty theories has received a great deal of attention over the past years. Among others, there have been extensions of logic programming using probabilistic reasoning [12, 28, 50, 51, 61, 62, 74], possibilistic reasoning [2, 63, 64], fuzzy reasoning [8, 33, 52–56, 67, 74, 80–82], and more general many-valued or uncertainty reasoning [10, 11, 13–15, 24, 26, 38–41, 43–45, 47, 48, 60, 70, 72, 73, 75, 76]. Roughly, one can divide [73] these approaches in two classes, viz. *implication-based* (IB) and *annotation-based* (AB) frameworks. In the implication-based setting a rule is generally of the form

$$a \overset{w}{\leftarrow} \alpha$$

where a is an atom, α is a body expression, and $w \in \mathcal{L}$, with $\mathcal{L}$ the lattice used for truth values. Intuitively, such a rule denotes that in any model of the program the truth degree of the implication $\alpha \rightarrow a$ must be greater than or equal to the weight w. In the annotation-based approaches one considers *annotations*, which are either constants from the truth lattice $\mathcal{L}$, variables ranging over this truth lattice, or functions over elements of this truth lattice applied to annotations. A rule is then of the form

$$a : \mu \leftarrow b_1 : \mu_1, \ldots, b_n : \mu_n$$

where $a, a_1, \ldots, a_n$ are atoms and $\mu, \mu_1, \ldots, \mu_n$ are annotations. Intuitively, an annotated rule denotes that if the certainty of each b_i, $1 \leq i \leq n$, is at least μ_i, then the certainty of a is at least μ. The links between these two approaches are well-studied in e.g. [15, 39, 44, 45] and we will therefore not repeat these results. In this section, we give an overview of these related approaches and study the links between our framework and related proposals.

5.1 Fuzzy and Many-Valued Logic Programming Without Partial Rule Satisfaction

Many proposals for fuzzy and many-valued logic programming with rules that have to be completely fulfilled have been published. In this category one finds most annotation-based approaches, e.g. [8,38,39,67,73,76] and some implication-based approaches where the weight of each rule is 1, e.g. [13–15, 41]. Some of these proposals only contain monotonic functions in rules (e.g. the AB approach from [8,39] and the IB approaches from [14, 15, 41]), while others feature negation (e.g. the AB approaches from [38, 67, 73, 76]) or even arbitrary decreasing functions (e.g. the IB approach from [13, 72]). These proposals differ from ours as they do not incorporate the idea of partial rule satisfaction.

We can readily embed the IB approaches in our framework by supplying these programs with the infimum aggregator. When modelled like this, the 1-answer sets of the embedding will correspond exactly to the answer sets of programs from the aforementioned frameworks. Thanks to this embedding we also inherit the modelling power that is already present in some of these proposals. For example, from the embeddings shown in [12, 15], and using the fact that we can embed [13] in our approach, we inherit the capacity to model Generalized Annotated Logic Programs [39], Probabilistic Deductive Databases [42], Possibilistic Logic Programming [21], Hybrid Probabilistic Logic Programs [17][3] and Fuzzy Logic Programming [81].

The AB approach from [8] is interesting in that annotations are actually fuzzy sets, which allows for an intuitive modelling language. Whether the semantics of this specific framework can be truthfully embedded in our approach is not immediately clear, but as the family of all fuzzy sets in a given universe forms a complete lattice, when equipped with Zadeh intersection and union, the use of fuzzy sets, together with functions over fuzzy sets is certainly possible in the AFASP language.

5.2 Weighted Rule Satisfaction Approaches

Some IB approaches to fuzzy and many-valued logic programming feature partial rule fulfillment by adding "weights" to rules (e.g. [10, 11, 24, 44, 45, 47, 48, 52, 53, 55–58, 70]). These weights are specified manually and they reflect the minimum degree of fulfillment required for a rule. Formally, such a rule takes on the form of

$$a \overset{w}{\leftarrow} \alpha$$

where a is an atom, α is a body expression, $\leftarrow$ is a residual implicator over $[0, 1]$ and w is a value of $[0, 1]$. We will use r_h and r_b to refer to the head, resp. the body of a rule r as usual, and r_w to refer to the weight w. In the case of [24, 52, 53, 55, 56, 70], the bodies of rules are restricted to combinations of triangular norms, possibly with negation-as-failure literals in [52, 53, 55–58], whereas in [10, 11, 44, 45, 47, 48] the bodies consist of monotonically increasing functions, where some approaches do not feature non-monotonic negation [10, 11, 44, 45] and others feature negation under the well-founded semantics [47, 48]. Furthermore, [10] allows a combination of multiple lattices to be used for rules. This last feature is obtained in the AFASP setting by using the

[3] Note that the translation process in [12] is exponential in the size of the program, but, as the authors point out, this is to be expected as reasoning in these programs in most cases is exponential.

cartesian product of all these lattices as the lattice for program rules and using the corresponding projections to extend the operators used to this product lattice.

The semantics of a program consisting of weighted rules without negation-as-failure is defined in two ways in the literature. We will take [53] and [10] as examples of these two methods, but the following discussion equally applies to all the approaches mentioned earlier, barring some minor syntactical issues. In the case of [53], an interpretation M is called a **model** of a program P when for all r in P it holds that $M(r_h) \geq M(r_b \wedge_r r_w)$. **Answer sets** of these programs are then defined as minimal models. In [10], answer sets are defined as the least fixpoints of an **immediate consequence operator**, defined for a program P, interpretation I of P and atom $l \in \mathcal{B}_P$, as:

$$\Pi_P(I)(l) = \sup\{I(r_b \wedge_r r_w) \mid r \in P_l\}$$

It is known that these two semantics coincide, which can also be shown using the results on AFASP, as demonstrated below.

Note that due to Proposition 1, it holds that $\Pi_P = \Pi_{P,\rho_w}$, where $\rho_w(r) = r_w$. Hence, for simple AFASP programs, the semantics of [10] can be obtained by taking the least fixpoint w.r.t. the rule interpretation corresponding to the rule weights. Furthermore, from Proposition 7, we know that the least fixpoint of Π_{P,ρ_w} corresponds to the minimal ρ_w-rule model of P. An interpretation M is a ρ_w-rule model of P iff for each rule $r \in P$ it holds that $M(r_h \leftarrow r_b) \geq \rho_w(r)$ and hence, by the residuation principle, that $M(r_h) \geq M(r_b) \wedge_r \rho_w(r)$. This means that ρ_w-rule models correspond to models in the sense of [53]. Hence the semantics of [53] and [10] coincide and can both be generated from a simple AFASP program.

Furthermore, due to the equivalence between Π_P and Π_{P,ρ_w}, we can use the termination conditions from [10] to determine structural conditions that ensure that the computation of the least fixpoint of $\Pi_{P,\rho}$ ends.

Note that, different to AFASP, programs with weight rules without negation-as-failure only have a single answer set corresponding to the minimal model satisfying the rules to the stated weights. This means that the weight of a rule should not simply be seen as the minimal degree of satisfaction, but that the semantics of these programs correspond to a cautious use of the rules and their weights.

At first, one might think that these semantics can easily be embedded in the AFASP framework by moving the weights into the aggregator expression and using them as lower bounds on the satisfaction of their corresponding rules. Formally, this means that we define for a program P in the sense of [10, 53] and rule base $\{r_1, \ldots, r_n\}$ with corresponding weights $w_1, \ldots, w_n$, the program P' with rule base $\mathcal{R}_{P'} = \{r_i : r_h \leftarrow r_b \mid i \in 1 \ldots n\}$ and aggregator expression $\mathcal{A}_{P'}(\rho) = (\rho(r_1) \geq w_1) \wedge \ldots \wedge (\rho(r_n) \geq w_n)$. The semantics of programs P and P' do not coincide, however, as shown in the following example.

Example 17 Let P be a program in the sense of [10, 53] with rule base $\mathcal{R}_P = \{r : a \overset{0.5}{\leftarrow}_m 1\}$. It is easy to see that the unique answer set of this program is $\{a^{.5}\}$. The corresponding AFASP program P' with rule base $\mathcal{R}_{P'} = \{r : a \leftarrow_m 1\}$ and aggregator expression $\mathcal{A}_{P'}(\rho) = \rho(r) \geq 0.5$, however, has multiple k-answer sets for varying $k \in [0, 1]$, such as the 1-answer set $\{a^1\}$. The original answer set of P is only an 0.5-answer set of P'.

A proper embedding of these semantics in the AFASP framework can be obtained as follows. Suppose P is a program with weighted rules, then the corresponding AFASP

program P_{weight} is defined as

$$P_{weight} = \{r_h \leftarrow r_b \wedge_r r_w \mid r \in P\}$$

The lattice to be used for evaluating the rules is then $([0,1], \leq)$, the aggregator lattice is $(\mathbb{B}, \leq)$. The aggregator expression is given as

$$\mathcal{A}_{P_{weight}}(\rho) = \forall r \in P_{weight} \cdot (\rho(r) = 1)$$

Example 18 Consider program P from Example 17. Using the proper embedding discussed above we obtain an AFASP program P_{weight} with rule base $\mathcal{R}_{P_{weight}} = \{r : a \leftarrow_m (1 \wedge_m 0.5)\}$ and aggregator expression $\mathcal{A}_{P_{weight}} = (\rho(r) = 1)$. The 1-answer set of P_{weight} is now $\{a^{0.5}\}$, which corresponds to the answer set of P.

The following proposition shows that for simple AFASP programs, this is indeed a truthful embedding of these semantics:

Proposition 13 *Let P be a simple AFASP program in the sense of [10]. Then A is a 1-answer set of P_{weight} iff $A = \Pi_P^*$.*

For approaches featuring negation-as-failure under the answer set semantics in the body of rules, we will again take [53] as a representative example. Although the approaches in [55, 56] are slightly different in that the reduct operation moves the value of the negated literals in the weight of the rule instead of directly substituting it in the rule body, the net result is the same. Negation-as-failure in this approach is denoted as $not_\sim\ l$, where l is an atom and $\sim$ a negator over $[0,1]$. If P is a program with negation-as-failure in the sense of [53], then an interpretation A is called an **answer set** of P if A is the answer set of P^A, where P^A is a generalized Gelfond-Lifschitz transformation replacing all negation-as-failure literals $not_\sim\ l$ by the value $A(\sim l)$. It is easy to see that this Gelfond-Lifschitz transformation is a special case of the reduct (Definition 11) we introduced for AFASP programs. From this, it easily follows that the embedding mentioned before still works in the presence of negation-as-failure. Hence we obtain the following proposition.

Proposition 14 *Let P be a program in the sense of [53]. Then A is a 1-answer set of P_{weight} iff A is an answer set of P in the sense of [53].*

Although these proposals feature partial rule satisfaction and are therefore better equipped to model real-life phenomena than previous proposals, the use of weights introduces the new problem of "weight-guessing". The AFASP framework eliminates the weight-guessing problem by using an aggregator expression, which encodes which combinations of partially satisfied rules are more desirable than others. In this light, one could think of AFASP programs as programs with *variables* as rule-weights instead of fixed weights, where the variables must be chosen according to the aggregator expression. In effect, due to Corollary 2 and the fact that rule interpretations are analogous to weights on rules, this means that semantically, a single AFASP program corresponds to a set of programs with weights. This shows that the aggregator expression has a substantial modeling advantage over attaching arbitrary weights.

Interestingly, in [57, 58], a measure is introduced that shows how the weights of the rules of a program without stable models should be increased or decreased to obtain a new program with stable models. This is related our approach, where rule weights

are variable and can be changed to obtain approximate answer sets, if none exists. Our approach differs in the fact that we do not have predefined weights on rules and assume that the best solution is the solution satisfying all the rules best. Furthermore, due to the aggregator, we can compare approximate answer sets in our framework with a preference order that is chosen by the programmer.

5.3 Fuzzy Answer Set Programming

In [80], **Fuzzy Answer Set Programming** is introduced. Our presented approach generalizes this framework by allowing a much richer vocabulary of expressions to use in rules and by allowing more sophisticated aggregator expressions. Specifically, we allow arbitrary monotonic functions in rule bodies, whereas [80] only allows t-norms in bodies. Furthermore, we base our semantics on fixpoints, which more clearly shows the link between other FASP approaches, and also fixes a problem with the semantics of [80] when generalizing to arbitrary lattices as truth values, as demonstrated below. Moreover, it is not clear how the semantics of [80] can be extended to deal with arbitrary monotonic functions in rule bodies, where this is straightforward in our fixpoint approach.

Semantically, in [80], an answer set M has a degree k, which, as in the present approach, reflects the value of an *aggregator* function that combines the degree of satisfaction of the rules in the program. However, as opposed to the present approach, this aggregator must have a value in the same lattice as the one used for the rules. As shown throughout the examples in this paper, our more general aggregator can be advantageous for modelling certain real-life problems. Furthermore, an answer set is defined in [80] as a model that is free from *unfounded sets*. Intuitively, the concept of unfounded set provides a direct formalization of "badly motivated" (as described in Section 3.2) conclusions.

Formally, a set X of atoms is called *unfounded* w.r.t. an interpretation I of a program P iff for all $a \in X$, every rule $r : a \leftarrow \alpha \in P$ satisfies either

(i) $X \cap \alpha \neq \emptyset$, or
(ii) $I_s(a, \rho_I(r)) < I(a)$, or
(iii) $I(r_b) = 0$

Intuitively, condition (i) above describes a circular motivation while (ii) asserts that a is overvalued w.r.t. r. Condition (iii) is needed to ensure that the semantics are a proper generalization of the classical semantics. Answer sets to a degree k are then defined in [80] as k-models that are free from unfounded sets, i.e. a k-model M is a k-answer set iff $supp(M) \cap X = \emptyset$ for any unfounded set X. A nice feature is that this single definition covers any program P, regardless of whether it is positive, or has constraint rules. One may wonder whether the concept of unfounded set can be generalized to AFASP programs. For example, a natural generalization would simply replace the circularity definition $X \cap \alpha \neq \emptyset$ above by "some element of X occurs as an argument in which the body expression α is increasing". However, this approach fails, as illustrated by the program P from Example 11 where it can easily be verified that $\{a\}$ would be unfounded w.r.t. the interpretation $\{a^1\}$ since $r_1 \colon a \leftarrow_m 0.2$ satisfies (ii) above while $r_2 \colon a \leftarrow_m (a > 0)$ satisfies (i). Note that this failure is only due to the presence of $(a > 0)$ in rule bodies, which are not allowed in the framework presented in [80].

In [35] it is shown that, when a total ordering is used in the lattice, the semantics of [80] correspond to the semantics obtained through the fixpoint definition used in this paper.
When the ordering used is not total, however, this equivalence is no longer valid. For example, consider the lattice $(\mathbb{B}\times\mathbb{B}, \leq)$ such that $(1,1)$ is the top element of the lattice, $(0,0)$ is the bottom element and $(0,0) \leq (0,1) \leq (1,1)$ and $(0,0) \leq (1,0) \leq (1,1)$. Now consider an AFASP program P, with $\mathcal{A}_P(\rho) = \inf\{\rho(r) \mid r \in P\}$, over this lattice:

$$r_1 : a \leftarrow (1,0)$$
$$r_2 : a \leftarrow (0,1)$$

According to the property of residual implicators that $x \rightarrow y = 1$ iff $x \leq y$, any interpretation I of P that satisfies rule r_1 to the degree $(1,1)$ must obey $I(a) \geq (1,0)$. Likewise any interpretation I that satisfies r_2 to the degree $(1,1)$ must obey $I(a) \geq (0,1)$. Hence the only 1-model of P is $I = \{a^{(1,1)}\}$. However, according to rule (ii) above $\{a\}$ is an unfounded set, which means that under the unfounded semantics I is not an answer set of P. On the other hand, $I = \Pi^*_{P^I,\rho_I}$, and thus I is an answer set of P according to the fixpoint semantics. If we consider rules as constraints that need to be fulfilled, the fixpoint semantics correspond better to our intuition.

5.4 Valued Constraint Satisfaction Problems

A classical constraint satisfaction problem (CSP) consists of a set of variables $X = \{x_1, \ldots, x_n\}$, a set of finite domains $D = \{d_1, \ldots, d_n\}$ such that variable x_i ranges over domain d_i, and a set of constraints C of the form $c = (X_c, R_c)$ such that $X_c \subseteq X$ is a set of variables and R_c is a relation between the variables in X_c. In Valued Constraint Satisfaction Problems (VCSPs) [68], a CSP is augmented with a cost function φ, which associates a cost to every constraint. A solution to a VCSP is then an assignment of values to the variables in X such that the aggregated cost of all violated constraints is minimal. Typically, costs are represented as real numbers, and the maximum or sum is used to aggregate.

In the crisp case it has been noted that answer set programming can be used for solving constraint satisfaction problems [59, 65]. The idea is to write an answer set program containing generate rules[4] that generate possible assignments of values to each of the variables, and constraints which remove those assignments that violate any constraints. In this way the resulting answer set program models a constraint satisfaction problem in the sense that answer sets of the program correspond to the solutions of the problem under consideration. It should come as no surprise that the AFASP framework can likewise be used for modeling VCSPs, as VCSPs can be seen as CSPs with an added aggregation operator. Basically, a VCSP corresponds to an AFASP program that only uses choice rules (which are the fuzzy equivalent of generate rules, i.e. rules assigning a random truth value to a certain atom) and constraints. Hard constraints correspond to rules that are required to be greater than 1 in the aggregator, whereas soft contstraints are rules whose valuation in the aggregator can be lower than 1, such as rules aggregated using the infimum. An example of the use of this paradigm

[4] These are rules involving cyclic negation such as the ASP program $\{a \leftarrow not\, b.\ b \leftarrow not\, a\}$. The answer sets of this program are $\{a\}$ and $\{b\}$, hence this program allows us to choose one of two options a and b in a solution of the modeled problem.

is the fuzzy graph coloring program introduced in Section 1, where we model the constraint satisfaction problem of coloring a graph with continuous colors, given some soft and hard constraints.

5.5 Answer Set Optimization

In [6, 7], a framework for answer set optimization is proposed. The basic idea is that one can state preference rules which are combined to define an ordering over answer sets. For example, if we have a program that generates a class room schedule, this framework allows to state that if teacher *John* is teaching *Math*, we prefer *John* to also teach *Physics* as follows:

$$teaches(John, Physics) : 0 > teaches(Mark, Physics) : 1 \leftarrow teaches(John, Math)$$

A rule is of the general form $C_1 : p_1 > \ldots > C_k : p_k \leftarrow a_1, \ldots, a_n, not\ b_1, \ldots, not\ b_m$, where $C_i : p_i$ encodes that the penalty associated with the rule is p_i if i is the lowest index for which the atom set C_i is true. However, the penalty of the rule is only taken into account if the conditions on the left hand side, expressed as a conjunction of atoms, are true. Different rules can then be combined using strategies, which encode importance among these preference rules. Formally, [6] defines a *Preference Description Language PDL* in which one can for example write that answer sets should be ordered using the Pareto ordering on rules r_1 and r_2 as $(pareto\, r_1, r_2)$. Many other complex (combinations of) orderings can be written in this language, such as $(lex\, (pareto\, r_1, r_2), r_3)$ which denotes that two answer sets first need to be compared using the Pareto ordering on rules r_1 and r_2; if they are Pareto-equal, then one must try to discriminate between them on the basis of rule r_3.

It is clear that the ideas of this approach and the one we proposed in this paper are very similar. In [36] we showed how this framework could be generalized to AFASP, using an appropriate aggregator. For a practical implementation of AFASP it seems interesting to adopt the same strategy of having a fixed language for specifying the aggregator. For example, we could then write an aggregator defining a lexicographical ordering over the program rules r_1, r_2, r_3 as $(lex\, r_1, r_2, r_3)$.

5.6 Embedding AFASP in FASP

One may wonder whether AFASP can be trivially embedded in FASP. Indeed, at first sight it seems that AFASP programs can be simulated by using a new rule $aggr \leftarrow f(a_1, \ldots, a_n)$, where a_i, for $i \in \{1, \ldots, n\}$, is a new atom that corresponds to the value of rule r_i. However, this intuitive embedding changes the semantics of an AFASP program. For example, consider the following AFASP program P:

$$r_1 : a \leftarrow_m \sim_l b$$
$$r_2 : b \leftarrow_m 0.7$$

We assume that the aggregator is $\mathcal{A}_P(\rho) = \min(\rho(r_1), \rho(r_2))$. Intuitively one may expect that P can be simulated using the following program P':

$$\begin{aligned} r_1 &: \quad a \leftarrow \sim_l b \\ r_2 &: \quad b \leftarrow 0.7 \\ r_a &: \quad r'_1 \leftarrow (\sim_l b \rightarrow_m a) \\ r_b &: \quad r'_2 \leftarrow (0.7 \rightarrow_m b) \\ r_{aggr} &: aggr \leftarrow \min(r'_1, r'_2) \end{aligned}$$

Now consider the 0.5-answer set $A = \{a^{0.5}, b^{0.5}\}$ of P. The corresponding interpretation $A' = \{a^{0.5}, b^{0.5}, {r'_1}^{1}, {r'_2}^{0.5}, aggr^{0.5}\}$ is not a model of P' because rule r_2 is not satisfied to the degree 1. Therefore it is also not an answer set of P', meaning this translation does not preserve the semantics.

The correct translation is provided by the following program P'':

$$\begin{aligned} r_1 &: \quad a \leftarrow \sim_l b \wedge_m r'_1 \\ r_2 &: \quad b \leftarrow 0.7 \wedge_m r'_2 \\ r_a &: \quad r'_1 \leftarrow (\sim_l b \rightarrow_m (\sim_l not_a)) \\ r_b &: \quad r'_2 \leftarrow (0.7 \rightarrow_m (\sim_l not_b)) \\ r_{na} &: not_a \leftarrow \sim_l a \\ r_{nb} &: not_b \leftarrow \sim_l b \\ r_{aggr} &: aggr \leftarrow \min(r'_1, r'_2) \end{aligned}$$

One can easily verify that $A'' = A' \cup \{not_a^{0.5}, not_b^{0.5}\}$ is an answer set of P''. Furthermore it can be shown that there is a bijection between the answer sets of P and P''. However, as is evident from the above, this translation is not very intuitive. Moreover it only works for aggregators that map rule interpretations to the truth lattice of the program and not for aggregators mapping rule interpretations to more interesting preorders. Hence, even though (some) AFASP programs can be translated to FASP programs, the study and use of AFASP on its own is still worthwhile.

6 Conclusions

FASP can be used for modeling continuous optimization problems. Due to the continuity, it is natural to have some notion of approximate answer sets, which correspond to models that satisfy some of the rules only partially. In current approaches, however, the idea of partial rule satisfaction is implemented in a rather limited way: users are required to state to what extent each rule should be satisfied.

In this paper, we have introduced a more flexible method for handling partial rule satisfaction, which is based on aggregation operators that combine the degrees to which each of the rules are satisfied to a single value from a preordered set. In this sense, in our approach, the weights that are assigned to rules are variable, and the aggregator determines which combinations of rule weights are most desirable. Our work extends a previous proposal for using aggregators for FASP programs in two ways. First, the preorder determined by the aggregator is decoupled from the truth lattice considered in the program. This makes it possible to express preference among solutions of problems in a more flexible way. Second, we have developed a fixpoint semantics, in contrast to existing work which is based on unfounded semantics. This has the advantage that we are no longer restricted to using t-norms in rule bodies and more closely shows the

links between the other FASP approaches. Furthermore it solves a problem with the previous proposal when generalizing from $([0,1],\leq)$ to arbitrary truth lattices.

We have also illustrated our approach on two examples: continuous graph coloring and the reviewer assignment problem. By means of these examples we have illustrated the practical usefulness of our framework.

The implementation of our framework depends on the aggregators and body functions that are used. In [34] we have shown that when the aggregator ranges over the truth lattice of the program and the body functions are t-norms, we can solve AFASP programs using fuzzy SAT solvers. When the leximin or discrimin is used, we can use techniques from constraint satisfaction solving such as [20]. Last, for programs with a linear aggregator and linear functions in rule bodies we can use multi-level mixed integer programming, as mentioned in [69]. In future research we plan to investigate these implementation methods in more detail.

Acknowledgements Jeroen Janssen is funded by a research project from the Flemish Fund for Scientific Research (FWO-Vlaanderen). Steven Schockaert is a post-doctoral fellow of the Flemish Fund for Scientific Research (FWO-Vlaanderen).

A Proofs

A.1 Proofs of Section 3

Proof (Proof of Proposition 1) Suppose $r : a \leftarrow \alpha$ is defined over a lattice $\mathcal{L}$, then:

$$\begin{array}{rcll} I_s(r,c) = & \langle \text{Def. } I_s(r,c)\rangle & \inf\{y \in \mathcal{L} \mid I(\alpha) \rightarrow y \geq c\} \\ = & \langle \text{Residuation principle}\rangle & \inf\{y \in \mathcal{L} \mid y \geq I(\alpha) \wedge_r c\} \\ = & \langle \text{See below}\rangle & I(\alpha) \wedge_r c \end{array}$$

The last step follows from the fact that $I(\alpha) \wedge_r c$ is an element of $\mathcal{L}$ and is a lower bound of $\{y \in \mathcal{L} \mid y \geq I(\alpha) \wedge_r c\}$.

Proof (Proof of Proposition 3)

$$\begin{array}{rcll} \Pi_{P,\rho_1}(I)(a) = & \langle \text{Def. } \Pi_{P,\rho_1}\rangle & \sup\{I_s(\rho_1, r) \mid r \in P_a\} \\ = & \langle \text{Proposition 1}\rangle & \sup\{I(r_b) \wedge_r \rho_1(r) \mid r \in P_a\} \\ \leq & \langle \text{Monot. t-norm}\rangle & \sup\{I(r_b) \wedge_r \rho_2(r) \mid r \in P_a\} \\ = & \langle \text{Proposition 1}\rangle & \sup\{I_s(\rho_2, r) \mid r \in P_a\} \\ = & \langle \text{Def. } \Pi_{P,\rho_2}\rangle & \Pi_{P,\rho_2}(I)(a) \end{array}$$

Proof (Proof of Proposition 4) It is straightforward to show, using transfinite induction and Propositions 2 and 3, that, for any ordinal i, $J^1_i \leq J^2_i$, where $\mathcal{S}\langle P, \rho_1\rangle = \langle J^1_i \mid i \text{ an ordinal}\rangle$ and $\mathcal{S}\langle P, \rho_2\rangle = \langle J^2_i \mid i \text{ an ordinal}\rangle$.

Indeed: $J^1_0 = J^2_0 = \emptyset$. Propositions 2 and 3 then ensure that, for a successor ordinal i, $J^1_i = \Pi_{P,\rho_1}(J^1_{i-1}) \leq \Pi_{P,\rho_2}(J^2_{i-1}) = J^2_i$ follows from $\rho_1 \leq \rho_2$ and the induction hypothesis. For a limit ordinal i, $J^1_i = \bigcup_{j<i} J^1_j \leq \bigcup_{j<i} J^2_j = J^2_i$ is immediate from the induction hypothesis.

Proof (Proof of Proposition 5) Since P is constraint-free, we know that $\mathcal{R}_P = \bigcup_{a \in \mathcal{B}_P} P_a$. Using this property we obtain the stated as follows:

$$\begin{array}{rcl} & & M = \Pi_{P,\rho}(M) \\ \equiv & \langle \text{Def. } \Pi_{P,\rho}(M)\rangle & \forall a \in \mathcal{B}_P \cdot M(a) = \sup\{M_s(r, \rho(r)) \mid r \in P_a\} \\ \Rightarrow & \langle \text{sup is upper bound}\rangle & \forall a \in \mathcal{B}_P \cdot \forall r \in P_a \cdot M(a) \geq M_s(r, \rho(r)) \\ \equiv & \langle \text{Prop. 1, } \mathcal{R}_P = \cup_{a \in \mathcal{B}_P} P_a\rangle & \forall r \in \mathcal{R}_P \cdot M(r_h) \geq M(r_b) \wedge_r \rho(r) \\ \equiv & \langle \text{Residuation principle}\rangle & \forall r \in \mathcal{R}_P \cdot M(r_b) \rightarrow M(r_h) \geq \rho(r) \\ \equiv & \langle \text{Def. } \rho\text{-rule model}\rangle & M \text{ is a } \rho\text{-rule model of } P \end{array}$$

Proof (Proof of Proposition 6) Since P is constraint-free, we know that $\mathcal{R}_P = \bigcup_{a \in \mathcal{B}_P} P_a$. Now, suppose that M is a minimal ρ-rule model of P and not a fixpoint of $\Pi_{P,\rho}$. Then we first show that $\forall a \in \mathcal{B}_P \cdot M(a) \geq \sup_{r \in P_a} M_s(r, \rho(r))$ as follows:

$$\begin{array}{lll} & & M \text{ is a } \rho\text{-rule model of } P \\ \equiv & \langle \text{Def. } \rho\text{-rule model} \rangle & \forall r \in \mathcal{R}_P \cdot M(r_b) \rightarrow_r M(r_h) \geq \rho(r) \\ \equiv & \langle \text{Residuation principle} \rangle & \forall r \in \mathcal{R}_P \cdot M(r_h) \geq M(r_b) \wedge_r \rho(r) \\ \equiv & \langle \text{Prop. 1, } \mathcal{R}_P = \cup_{a \in \mathcal{B}_P} P_a \rangle & \forall a \in \mathcal{B}_P \cdot \forall r \in P_a \cdot M(a) \geq M_s(r, \rho(r)) \\ \equiv & \langle \text{Def. upper bound, sup} \rangle & \forall a \in \mathcal{B}_P \cdot M(a) \geq \sup_{r \in P_a} M_s(r, \rho(r)) \end{array}$$

As M is not a fixpoint of $\Pi_{P,\rho}$, for some $a \in \mathcal{B}_P$ it must hold that $M(a) > \sup_{r \in P_a} M_s(r, \rho(r))$. Consider then the interpretation M' defined as

$$M'(x) = \begin{cases} M(x) & \text{if } x \neq a \\ \sup_{r \in P_a} M_s(r, \rho(r)) & \text{if } x = a \end{cases}$$

Clearly $M' < M$. We will show that M' is a ρ-rule model, leading to a contradiction with the minimality of M. For any $x \in (\mathcal{B}_P \setminus \{a\})$ we show that

$$\forall r \in P_x \cdot M'(r_b) \rightarrow M'(x) \geq \rho(r) \tag{12}$$

Indeed:

$$\begin{array}{lll} M' < M \Rightarrow & \langle \text{Body is incr. function} \rangle & M'(r_b) \leq M(r_b) \\ \Rightarrow & \langle \text{Anti-monoton. } \rightarrow \rangle & M'(r_b) \rightarrow M(x) \geq M(r_b) \rightarrow M(x) \\ \Rightarrow & \langle M \text{ is } \rho\text{-rule model} \rangle & M'(r_b) \rightarrow M(x) \geq \rho(r) \\ \equiv & \langle \text{Def. } M', r_h = x \neq a \rangle & M'(r_b) \rightarrow M'(x) \geq \rho(r) \end{array}$$

For a we show that

$$\forall r \in P_a \cdot M'(r_b) \rightarrow M'(a) \geq \rho(r) \tag{13}$$

as follows:

$$\begin{array}{lll} & & M'(a) = \sup_{r \in P_a} M_s(r, \rho(r)) \\ \Rightarrow & \langle \text{sup is upper bound} \rangle & \forall r \in P_a \cdot M'(a) \geq M_s(r, \rho(r)) \\ \equiv & \langle \text{Prop. 1} \rangle & \forall r \in P_a \cdot M'(a) \geq M(r_b) \wedge_r \rho(r) \\ \Rightarrow & \langle M' < M, \text{ Monot. } \wedge_r, (*) \rangle & \forall r \in P_a \cdot M'(a) \geq M'(r_b) \wedge_r \rho(r) \\ \equiv & \langle \text{Residuation principle} \rangle & \forall r \in P_a \cdot M'(r_b) \rightarrow M'(a) \geq \rho(r) \end{array}$$

The (*) justification is that r_b contains an increasing function. By combining (12) and (13), we obtain that M' is a ρ-rule model, contradicting the assumption that M is a minimal ρ-rule model. Hence M must be a fixpoint of $\Pi_{P,\rho}$.

Proof (Proof of Proposition 7) First, we show that $\Pi^*_{P,\rho}$ is a minimal ρ-rule model of P. Due to Proposition 5 we know that $\Pi^*_{P,\rho}$ must be a ρ-rule model of P. Suppose M is a ρ-rule model such that $M \leq \Pi^*_{P,\rho}$. Without loss of generalisation, we can assume that M is a minimal ρ-rule model. Now from Proposition 6 we know that M must be a fixpoint of $\Pi_{P,\rho}$ and hence, as $\Pi^*_{P,\rho}$ is the least fixpoint of $\Pi_{P,\rho}$, that $M = \Pi^*_{P,\rho}$. Thus $\Pi^*_{P,\rho}$ is a minimal ρ-rule model of P.

Second, we show that no other minimal ρ-rule models of P exist. Suppose M is a minimal ρ-rule model of P. From Proposition 6 we know that M must be a fixpoint of $\Pi_{P,\rho}$ and hence $\Pi^*_{P,\rho} \leq M$ as $\Pi^*_{P,\rho}$ is the least fixpoint of $\Pi_{P,\rho}$. From this it follows that $M = \Pi^*_{P,\rho}$.

Proof (Proof of Corollary 1) Follows immediately from Proposition 7.

Lemma 1 *Let P be an AFASP program and I an interpretation of this program, then for each $a \in \mathcal{B}_P$ it holds that $I(a)$ is an upper bound of the set $\{I_s(r, \rho_I(r)) \mid r \in P_a\}$.*

Proof We show that for any $a \in \mathcal{B}_P$ and $r \in P_a$ it holds that $I(a) \geq I_s(r, \rho_I(r))$, from which the stated readily follows.

$$\begin{array}{lll} I_s(r, \rho_I(r)) = & \langle \text{Def. } I_s(r, \rho_I(r)) \rangle & \inf\{y \in \mathcal{L}_P \mid I(r_b) \rightarrow y \geq \rho_I(r)\} \\ = & \langle \text{Def. } \rho_I \rangle & \inf\{y \in \mathcal{L}_P \mid I(r_b) \rightarrow y \geq I(r_b) \rightarrow I(a)\} \\ \leq & \langle \text{Def. inf} \rangle & I(a) \end{array}$$

Proof (Proof of Proposition 8) First, note that from Proposition 5 we can immediately see that $\rho \leq \rho_M$ as M is a fixpoint of $\Pi_{P,\rho}$. Second, we show that $\Pi_{P,\rho_M}(M) = M$:

$$\begin{array}{rrll} M = \Pi^*_{P,\rho} & \Rightarrow & \langle\text{Def. fixpoint}\rangle & \forall a \in \mathcal{B}_P \cdot M(a) = \Pi_{P,\rho}(M)(a) \\ & \Rightarrow & \langle \rho \leq \rho_M, \text{Prop. 3}\rangle & \forall a \in \mathcal{B}_P \cdot M(a) \leq \Pi_{P,\rho_M}(M)(a) \\ & \equiv & \langle\text{Def. } \Pi_{P,\rho_M}\rangle & \forall a \in \mathcal{B}_P \cdot M(a) \leq \sup_{r \in P_a} M_s(r, \rho_M(r)) \\ & \Rightarrow & \langle\text{Lemma 1}\rangle & \forall a \in \mathcal{B}_P \cdot M(a) = \sup_{r \in P_a} M_s(r, \rho_M(r)) \\ & \equiv & \langle\text{Def. } \Pi_{P,\rho_M}\rangle & \forall a \in \mathcal{B}_P \cdot M(a) = \Pi_{P,\rho_M}(M)(a) \end{array}$$

By Proposition 4 and the fact that $\rho \leq \rho_M$ we obtain that $\Pi^*_{P,\rho} \leq \Pi^*_{P,\rho_M}$ and thus by definition of M that $M \leq \Pi^*_{P,\rho_M}$. As we have shown that M is a fixpoint of Π_{P,ρ_M}, and thus $\Pi^*_{P,\rho_M} \leq M$, this means that $M = \Pi^*_{P,\rho_M}$.

Proof (Proof of Proposition 9) Due to Proposition 8 we already know that $M = \Pi^*_{P,\rho_M}$. We thus only need to show that $\mathcal{A}_P(\rho_M) \geq \mathcal{A}_P(\rho)$. From Proposition 5, we know that M is a ρ-rule model, i.e. $\forall r \in \mathcal{R}_P \cdot \rho_M(r) \geq \rho(r)$. This implies $\mathcal{A}_P(\rho_M) \geq \mathcal{A}_P(\rho)$ because $\mathcal{A}_P$ is increasing.

Proof (Proof of Corollary 2) First, suppose M is a k-answer set of a simple AFASP program P. By definition of k-answer sets it must then hold that $\mathcal{A}_P(\rho_M) \geq k$ and that $M = \Pi^*_{P,\rho_M}$, hence some rule interpretation ρ exists such that $M = \Pi^*_{P,\rho}$ and $\mathcal{A}_P(\rho) \geq k$.

Second, suppose there is some rule interpretation ρ for which $\mathcal{A}_P(\rho) \geq k$ and $M = \Pi^*_{P,\rho}$. From Proposition 9 we then know that M is a $\mathcal{A}_P(\rho)$-answer set of P, from which we know that $\mathcal{A}_P(\rho_M) \geq \mathcal{A}_P(\rho)$ and hence, as $\mathcal{A}_P(\rho) \geq k$, it follows that $\mathcal{A}_P(\rho_M) \geq k$. Thus M is a k-answer set.

Proof (Proof of Corollary 3) Let P be a simple AFASP program. The desired answer set is obtained by applying Proposition 8 to the rule interpretation $\rho_\top$.

Proof (Proof of Proposition 10) From definitions 11 and 12, it immediately follows that $\mathcal{A}_P(\rho_M) = \mathcal{A}_{P^M}(\rho_M) \geq k$.

Proof (Proof of Proposition 11) Let M be a k-answer set of P, then by definition of answer sets we know that $M = \Pi^*_{P^M,\rho_M}$. We will show by contradiction that no $M' \subset M$ can exist such that M' is a ρ_M-rule model of P. Suppose $M' \subset M$ such that M' is a ρ_M-rule model of P. From this it follows that for any rule $r \in P$ we have $M'(r_b) \geq M'(r_b^M)$ as only the arguments in which the body function is decreasing are replaced by their value in M. We can then proceed as follows:

$$\begin{array}{rrl} & & M' \text{ is a } \rho_M\text{-rule model of } P \\ \equiv & \langle\text{Def. } \rho_M\text{-rule model}\rangle & \forall r \in \mathcal{R}_P \cdot M'(r_b) \rightarrow M'(r_h) \geq \rho_M(r) \\ \equiv & \langle\text{Property reduct}\rangle & \forall r \in \mathcal{R}_P \cdot M'(r_b) \rightarrow M'(r_h) \geq \rho_M(r^M) \\ \Rightarrow & \langle M'(r_b) \geq M'(r_b^M)\rangle & \forall r \in \mathcal{R}_P \cdot M'(r_b^M) \rightarrow M'(r_h) \geq \rho_M(r^M) \\ \equiv & \langle r_h \in \mathcal{B}_P, \text{Def. reduct}\rangle & \forall r \in \mathcal{R}_P \cdot M'(r_b^M) \rightarrow M'(r_h^M) \geq \rho_M(r^M) \\ \equiv & \langle\text{Def. } P^M\rangle & \forall r \in \mathcal{R}_{P^M} \cdot M'(r_b) \rightarrow M'(r_h) \geq \rho_M(r) \\ \equiv & \langle\text{Def. } \rho_M\text{-rule model}\rangle & M' \text{ is a } \rho_M\text{-rule model of } P^M \end{array}$$

From Proposition 7 and the fact that $M = \Pi^*_{P^M,\rho_M}$ we know that M is the unique minimal ρ_M-rule model of P^M, leading to a contradiction with the fact that M' is a ρ_M-rule model of P^M and $M' \subset M$.

Lemma 2 *Let P be an AFASP program, then an interpretation I is a fixpoint of $\Pi_{P,\rho}$ iff it is a fixpoint of $\Pi_{P^I,\rho}$.*

Proof First, remark that for any expression α and interpretation I we have that $I(\alpha) = I(\alpha^I)$. Then we proceed as follows.

$$\begin{array}{rrl} & & I = \Pi_{P,\rho}(I) \\ \equiv & \langle\text{Equality of functions}\rangle & \forall a \in \mathcal{B}_P \cdot I(a) = \Pi_{P,\rho}(I)(a) \\ \equiv & \langle\text{Def. } \Pi_{P,\rho}\rangle & \forall a \in \mathcal{B}_P \cdot I(a) = \sup_{r \in P_a} I_s(r, \rho(r)) \\ \equiv & \langle\text{Prop. 1}\rangle & \forall a \in \mathcal{B}_P \cdot I(a) = \sup_{r \in P_a} I(r_b) \wedge_r \rho(r) \\ \equiv & \langle I(\alpha) = I(\alpha^I)\rangle & \forall a \in \mathcal{B}_P \cdot I(a) = \sup_{r \in P_a} I(r_b^I) \wedge_r \rho(r) \\ \equiv & \langle\text{Def. } \Pi_{P,\rho}, \text{Def. } P^I\rangle & \forall a \in \mathcal{B}_P \cdot I(a) = \Pi_{P^I,\rho}(I)(a) \\ \equiv & & I = \Pi_{P^I,\rho}(I) \end{array}$$

Proof (Proof of Proposition 12) From Lemma 2, we know that any answer set M must be a fixpoint of Π_{P,ρ_M}. Suppose M is not a minimal fixpoint and thus, that some N exists, $N \subset M$, such that N is a fixpoint of Π_{P,ρ_M}. We will show that any such N is also a ρ_M-rule model of P^M, leading to $N \supseteq M$ due to Proposition 7, a contradiction.

First, as $N \subset M$ and since the reduct only substitutes negative subexpressions for their corresponding values, it follows that, for any expression α:

$$N(\alpha^N) \geq N(\alpha^M)$$

and thus, since $N(\alpha) = N(\alpha^N)$ that

$$N(\alpha) \geq N(\alpha^M)$$

We can now show that N is a ρ_M-rule model of P^M as follows.

$$\begin{array}{lcl} & & N = \Pi_{P,\rho_M}(N) \\ \Rightarrow & \langle \text{Prop. 5} \rangle & \forall r \in \mathcal{R}_P \cdot N(r_b) \rightarrow N(r_h) \geq \rho_M(r) \\ \Rightarrow & \langle \text{Anti-monot. } \rightarrow, N(r_b) \geq N(r_b^M) \rangle & \forall r \in \mathcal{R}_P \cdot N(r_b^M) \rightarrow N(r_h) \geq \rho_M(r) \\ \equiv & \langle \text{Def. reduct} \rangle & \forall r \in \mathcal{R}_{P^M} \cdot N(r_b) \rightarrow N(r_h) \geq \rho_M(r) \\ \equiv & \langle \text{Def. } N(r) \rangle & \forall r \in \mathcal{R}_{P^M} \cdot N(r) \geq \rho_M(r) \end{array}$$

A.2 Proofs of Section 5

Proof (Proof of Proposition 13) First, remark that there is only a single rule interpretation ρ such that $\forall r \in P_{weight} \cdot \rho(r) = 1$ holds, viz. $\rho_\top$. As A is only a 1-answer set of P_{weight} iff $A = \Pi^*_{P,\rho}$ for some rule interpretation satisfying $\forall r \in P_{weight} \cdot \rho(r) = 1$ due to Corollary 2, it must hold that there is a unique 1-answer set A of P_{weight} where $A = \Pi^*_{P,\rho_\top}$. We can now show that $\Pi_P = \Pi_{P,\rho_\top}$, leading to the stated equivalence, as follows. Suppose I is some interpretation of P (and hence also of P_{weight}) and $l \in \mathcal{B}_P$ (and hence also $l \in \mathcal{B}_{P_{weight}}$), then:

$$\begin{array}{rcl} \Pi_P(I)(l) = & \langle \text{Def. } \Pi_P \rangle & \sup\{I(r_b \wedge_r r_w) \mid r \in P_l\} \\ = & \langle \text{Def. } P_{weight} \rangle & \sup\{I(r_b) \mid r \in (P_{weight})_l\} \\ = & \langle \text{Def. } \rho_\top \rangle & \sup\{I(r_b \wedge_r \rho_\top(r)) \mid r \in (P_{weight})_l\} \\ = & \langle \text{Def. } \Pi_{P_{weight},\rho_\top} \rangle & \Pi_{P_{weight},\rho_\top}(I)(l) \end{array}$$

From this, the stated readily follows.

Proof (Proof of Proposition 14) The proof follows easily from Proposition 13 and the fact that the Gelfond-Lifschitz reduct introduced in [53] is a special case of our reduct.

References

1. Fundamentals of Uncertainty Calculi with Applications to Fuzzy Inference. Kluwer Academics Publishers (1995)
2. Alsinet, T., Godo, L., Sandri, S.: Two formalisms of extended possibilistic logic programming with context-dependent fuzzy unification: A comparative description. Electronic Notes in Theoretical Computer Science **66**(5), 1 – 21 (2002)
3. Balog, K., de Rijcke, M.: Determining expert profiles (with an application to expert finding). In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI07), pp. 2657–2662 (2007)
4. Baral, C.: Knowledge Representation, Reasoning and Declarative Problem Solving. Cambridge University Press (2003)
5. Bauters, K., Schockaert, S., De Cock, M., Vermeir, D.: Possibilistic answer set programming revisited. In: Proceedings of the 26th Conference on Uncertainty in Artificial Intelligence (UAI) (2010)
6. Brewka, G.: Complex preferences for answer set optimization. In: D. Dubois, C.A. Welty, M.A. Williams (eds.) KR, pp. 213–223 (2004)

7. Brewka, G., Niemelä, I., Truszczyński, M.: Answer set optimization. In: Proceedings of the 18th International Joint Conference on Artificial Intelligence, pp. 867–872 (2003)
8. Cao, T.H.: Annotated fuzzy logic programs. Fuzzy Sets & Systems **113**(2), 277–298 (2000)
9. Choquet, G.: Theory of capacities. Annales de l'Institut Fourier **5**, 131–295 (1954)
10. Damásio, C.V., Medina, J., Ojeda-Aciego, M.: Sorted multi-adjoint logic programs: termination results and applications. In: Proceedings of the 9th European Conference on Logics in Artificial Intelligence (JELIA'04), pp. 260–273 (2004)
11. Damásio, C.V., Medina, J., Ojeda-Aciego, M.: Termination of logic programs with imperfect information: applications and query procedure. Journal of Applied Logic **5**(3), 435–458 (2007)
12. Damásio, C.V., Pereira, L.M.: Hybrid probabilistic logic programs as residuated logic programs. In: Proceedings of the 7th European Workshop on Logics in Artificial Intelligence (JELIA'00), pp. 57–72 (2000)
13. Damásio, C.V., Pereira, L.M.: Antitonic logic programs. In: Proceedings of the 6th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'01), pp. 379–392 (2001)
14. Damásio, C.V., Pereira, L.M.: Monotonic and residuated logic programs. In: Proceedings of the 6th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU'01), pp. 748–759 (2001)
15. Damásio, C.V., Pereira, L.M.: Sorted monotonic logic programs and their embeddings. In: Proceedings of Information Processing and Management of Uncertainty (IPMU04), pp. 807–814 (2004)
16. De Cock, M.: A thorough study of linguistic modifiers in fuzzy set theory. Ph.D. thesis, Ghent University (2002)
17. Dekhtyar, A., Subrahmanian, V.S.: Hybrid probabilistic programs. In: Proceedings of the Fourteenth International Conference on Logic Programming (ICLP'97), pp. 391–405 (1997)
18. Detyniecki, M.: Numerical aggregation operators: state of the art. In: Proceedings of the First International Summer School on Aggregation Operators and their Applications (2001)
19. Dubois, D., Fargier, H., Prade, H.: Refinements of the maximin approach to decision-making in a fuzzy environment. Fuzzy Sets & Systems **81**(1), 103–122 (1996)
20. Dubois, D., Fortemps, P.: Computing improved optimal solutions to max-min flexible constraint satisfaction problems. European Journal of Operational Research **118**, 95–126 (1999)
21. Dubois, D., Lang, J., Prade, H.: Towards possibilistic logic programming. In: Proceedings of the Eigth International Conference on Logic Programming (ICLP'91), pp. 581–595 (1991)
22. Dubois, D., Prade, H.: Weighted minimum and maximun operations in fuzzy sets theory. Information Sciences **39**(2), 205–210 (1986)
23. Dubois, D., Prade, H.: Advances in the egalitarist approach to decision-making in a fuzzy environment. In: Y. Yoshida (ed.) Dynamical Aspect in Fuzzy Decision Making, *Studies in Fuzziness and Soft Computing*, vol. 73, pp. 213–240 (2001)
24. Emden, M.H.v.: Quantitative deduction and its fixpoint theory. Journal of Logic Programming **30**(1), 37–53 (1986)
25. Fagin, R., Wimmers, E.L.: A formula for incorporating weights into scoring rules. Theoretical Computer Science **239**, 309–338 (1998)
26. Fitting, M.: Bilattices and the semantics of logic programming. Journal of Logic Programming **11**(2), 91–116 (1991)
27. Fodor, J.C., Yager, R.R., Rybalov, A.: Structure of uninorms. Int. J. Uncertain. Fuzziness Knowl.-Based Syst. **5**(4), 411–427 (1997)
28. Fuhr, N.: Probabilistic datalog: implementing logical information retrieval for advanced applications. Journal of the American Society for Information Science **51**(2), 95–110 (2000)
29. Garg, N., Kavitha, T., Kumar, A., Mehlhorn, K., Mestre, J.: Assigning papers to referees (2008)
30. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: Proceedings of the Fifth International Conference and Symposium on Logic Programming (ICLP/SLP'88), pp. 1081–1086 (1988)
31. Grabisch, M.: k-additive fuzzy measures. In: Proceedings of the 6th international conference on information processing and management of uncertainty in Knowledge-Based systems (IPMU96) (1996)

32. Hájek, P.: Metamathematics of Fuzzy Logic (Trends in Logic) (2001)
33. Ishizuka, M., Kanai, N.: Prolog-elf incorporating fuzzy logic. In: Proceedings of the 9th international joint conference on Artificial intelligence (IJCAI'85), pp. 701–703 (1985)
34. Janssen, J., Heymans, S., Vermeir, D., De Cock, M.: Compiling fuzzy answer set programs to fuzzy propositional theories. In: Proceedings of the 24th International Conference on Logic Programming (ICLP08) (2008)
35. Janssen, J., Schockaert, S., Vermeir, D., De Cock, M.: Reducing fuzzy answer set programming to model finding in fuzzy logics. Submitted.
36. Janssen, J., Schockaert, S., Vermeir, D., De Cock, M.: Fuzzy answer set programming with literal preferences. In: Proceedings of the Joint 2009 International Fuzzy Systems Association World Congress and 2009 European Society of Fuzzy Logic and Technology Conference (IFSA/EUSFLAT2009), pp. 1347–1352 (2009)
37. Janssen, J., Schockaert, S., Vermeir, D., De Cock, M.: General fuzzy answer set programs. In: V.D. Gesù, S.K. Pal, A. Petrosino (eds.) WILF, *Lecture Notes in Computer Science*, vol. 5571, pp. 352–359 (2009)
38. Kifer, M., Li, A.: On the semantics of rule-based expert systems with uncertainty. In: Proceedings of the 2nd International Conference on Database Theory (ICDT'88), pp. 102–117 (1988)
39. Kifer, M., Subrahmanian, V.S.: Theory of generalized annotated logic programming and its applications. Journal of Logic Programming **12**(3&4), 335–367 (1992)
40. Lakshmanan, L.V.S.: An epistemic foundation for logic programming with uncertainty. In: Proceedings of the 14th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'94), pp. 89–100 (1994)
41. Lakshmanan, L.V.S., Sadri, F.: Modeling uncertainty in deductive databases. In: Proceedings of the 5th International Conference on Database and Expert Systems Applications (DEXA'94), pp. 724–733 (1994)
42. Lakshmanan, L.V.S., Sadri, F.: Probabilistic deductive databases. In: Proceedings of the 1994 International Symposium on Logic programming (ILPS'94), pp. 254–268. MIT Press, Cambridge, MA, USA (1994)
43. Lakshmanan, L.V.S., Sadri, F.: Uncertain deductive databases: a hybrid approach. Information Systems **22**(9), 483–508 (1997)
44. Lakshmanan, L.V.S., Shiri, N.: A parametric approach to deductive databases with uncertainty. IEEE Transactions on Knowledge and Data Engineering **13**(4), 554–570 (2001)
45. Lakshmanan, V.S.: Towards a generalized theory of deductive databases with uncertainty. Ph.D. thesis, Concordia University (1997)
46. Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., Scarcello, F.: The dlv system for knowledge representation and reasoning. ACM Transactions on Computational Logic **7**(3), 499–562 (2006)
47. Loyer, Y., Straccia, U.: The well-founded semantics in normal logic programs with uncertainty. In: Proceedings of the 6th International Symposium on Functional and Logic Programming (FLOPS'02), pp. 152–166 (2002)
48. Loyer, Y., Straccia, U.: The approximate well-founded semantics for logic programs with uncertainty. In: Proceedings of the 28th International Symposium on Mathematical Foundations of Computer Science (MFCS'03), pp. 541–550 (2003)
49. Luhandjula, M.K.: Compensatory operators in fuzzy linear programming with multiple objectives. Fuzzy Sets & Systems **8**(3), 245–252 (1982)
50. Lukasiewicz, T.: Probabilistic logic programming. In: Proceedings of the 13th European Conference on Artificial Intelligence (ECAI'98), pp. 388–392 (1998)
51. Lukasiewicz, T.: Many-valued disjunctive logic programs with probabilistic semantics. In: Proceedings of the 5th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'99), pp. 277–289 (1999)
52. Lukasiewicz, T.: Fuzzy description logic programs under the answer set semantics for the semantic web. In: Proceedings of the Second International Conference on Rules and Rule Markup Languages for the Semantic Web (RuleML'06), pp. 89–96 (2006)
53. Lukasiewicz, T., Straccia, U.: Tightly integrated fuzzy description logic programs under the answer set semantics for the semantic web. In: Proceedings of the First International Conference on Web Reasoning and Rule Systems (RR'07), pp. 289–298 (2007)
54. Lukasiewicz, T., Straccia, U.: Top-k retrieval in description logic programs under vagueness for the semantic web. In: Proceedings of the 1st international conference on Scalable Uncertainty Management (SUM'07), pp. 16–30 (2007)

55. Madrid, N., Ojeda-Aciego, M.: Towards a fuzzy answer set semantics for residuated logic programs. In: Proceedings of the 2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT'08), pp. 260–264 (2008)
56. Madrid, N., Ojeda-Aciego, M.: On coherence and consistence in fuzzy answer set semantics for residuated logic programs. In: Proceedings of the 8th International Workshop on Fuzzy Logic and Applications (WILF'09), pp. 60–67 (2009)
57. Madrid, N., Ojeda-Aciego, M.: Measuring instability in normal residuated logic programs: discarding information. Communications in Computer and Information Science **80**, 128–137 (2010)
58. Madrid, N., Ojeda-Aciego, M.: On the measure of instability in normal residuated logic programs. In: Proceedings of FUZZ-IEEE'10 (2010)
59. Marek, V., Truszczyński, M.: Stable models and an alternative logic programming paradigm, chap. 7, pp. 169–181 (1999)
60. Nerode, A., Remmel, J.B., Subrahmanian, V.S.: Annotated nonmonotonic rule systems. Theoretical Computer Science **171**(1-2), 111–146 (1997)
61. Ng, R., Subrahmanian, V.S.: A semantical framework for supporting subjective and conditional probabilities in deductive databases. Journal of Automated Reasoning **10**(2), 191–235 (1993)
62. Ng, R., Subrahmanian, V.S.: Stable semantics for probabilistic deductive databases. Information and Computation **110**(1), 42–83 (1994)
63. Nicolas, P., Garcia, L., Stéphan, I.: Possibilistic stable models. In: Nonmonotonic Reasoning, Answer Set Programming and Constraints, Dagstuhl Seminar Proceedings (2005)
64. Nicolas, P., Garcia, L., Stéphan, I., Lefèvre, C.: Possibilistic uncertainty handling for answer set programming. Annals of Mathematics and Artificial Intelligence **47**(1-2), 139–181 (2006)
65. Niemelä, I.: Logic programs with stable model semantics as a constraint programming paradigm. Annals of Mathematics and Artificial Intelligence **25**, 241–273 (1999)
66. Novák, V., Perfilieva, I., Močkoř, J.: Mathematical Principles of Fuzzy Logic. Kluwer Academic Publishers (1999)
67. Saad, E.: Extended fuzzy logic programs with fuzzy answer set semantics. In: Proceedings of the 3rd International Conference on Scalable Uncertainty Management (SUM'09), pp. 223–239 (2009)
68. Schiex, T., Fargier, H., Verfaillie, G.: Valued constraint satisfaction problems: hard and easy problems. In: Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI'95), pp. 631–637 (1995)
69. Schockaert, S., Janssen, J., Vermeir, D., De Cock, M.: Answer sets in a fuzzy equilibrium logic. In: Proceedings of Rule Reasoning 2009 (RR2009), pp. 135–149 (2009)
70. Shapiro, E.Y.: Logic programs with uncertainties: a tool for implementing rule-based systems. In: Proceedings of the Eighth international joint conference on Artificial intelligence (IJCAI'83), pp. 529–532 (1983)
71. Simons, P.: Extending and implementing the stable model semantics. Ph.D. thesis, Helsinki University of Technology (2000)
72. Straccia, U.: Query answering in normal logic programs under uncertainty. In: In 8th European Conferences on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU-05), pp. 687–700 (2005)
73. Straccia, U.: Annotated answer set programming. In: Proceedings of the 11th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU'06) (2006)
74. Straccia, U.: Managing uncertainty and vagueness in description logics, logic programs and description logic programs. In: Reasoning Web: 4th International Summer School 2008, pp. 54–103 (2008)
75. Straccia, U., Ojeda-Aciego, M., Damásio, C.V.: On fixed-points of multivalued functions on complete lattices and their application to generalized logic programs. SIAM Journal on Computing **38**(5), 1881–1911 (2009)
76. Subrahmanian, V.S.: Amalgamating knowledge bases. ACM Transactions on Database Systems **19**(2), 291–331 (1994)
77. Sugeno, M.: Theory of fuzzy integrals and its application. Ph.D. thesis, Tokyo Institute of Technology (1974)
78. Tarski, A.: A lattice theoretical fixpoint theorem and its application. Pacific Journal of Mathematics **5**, 285–309 (1955)

79. Turksen, I.B.: Interval-valued fuzzy sets and "compensatory and". Fuzzy Sets & Systems **51**(3), 295–307 (1992)
80. Van Nieuwenborgh, D., De Cock, M., Vermeir, D.: An introduction to fuzzy answer set programming. Annals of Mathematics and Artificial Intelligence **50**(3-4), 363–388 (2007)
81. Vojtás, P.: Fuzzy logic programming. Fuzzy Sets and Systems **124**(3), 361–370 (2001)
82. Wagner, G.: Negation in fuzzy and possibilistic logic programs. Uncertainty Theory in Artificial Intelligence Series (3), 113–128 (1998)
83. Yager, R.R.: A new methodology for ordinal multiple aspect decisions based on fuzzy sets. Decision Sciences **12**, 589–600 (1981)
84. Yager, R.R.: On ordered weighted averaging aggregation operators in multicriteria decisionmaking. IEEE Transactions on Systems, Man and Cybernetics **18**(1), 183–190 (1988)
85. Yager, R.R., Kacprzyk, J. (eds.): The ordered weighted averaging operators: theory and applications. Kluwer Academic Publishers (1997)

$about(P,T)$	t_1	t_2	t_3	t_4	t_5	t_6	t_7
p_1	0.8	1	0.2	0.4	0.2	0	0
p_2	0	0.2	0.7	0.5	0.9	0.4	0.3
p_3	1	0.1	0.1	0.2	0.3	0.2	0
p_4	0.5	0	0	1	0.2	0.4	0.8
p_5	0.3	0.9	0.8	0.9	0.4	0.2	0.1
p_6	0.6	0.8	0.3	0.7	0.8	0.3	0
p_7	0	1	1	1	0	0.3	0.1
p_8	0.9	0.2	0.1	0.7	0	0.1	1
p_9	0.1	0.7	0.1	0.2	0.1	1	0.7
p_{10}	0.7	0.6	0.6	0.3	0.2	1	0.1

Table 3: $about(P, T)$ scores

$expert(R,T)$	t_1	t_2	t_3	t_4	t_5	t_6	t_7
r_1	0.8	0.5	0.2	0.5	0.9	1	0
r_2	0.2	0.9	1	0.4	0.2	0.1	0.1
r_3	0.5	0.2	0.8	0.7	0.4	0.6	0.2
r_4	0	0.3	0.2	0.4	0.5	0.9	0.9
r_5	1	0.7	0.8	0.6	0.9	0.7	0.7

Table 4: $expert(R, T)$ scores

$close(U,U')$	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8	u_9	u_{10}
u'_1	0.5	0.3	0.4	0.8	0.1	1	0.4	0.3	0.9	0.4
u'_2	0.5	0.8	0.5	0.4	0.4	0.5	0.1	0	0.4	0.1
u'_3	0	0	0	1	0.3	0.6	0.3	0.9	0.2	0.1
u'_4	1	0.9	0.4	0.2	0.1	1	0.5	0.2	0.9	1
u'_5	0.8	0.1	0.1	0.4	0.2	0.3	1	0.3	0.1	0.2

Table 5: $close(U, U')$ scores

$conflict(R,P)$	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}
r_1	0.5	0.3	0.4	0.8	0.1	1	0.4	0.3	0.9	0.4
r_2	0.5	0.8	0.5	0.4	0.4	0.5	0.1	0	0.4	0.1
r_3	0	0	0	1	0.3	0.6	0.3	0.9	0.2	0.1
r_4	1	0.9	0.4	0.2	0.1	1	0.5	0.2	0.9	1
r_5	0.8	0.1	0.1	0.4	0.2	0.3	1	0.3	0.1	0.2

Table 6: $conflict(R, P)$ scores

$appropriate(R,P)$	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}
r_1	0.5	0.7	0.6	0.2	0.4	0	0.5	0.7	0.1	0.6
r_2	0.5	0.2	0.2	0.4	0.6	0.5	0.9	0.1	0.6	0.6
r_3	0.3	0.5	0.5	0	0.6	0.4	0.7	0.1	0.6	0.6
r_4	0	0.1	0.1	0.7	0.3	0	0.4	0.8	0.1	0
r_5	0.2	0.8	0.9	0.6	0.6	0.7	0	0.7	0.7	0.7

Table 7: $appropriate(R, P)$ scores

assignments	A_1	A_2
p_1	r_1, r_2, r_3	r_1, r_2, r_4
p_2	r_1, r_3, r_5	r_1, r_3, r_4
p_3	r_1, r_3, r_5	r_1, r_4, r_5
p_4	r_2, r_4, r_5	r_2, r_4, r_5
p_5	r_2, r_3, r_5	r_2, r_3, r_5
p_6	r_2, r_3, r_5	r_2, r_3, r_5
p_7	r_1, r_2, r_3	r_1, r_2, r_3
p_8	r_1, r_4, r_5	r_1, r_4, r_5
p_9	r_2, r_3, r_5	r_2, r_3, r_5
p_{10}	r_1, r_3, r_5	r_1, r_3, r_5

Table 8: Assignments for answer sets A_1 and A_2

$inappropriate(R, P)$	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}
r_1	0	0	0	0.8	0.6	1	0	0	0.9	0
r_2	0	0.8	0.8	0	0	0	0	0.9	0	0.4
r_3	0	0	0	1	0	0	0	0.9	0	0
r_4	1	0.9	0.9	0	0.7	1	0.6	0	0.9	1
r_5	0.8	0	0	0	0	0	1	0	0	0

Table 9: $inappropriate(R, P)$ scores for A_1

$inappropriate(R, P)$	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}
r_1	0	0	0	0.8	0.6	1	0	0	0.9	0
r_2	0	0.8	0.8	0	0	0	0	0.9	0	0.4
r_3	0.7	0	0.5	1	0	0	0	0.9	0	0
r_4	0	0	0	0	0.7	1	0.6	0	0.9	1
r_5	0.8	0.2	0	0	0	0	1	0	0	0

Table 10: $inappropriate(R, P)$ scores for A_2