Fuzzy Answer Set Programming

Davy Van Nieuwenborgh[1,⋆], Martine De Cock[2], and Dirk Vermeir[1]

[1] Vrije Universiteit Brussel, VUB
Dept. of Computer Science
Pleinlaan 2, B-1050 Brussels, Belgium
{dvnieuwe,dvermeir}@vub.ac.be
[2] Universiteit Gent, UGent
Dept. of Applied Mathematics and Computer Science
Krijgslaan 281 (S9), B-9000 Ghent, Belgium
martine.decock@ugent.be

Abstract. In this paper we show how the concepts of answer set programming and fuzzy logic can be succesfully combined into the single framework of fuzzy answer set programming (FASP). The framework offers the best of both worlds: from the answer set semantics, it inherits the truly declarative non-monotonic reasoning capabilities while, on the other hand, the notions from fuzzy logic in the framework allow it to step away from the sharp principles used in classical logic, e.g., that something is either completely true or completely false. As fuzzy logic gives the user great flexibility regarding the choice for the interpretation of the notions of negation, conjunction, disjunction and implication, the FASP framework is highly configurable and can, e.g., be tailored to any specific area of application. Finally, the presented framework turns out to be a proper extension of classical answer set programming, as we show, in contrast to other proposals in the literature, that there are only minor restrictions one has to demand on the fuzzy operations used, in order to be able to retrieve the classical semantics using FASP.

1 Introduction

The answer set programming (ASP) paradigm [15] has gained a lot of popularity in the last years, due to its truly declarative non-monotonic semantics, which has been proven useful in a number of interesting applications, e.g. [21, 12, 19, 16]. The idea behind the answer set semantics, a generalisation of the stable model semantics [14], is both intuitive and elegant. Given a program P and a candidate answer set M, one computes a reduct program P^M of a simpler type for which a semantics $(P^M)^\star$ is known. The reduct P^M is obtained from P by taking into account the consequences of accepting the proposed truth values of the literals in M. The candidate set M is then an answer set just when $(P^M)^\star = M$, i.e. M is "self-producible".

Although ASP provides a powerful solution for knowledge representation and non-monotonic reasoning, it has some drawbacks regarding the configureability of the semantics w.r.t. the type of application under consideration, as witnessed by the large

⋆ Supported by the Flemish Fund for Scientific Research (FWO-Vlaanderen).

number of extensions, both syntactically and semantically, that have been proposed in the literature [7, 10, 5, 2]. E.g., most[3] ASP semantics demand that a solution to a program satisfies all the rules. Further, the literals available in the program, i.e. the building blocks of rules, can only be true or false (or unknown when one considers the well-founded semantics [23]), and classical consistency is mandatory, i.e. a and $\neg a$ cannot be true at the same time (or not even "a bit" true at the same time). Also the interpretation of negation as failure, the construct that gives ASP its non-monotonicity, is very sharp: *not* a is true iff a is not true.

Sometimes however, it is impossible to find a solution that fully satisfies all rules of the program. In this case, one might still wish to look for a solution satisfying the program at least to a reasonably high degree. At other times, it may not even be required to obtain a solution that satisfies a program fully. That is, one might be more interested in a solution satisfying the program to a satisfactory high degree, especially if this solution comes at a lower cost. Consider the following problem based on an example from [2].

Example 1. There are four different kinds of sports that we like to practice to some degree. However, only certain combinations of sports lead to a full-body exercise. Furthermore, some of the sports complement each other, i.e. less practice of one automatically leads to more practice of the other (rules $r_1 \dots r_4$ in the program below).

$$
\begin{aligned}
r_1 &: & \mathit{lift\_weights} &\leftarrow \textit{not } \mathit{swim} \\
r_2 &: & \mathit{swim} &\leftarrow \textit{not } \mathit{lift\_weights} \\
r_3 &: & \mathit{run} &\leftarrow \textit{not } \mathit{play\_ball} \\
r_4 &: & \mathit{play\_ball} &\leftarrow \textit{not } \mathit{run} \\
r_5 &: \mathit{full\_body\_exercise} & &\leftarrow \mathit{lift\_weights}, \mathit{run} \\
r_6 &: \mathit{full\_body\_exercise} & &\leftarrow \mathit{swim}, \mathit{play\_ball} \\
r_7 &: & &\leftarrow \textit{not } \mathit{full\_body\_exercise}
\end{aligned}
$$

The two classical answer sets of this program are $\{\mathit{full\_body\_exercise}, \mathit{lift\_weights}, \mathit{run}\}$ and $\{\mathit{play\_ball}, \mathit{full\_body\_exercise}, \mathit{swim}\}$. Hence, to achieve a full body exercise, one needs to practice either weight lifting and running, or ball playing and swimming to the highest degree. However, in addition, we might be interested to know which combinations of the four sports we should practice, and to what degree, such that an acceptable degree, e.g. 0.7, of full-body exercise is obtained.

Fuzzy logic is a suitable framework for dealing with degrees of truth and satisfaction [26]. In its most general form, fuzzy logic considers a complete lattice $\mathcal{L}$ of truth values on which it redefines the classical operations of negation, conjunction, disjunction and implication; in such a way that they correspond to the classical ones in the top and bottom elements of the lattice. One of the strengths of fuzzy logic regarding these operations is that a user can freely choose, depending on the type of application under consideration, which specific definition she uses for the operations.

A combination with fuzzy logic increases the flexibility and hence the application potential of ASP. Such flexibility can be introduced at several levels. In the fuzzy answer set programming (FASP) framework introduced in this paper, we consider fuzzy answer sets, which means that literals can belong to an answer set to a certain extent, as opposed

[3] Some semantics that deal with preferences among rules [13, 6, 24] are more flexible.

to either belonging to the answer set or not. In accordance, the literals in a program can be true to a certain degree. We relax the definition of consistency to allow that, if desired, both a and $\neg a$ can be true to a certain degree at the same time without necessarily loosing consistency. Similarly, we allow for a more flexible interpretation of negation as failure. Crucial to our approach is the notion of a satisfaction function, as it enables us to compute the extent to which a rule is satisfied under a given fuzzy interpretation. The satisfaction function is then used to develop the concept of a fuzzy model. As in traditional ASP, in FASP the fuzzy answer sets of simple programs, i.e. programs without negation as failure, coincide with the fuzzy minimal models. For programs containing negation as failure, the idea underlying GL-reduct is extended to a technique that allows to bring to surface whether a fuzzy model is indeed supported by a program, in other words whether it deserves the name of fuzzy answer set.

The rest of the paper is organized as follows. In Section 2 we give some preliminaries on fuzzy logic and answer set programming, while we introduce the combination of both, i.e. fuzzy answer set programming (FASP), in Section 3. Before giving some comparison with related work in Section 5, we show in Section 4 how the classical answer set semantics can be retrieved from FASP. Finally, we conclude and give some directions for future research in Section 6.

2 Preliminaries

2.1 Truth Lattices

In this paper, we consider a complete truth lattice, i.e. a partially ordered set $(\mathcal{L}, \leq_{\mathcal{L}})$ such that every subset of $\mathcal{L}$ has an infimum (greatest lower bound) and a supremum (least upper bound), which we denote by $\inf$ and $\sup$ respectively [4]. Such a lattice is often denoted by $\mathcal{L}$, tacitly assuming the ordering $\leq_{\mathcal{L}}$. Furthermore, we use $0_{\mathcal{L}}$ and $1_{\mathcal{L}}$ to denote respectively the smallest and the greatest element[4] of $\mathcal{L}$.

The traditional logical operations of negation, conjunction, disjunction, and implication can be generalized to logical operators acting on truth values of $\mathcal{L}$ (see e.g. [20]). A *negator* on $\mathcal{L}$ is any decreasing $\mathcal{L} \to \mathcal{L}$ mapping $\mathcal{N}$ satisfying $\mathcal{N}(0_{\mathcal{L}}) = 1_{\mathcal{L}}$ and $\mathcal{N}(1_{\mathcal{L}}) = 0_{\mathcal{L}}$. It is called involutive if $\mathcal{N}(\mathcal{N}(x)) = x$ for all x in $\mathcal{L}$. A *triangular norm* $\mathcal{T}$ on $\mathcal{L}$ is any commutative and associative $\mathcal{L}^2 \to \mathcal{L}$ mapping $\mathcal{T}$ satisfying $\mathcal{T}(1_{\mathcal{L}}, x) = x$, for all x in $\mathcal{L}$. Moreover we require $\mathcal{T}$ to be increasing in both of its components. A triangular norm, or t-norm for short, corresponds to conjunction. A *triangular conorm* $\mathcal{S}$ on $\mathcal{L}$ is any increasing, commutative and associative $\mathcal{L}^2 \to \mathcal{L}$ mapping satisfying $\mathcal{S}(0_{\mathcal{L}}, x) = x$, for all x in $\mathcal{L}$. Moreover we require $\mathcal{S}$ to be increasing in both of its components. A triangular conorm, t-conorm for short, corresponds to disjunction. An *implicator* $\mathcal{I}$ on $\mathcal{L}$ is any $\mathcal{L}^2 \to \mathcal{L}$–mapping satisfying $\mathcal{I}(0_{\mathcal{L}}, 0_{\mathcal{L}}) = 1_{\mathcal{L}}$, and $\mathcal{I}(1_{\mathcal{L}}, x) = x$, for all x in $\mathcal{L}$. Moreover we require $\mathcal{I}$ to be decreasing in its first, and increasing in its second component.

The dual of a t-norm $\mathcal{T}$ w.r.t. a negator $\mathcal{N}$ is a t-conorm $\mathcal{S}$ defined as $\mathcal{S}(x, y) = \mathcal{N}(\mathcal{T}(\mathcal{N}(x), \mathcal{N}(y)))$ for all x and y in $\mathcal{L}$. The mapping $\mathcal{I}_{\mathcal{S},\mathcal{N}}$ defined by $\mathcal{I}_{\mathcal{S},\mathcal{N}}(x, y) = \mathcal{S}(\mathcal{N}(x), y)$ is an implicator, usually called S-implicator (induced by $\mathcal{S}$ and $\mathcal{N}$). On the other hand, the mapping $\mathcal{I}_{\mathcal{T}}$ defined by $\mathcal{I}_{\mathcal{T}}(x, y) = \sup\{\lambda | \lambda \in \mathcal{L} \text{ and } \mathcal{T}(x, \lambda) \leq_{\mathcal{L}} y\}$ is an implicator, usually called the residual implicator or R-implicator (of $\mathcal{T}$).

[4] In the literature one will also find the notation $\bot$ and $\top$ to denote $0_{\mathcal{L}}$ and $1_{\mathcal{L}}$ respectively.

While the framework we will introduce to perform fuzzy answer set programming in Section 3 can be used in combination with any complete lattice, we will restrict ourselves for the examples in the current paper to the complete lattice $([0,1], \leq)$. The following example presents some fuzzy logical operators on this lattice.

Example 2. The mapping $\mathcal{N}_s$ defined as $\mathcal{N}_s(x) = 1 - x$ for all x in $[0,1]$ is called the standard negator. The t-norms $\mathcal{T}_{\mathrm{M}}$, $\mathcal{T}_{\mathrm{P}}$, and $\mathcal{T}_{\mathrm{W}}$ and their dual t-conorms $\mathcal{S}_{\mathrm{M}}$, $\mathcal{S}_{\mathrm{P}}$, and $\mathcal{S}_{\mathrm{W}}$ w.r.t. the standard negator, are defined as

$$\begin{array}{ll} \mathcal{T}_{\mathrm{M}}(x,y) = \min(x,y) & \mathcal{S}_{\mathrm{M}}(x,y) = \max(x,y) \\ \mathcal{T}_{\mathrm{P}}(x,y) = x \cdot y & \mathcal{S}_{\mathrm{P}}(x,y) = x + y - x \cdot y \\ \mathcal{T}_{\mathrm{W}}(x,y) = \max(x+y-1,0) & \mathcal{S}_{\mathrm{W}}(x,y) = \min(x+y,1) \end{array}$$

for all x and y in $[0,1]$. They induce the following implicators (the mappings on the right are R-implicators while those on the left are S-implicators; for ease of notation the inducing negator $\mathcal{N}_s$ has been omitted):

$$\begin{array}{ll} \mathcal{I}_{\mathcal{S}_{\mathrm{M}}}(x,y) = \max(1-x,y) & \mathcal{I}_{\mathcal{T}_{\mathrm{M}}}(x,y) = \begin{cases} 1, \text{ if } x \leq y \\ y, \text{ else} \end{cases} \\ \mathcal{I}_{\mathcal{S}_{\mathrm{P}}}(x,y) = 1 - x + x \cdot y & \mathcal{I}_{\mathcal{T}_{\mathrm{P}}}(x,y) = \begin{cases} 1, \text{ if } x \leq y \\ \frac{y}{x}, \text{ else} \end{cases} \\ \mathcal{I}_{\mathcal{S}_{\mathrm{W}}}(x,y) = \min(1-x+y,1) & \mathcal{I}_{\mathcal{T}_{\mathrm{W}}}(x,y) = \min(1-x+y,1) \end{array}$$

Every implicator induces a negator by defining $\mathcal{N}(x) = \mathcal{I}(x, 0_{\mathcal{L}})$. The above mentioned S-implicators induce the standard negator $\mathcal{N}_s$, while $\mathcal{I}_{\mathcal{T}_{\mathrm{M}}}$ and $\mathcal{I}_{\mathcal{T}_{\mathrm{P}}}$ induce the Gödel negator[5] $\mathcal{N}_g$ defined by $\mathcal{N}_g(x) = 1$ if $x = 0$ and $\mathcal{N}_g(x) = 0$ otherwise.

A fuzzy set in U is a $U \mapsto \mathcal{L}$ mapping. For fuzzy sets A and B in U, A is said to be included in B, denoted by $A \preceq_{\mathcal{L}} B$, iff $A(u) \leq_{\mathcal{L}} B(u)$ for all u in U. As usual, we have $A \prec_{\mathcal{L}} B$ iff $A \preceq_{\mathcal{L}} B$ and not $B \preceq_{\mathcal{L}} A$.

2.2 Answer Set Programming

We give some preliminaries concerning the answer set semantics for logic programs [3]. A *literal* is an atom a or a negated atom $\neg a$. For a set of literals X, we use $\neg X$ to denote $\{\neg l \mid l \in X\}$ where $\neg\neg a = a$. When $X \cap \neg X = \emptyset$ we say that X is *consistent*. An *extended literal* is a literal or a *naf-literal* of the form *not* l where l is a literal. The latter form denotes negation as failure. For a set of extended literals Y, we use Y^- to denote the set of ordinary literals underlying the naf-literals in Y, i.e. $Y^- = \{l \mid \mathit{not}\ l \in Y\}$. Further, we use *not* X to denote the set $\{\mathit{not}\ l \mid l \in X\}$. An extended literal l is true w.r.t. X, denoted $X \models l$ if $l \in X$ in case l is ordinary, or $a \notin X$ if $l = \mathit{not}\ a$ for some ordinary literal a. As usual, $X \models Y$ iff $\forall l \in Y \cdot X \models l$.

A *rule* is of the form $\alpha \leftarrow \beta$ where[6] $\alpha \cup \beta$ is a finite set of extended literals and $|\alpha| \leq 1$. Thus the *head* of a rule is either an extended literal or empty. A finite set of rules is called a *(logic) program*. The *Herbrand base* $\mathcal{B}_P$ of a program P contains all atoms appearing in P. The set of all literals that can be formed with the atoms in P, denoted by Lit_P, is defined by $Lit_P = \mathcal{B}_P \cup \neg\mathcal{B}_P$. Similarly, we define the set of all extended literals that can be formed with the atoms in P as $Elit_P = Lit_P \cup \mathit{not}\ Lit_P$. Any consistent subset $I \subseteq Lit_P$ is called an *interpretation* of P.

[5] This negator is also known in the literature as the Heyting negator.

[6] For simplicity, we assume that programs have already been grounded.

A rule $r = \alpha \leftarrow \beta$ is *satisfied* by an interpretation I, denoted $I \models r$, if $I \models \alpha$ and $\alpha \neq \emptyset$, whenever $I \models \beta$, i.e. if r is *applicable* ($I \models \beta$), then it must be *applied* ($I \models \alpha \cup \beta$ and $\alpha \neq \emptyset$). Note that this implies that a *constraint*, i.e. a rule with empty head ($\alpha = \emptyset$), can only be satisfied if it is not applicable ($I \not\models \beta$). For a program P, an interpretation I is called a *model* of P if $\forall r \in P \cdot I \models r$, i.e. I satisfies all rules in P. It is a minimal model of P if there is no model J of P such that $J \subset I$.

A *simple program* is a program without negation as failure. For simple programs P, we define an *answer set* of P as a minimal model of P. On the other hand, for a program P, i.e. a program containing negation as failure, we define the *GL-reduct* [14] for P w.r.t. I, denoted P^I, as the program consisting of those rules[7] $(\alpha \backslash \mathit{not}\, \alpha^-) \leftarrow (\beta \backslash \mathit{not}\, \beta^-)$ where $\alpha \leftarrow \beta$ is in P, $I \models \mathit{not}\, \beta^-$ and $I \models \alpha^-$. Note that all rules in P^I are free from negation as failure, i.e. P^I is a simple program. An interpretation I is then an *answer set* of P iff I is a minimal model of the GL-reduct P^I.

Example 3. Consider the program

$$r_1 : a \leftarrow \mathit{not}\ b \qquad r_2 : b \leftarrow \mathit{not}\ a$$

Clearly, both $\{a\}$ and $\{b\}$ are answer sets of this program as the GL-reducts $P^{\{a\}} = \{a \leftarrow\}$ and $P^{\{b\}} = \{b \leftarrow\}$ have $\{a\}$ and $\{b\}$ respectively as their minimal model. On the other hand, $\emptyset$ and $\{a, b\}$ are not answer sets. For the former interpretation, the reduct $P^{\emptyset} = \{a \leftarrow\ ;\ b \leftarrow\}$ has $\{a, b\}$ as its minimal model which differs from $\emptyset$, while the latter has an empty reduct, thus an empty minimal model, which differs from $\{a, b\}$.

3 Fuzzy Answer Set Programming

Classical ASP, as defined in the previous subsection, is in some ways a very strict framework in its semantics. In particular, an answer set is required to satisfy all rules of the program fully. In a more flexible setting, we wish to be able to deal with interpretations that satisfy rules possibly only to a certain extent. To this end, we allow literals to be true to a degree, as opposed to either being true or not true. As such, interpretations, and hence also answer sets, become fuzzy sets in Lit_P.

As the high configurability of fuzzy logic can be seen as one of its main strengths, we will adopt this behavior to the FASP framework presented in this section. Therefore, we allow a user to choose, in function of the application at hand, how the different classical operations need to be interpreted. More specifically, a user has to fix a complete lattice $\mathcal{L}$ first. Then, she has to choose two negators $\mathcal{N}_c$ and $\mathcal{N}_n$, which will be used to define consistency and the semantics of negation as failure respectively. Further, two t-norms $\mathcal{T}_c$ and $\mathcal{T}_a$ need to be fixed, respectively used for defining consistency and applicability of rules. Also an implicator $\mathcal{I}$ is needed to obtain the degree of satisfaction of a rule. Finally, an aggregator $\mathcal{A}$ is needed that combines all the degrees of satisfaction of rules into a single truth value denoting the degree in which a fuzzy interpretation is a fuzzy model. For the rest of this paper, we assume, without loss of generality, that the above choices have been made, and we will not repeat them everytime in the definitions, but just use them.

[7] As usual, $\backslash$ denotes set difference.

The first classical notions that need to be tackled are containment and consistency. In ASP a literal l is either true or false; and thus it is either contained in an interpretation or not. When both l and $\neg l$ are contained in an interpretation, it is said to be inconsistent. In a fuzzy context, a literal l can be a bit true, and both l and $\neg l$ can be a bit true in a consistent way, making a modified notion of consistency necessary.

Definition 1. *Let P be a program. A **fuzzy interpretation** I for P is a fuzzy set in Lit_P, i.e. a $I : Lit_P \mapsto \mathcal{L}$ mapping. I is called **x-consistent**, $x \in \mathcal{L}$, iff*

$$\mathcal{N}_c(\sup_{a \in \mathcal{B}_P} \mathcal{T}_c(I(a), I(\neg a))) \geq_{\mathcal{L}} x \ .$$

Intuitively, the definition of x-consistency allows a user to choose the point where the degree of containment of both l and $\neg l$ in a fuzzy interpretation I, makes that interpretation inconsistent. The classical notion of an interpretation emerges from the above definition for the lattice $\mathcal{L} = \{0_{\mathcal{L}}, 1_{\mathcal{L}}\}$. In this particular case, an interpretation I is called $1_{\mathcal{L}}$-consistent iff there does not exist an a in $\mathcal{B}_P$ such that both $I(a) = 1_{\mathcal{L}}$ and $I(\neg a) = 1_{\mathcal{L}}$.

As fuzzy interpretations only assign truth values to ordinary literals explicitly, we need a mechanism to retrieve truth values for naf-literals. While complementary literals l and $\neg l$ are only weakly related to each other using $\mathcal{N}_c$, $\mathcal{T}_c$, and a certain x-consistency boundary, naf-literals l and *not* l need a tighter connection since, intuitively, a naf-literal *not* l can only be true to the degree that the underlying ordinary literal l is false, and vice versa. Hence, we use $\mathcal{N}_n$ to extend a fuzzy interpretation I to cover naf-literals by defining $I(\textit{not } l) = \mathcal{N}_n(I(l))$ for each $l \in Lit_P$.

Having fuzzy interpretations and x-consistency, we need to redefine the satisfaction of rules. While a rule in ASP is either satisfied or not, in a more flexible setting we should allow a rule to be partially (to a certain degree) satisfied. Further, each rule does not have to be satisfied to the same degree, which is, e.g., useful in applications having preferences among rules. To obtain these degrees, we use $\mathcal{T}_a$ and $\mathcal{I}$ to induce, for a fuzzy interpretation I, a satisfaction function $I_{\models}$ that assigns a truth value to the bodies of rules and to the rules themselves. Later on, this satisfaction function will be used, in combination with the aggregator $\mathcal{A}$, to obtain the degree in which a fuzzy interpretation is a model of a program.

Definition 2. *Let P be a program and let I be a fuzzy interpretation. The induced satisfaction function $I_{\models} : 2^{Elit_P} \cup P \mapsto \mathcal{L}$ is defined by*

$$\begin{aligned} I_{\models}(\emptyset) &= 1_{\mathcal{L}} \\ I_{\models}(\{l\} \cup \beta) &= \mathcal{T}_a(I(l), I_{\models}(\beta)) \\ I_{\models}(\ \leftarrow \beta) &= \mathcal{I}(I_{\models}(\beta), 0_{\mathcal{L}}) \\ I_{\models}(l \leftarrow \beta) &= \mathcal{I}(I_{\models}(\beta), I(l)) \end{aligned}$$

Note that $I_{\models}(\{l\}) = I(l)$ and $I_{\models}(\textit{not } l) = \mathcal{N}_n(I(l))$. Intuitively, $I_{\models}(s)$, with $s \in P$, defines to which degree a rule s is satisfied taking into account the truth assignments of the head and body of s in I. To define a fuzzy model, the different $I_{\models}(s)$, with $s \in P$, need to be accumulated in some way. The user defined aggregator $\mathcal{A}$, which takes as

input a program and a satisfaction function, will accomplish this job and result in a truth value denoting the degree in which the fuzzy interpretation I is a model of P. However, we demand that an aggregator is increasing whenever the degrees of satisfaction of the rules increase.

Definition 3. *Let P be a program and let I be an x-consistent fuzzy interpretation. Then, I is an x-consistent **fuzzy** y-**model** of P, $y \in \mathcal{L}$, iff $\mathcal{A}(P, I_{\models}) \geq y$.*

Example 4. Consider the lattice $\mathcal{L} = [0, 1]$ and the program

$$r_1 : a \leftarrow \mathit{not}\ b \qquad r_2 : b \leftarrow \mathit{not}\ a \qquad r_3 : c \leftarrow a$$

and consider the fuzzy interpretations[8] $K = \{(a, 0.9), (b, 0.3), (c, 0.2)\}$ and $L = \{(a, 0.4), (b, 0.7), (c, 0.8)\}$. Both of these fuzzy interpretations are 1-consistent, independently of the choices for $\mathcal{N}_c$ and $\mathcal{T}_c$. For negation as failure, we use the negator $\mathcal{N}_s$. To compute the satisfaction of the rules, we use the implicator $\mathcal{I}_{\mathcal{S}_M}$. Finally, as an aggregator we use $\mathcal{A}(P, I_{\models}) = \inf\{I_{\models}(s) \mid s \in P\}$, i.e. the weakest rule dominates the solution.

We have $K_{\models}(r_1) = max(1 - K(\mathit{not}\ b), K(a)) = max(1 - \mathcal{N}_s(K(b)), K(a)) = max(1 - (1 - 0.3), 0.9) = 0.9$. Similarly, $K_{\models}(r_2) = max(1 - (1 - 0.9), 0.3) = 0.9$ and $K_{\models}(r_3)$ = $max(1 - 0.9, 0.2)$ = 0.2. As a result, K is a 1-consistent 0.2-model of P. On the other hand, one can verify that $L_{\models}(r_1) = L_{\models}(r_2) = 0.7$ and $L_{\models}(r_3) = 0.8$, yielding that L is a 1-consistent 0.7-model of P.

The above definitions are conservative extensions of classical principles, i.e. the classical definitions are special cases of the ones presented here. Hence it is not surprising that the extensions suffer the same difficulties when used to define a fuzzy answer set semantics. For instance, both $I = \{(a, 0_{\mathcal{L}}), (b, 0_{\mathcal{L}})\}$ and $J = \{(a, 1_{\mathcal{L}}), (b, 1_{\mathcal{L}})\}$ are "perfect" fuzzy interpretations of the program $\{a \leftarrow b\ ;\ b \leftarrow a\}$ as they both satisfy all rules to a maximal degree $1_{\mathcal{L}}$. In traditional ASP, the set $\{a, b\}$ is called "unfounded"[23] and answer sets should be free of such sets. This is achieved by imposing a minimality requirement.

Definition 4. *Let P be a program. An x-consistent y-model M is an x-consistent **minimal fuzzy** y-**model** iff M is $\prec_{\mathcal{L}}$ minimal among all x-consistent fuzzy y-models of P.*

Applied to the examples I and J above, this results in $I \prec_{\mathcal{L}} J$, yielding that I is the single $1_{\mathcal{L}}$-consistent minimal fuzzy $1_{\mathcal{L}}$-model of the two rules.

Example 5. Reconsider the program and the choices for logical operators from Example 4. One can check that the fuzzy interpretations $M = \{(a, 0.9), (b, 0.8), (c, 1)\}$ and $N = \{(a, 0.9), (b, 0.2), (c, 1)\}$ are both 1-consistent 0.9-models of P. However, one can verify that $N \prec_{\mathcal{L}} M$, which fits our intuition as the degree in which b is assumed true is overestimated in M. Still, N is not minimal, as one can verify that $S = \{(a, 0.9), (c, 0.9)\}$ is $\prec_{\mathcal{L}}$-minimal[9], i.e. a 1-consistent minimal fuzzy 0.9-model of P.

[8] As usual, a fuzzy set I in Lit_P is denoted as $\{(l, x) \mid I(l) = x \wedge l \in Lit_P\}$, omitting the literals $(l, 0_{\mathcal{L}})$.

[9] Note that when another implicator is chosen, S not necessarily remains a minimal fuzzy 0.9-model of P. E.g., using $\mathcal{I}_{\mathcal{T}_M}$ would make S only a fuzzy 0-model.

While the above minimization process is necessary, it does not yet suffice to prevent unwanted models, as witnessed by the following example.

Example 6. Consider the program

$$r_1 : a \leftarrow \qquad\qquad r_2 : b \leftarrow a, \mathit{not}\ c$$

and the fuzzy interpretations $K = \{(a, 0.9), (b, 0.9)\}$ and $L = \{(a, 0.9), (c, 0.9)\}$. We make the same choice for the logical operators as in Example 4. To evaluate the body of r_2 we use $\mathcal{T}_a = \mathcal{T}_{\mathrm{M}}$. One can verify that K and L are both minimal fuzzy 0.9-models. However, intuitively L is not acceptable as a good solution as there is no support for accepting c at degree 0.9, i.e. there is no applicable rule with c in the head.

In traditional ASP, the above problem is solved by taking the GL-reduct which will remove, for $I = \{a, c\}$, the rule r_2 from the reduct P^I, because r_2 is not applicable due to the *not* c literal in its body. Now, the minimal model of this reduct does not equal I, hence, it is rejected as an answer set. Note that the removal of a rule does not mean that this rule does not have to be satisfied anymore. On the contrary, in the example above, rule r_2 is removed because it is not applicable under interpretation I, hence it is satisfied by default, independently of the truth value of b.

Note that in traditional ASP, there are two possible scenarios for a model I to satisfy a rule of the form $l \leftarrow \beta$. Either it is applicable ($I \models \beta$), hence l must assume the truth value $1_{\mathcal{L}}$ to satisfy the rule, or it is unapplicable ($I \not\models \beta$), hence the rule is satisfied by default and l can assume any truth value from the lattice $\mathcal{L} = \{0_{\mathcal{L}}, 1_{\mathcal{L}}\}$. In the first case, the truth value of l is fully determined by the rule, while in the latter case, the rule does not impose any restrictions on the truth value of l, hence taking it into account does not influence the result and we can remove the rule.

In FASP, such a removal strategy for naf-literals is not feasible as such literals may be true only to a certain degree, making the bodies of some rules applicable to a certain degree, which requires that they also need to be applied to a certain degree. Hence, as opposed to either fully determining the truth value of the head of a rule (full information), or leaving it completely arbitrary (no information), in FASP a rule may also carry *some* information that delimits the set of possible truth values that can be assumed by the head.

Thus, we define for each rule in the program a subset $Y \subseteq \mathcal{L}$ such that none of the values in Y lowers the degree of satisfaction of the rule. Next, for a literal $l \in Lit_P$, we consider these sets Y for each rule of the form $l \leftarrow \beta$. By taking the intersection of these sets, we obtain a range of truth values. Choosing an alternative truth value for l within this range does not lower the degrees of satisfaction of the rules with l in the head. However, interpretations that choose the lower values in the range are called better supported.

Definition 5. *Let P be a program and let I be a fuzzy interpretation. The* ***supportedness function*** *I_s associated with I is defined by*

$$I_s(l) = \bigcap_{\{l\} \leftarrow \beta \in P} \{y \in \mathcal{L} \mid \mathcal{I}(I_{\models}(\beta), y) \geq_{\mathcal{L}} I_{\models}(\{l\} \leftarrow \beta)\} \ ,$$

for each $l \in Lit_P$, where, by definition, $\bigcap \emptyset = \{0_{\mathcal{L}}\}$. A minimal x-consistent fuzzy y-model of P is called an x-consistent ***fuzzy y-answer set*** *iff we have for each $l \in Lit_P$ that $I(l) = \inf(I_s(l))$.*

Example 7. Reconsider Example 6. Clearly, $L_s(c) = \{0\}$, yielding that L is not a fuzzy 0.9-answer set of P. On the other hand, one can verify that $K_s(a) = K_s(b) = [0.9, 1]$ and $K_s(c) = \{0\}$, implying that K is a fuzzy 0.9-answer set of P.

Proposition 1. *For a simple program P, I is a minimal x-consistent fuzzy y-model of P iff I is an x-consistent fuzzy y-answer set of P.*

Example 8. Reconsider Example 1 from the introduction. We are interested to know to what degrees we have to practice the various sports such that an acceptable degree, e.g. 0.7, of full-body exercise is obtained. Since our main concern is a satisfactory degree of full-body exercise, we will use an aggregator that gives more importance to the constraint rule r_7. An appropriate choice could be an aggregator that only takes r_7 into account. In this case a fuzzy interpretation is a model to the degree to which it satisfies r_7. Of course we also require the model to be minimal and supported, which is where other rules come into play.

Further, we will also use $\mathcal{T}_a = \mathcal{T}_{\mathrm{M}}$ to evaluate the body of rules, and the implicator $\mathcal{I} = \mathcal{I}_{\mathcal{T}_W}$ to evaluate the satisfaction of the rules. A fuzzy 0.7-answer set K for the above program must at least satisfy $K_{\models}(r_7) = 0.7$. This yields that

$$min(1 - K(\textit{not } full\_body\_exercise) + 0, 1) = 0.7 \ ,$$

which implies that $K(\textit{not } full\_body\_exercise) = 0.3$, and thus, using $\mathcal{N}_s$ for negation as failure, that $K(full\_body\_exercise) = 0.7$. To have support for the literal, i.e. $\inf(K_s(full\_body\_exercise)) = 0.7$, one of the two rules r_5 or r_6 have to be made applicable to a certain degree, in turn implying that some of the four sports will have to be exercised in a higher degree than others to achieve that sufficient degree of applicability of r_5 or r_6[10]. One can verify that

$$K = \{(lift\_weights, 0.8), (swim, 0.2), (run, 0.7), (play\_ball, 0.3), \\ (full\_body\_exercise, 0.7)\} \ ,$$

is a fuzzy 0.7-answer set of the above program.

Intuitively, this solution is acceptable as it describes a configuration where two sports, which are together in rule r_5, are assigned a higher degree than there complementary variants, and due to this choice we have support for full-body exercise up to a degree of 0.7.

On the other hand, one can check that for the fuzzy interpretation

$$L = \{(lift\_weights, 0.8), (swim, 0.2), (run, 0.3), (play\_ball, 0.7), \\ (full\_body\_exercise, 0.7)\} \ ,$$

it turns out that $L_s(full\_body\_exercise) = [0.3, 1] \cap [0.2, 1] = [0.3, 1]$. Hence, L is not a fuzzy 0.7-answer set of the program, fitting our intuition.

[10] Note that this example also illustrates how the proposed framework can be used to do fuzzy diagnostic reasoning. The constraint r_7 can be seen as an encoding of the observations, r_5 and r_6 represent the system description, while r_1, r_2, r_3 and r_4 provide the explanations.

4 Retrieving Classical Answer Sets

The FASP framework presented in the previous section turns out to be a proper generalisation of the classical answer set programming paradigm with the notions of fuzzy logic. First of all, ASP can be retrieved as a special case of FASP by choosing the truth lattice $\mathcal{L} = \{0_\mathcal{L}, 1_\mathcal{L}\}$.

Proposition 2. *Consider a program P and let $\mathcal{L}$ be the lattice $\{0_\mathcal{L}, 1_\mathcal{L}\}$. Furthermore let the aggregator $\mathcal{A}$ be such that $\mathcal{A}(P, I_\models) = 1_\mathcal{L}$ iff $I_\models(s) = 1_\mathcal{L}$ for every rule $s \in P$. An interpretation M is an answer set of P iff the fuzzy interpretation f_M, with $f_M(l) = 1_\mathcal{L}$ if $l \in M$ and $f_M(l) = 0_\mathcal{L}$ otherwise, is a $1_\mathcal{L}$-consistent fuzzy $1_\mathcal{L}$-answer set of P.*

Note that $1_\mathcal{L}$-consistency is needed to forbid (classical) contradictions, and the restriction to fuzzy $1_\mathcal{L}$-answer sets is mandated by the need to classically satisfy all rules and have the foundedness property of answer sets.

Example 9. Reconsider the program from Example 3. The empty set is not a $1_\mathcal{L}$-model of this program as it satisfies neither of the rules to degree $1_\mathcal{L}$. $K = \{(a, 1_\mathcal{L})\}$, $L = \{(b, 1_\mathcal{L})\}$, and $M = \{(a, 1_\mathcal{L}), (b, 1_\mathcal{L})\}$ are $1_\mathcal{L}$-models, but the latter is obviously not minimal. One can verify that $K_s(a) = \{1_\mathcal{L}\}$ and $K_s(b) = \{0_\mathcal{L}, 1_\mathcal{L}\}$, and similarly that $L_s(a) = \{0_\mathcal{L}, 1_\mathcal{L}\}$ and $L_s(b) = \{1_\mathcal{L}\}$, in other words both K and L are 1-consistent fuzzy 1-answer sets.

In the proposition above, no choice for $\mathcal{N}_c$, $\mathcal{N}_n$, $\mathcal{T}_c$, $\mathcal{T}_a$, and $\mathcal{I}$ is specified as all negators, t-norms and implicators on $\{0_\mathcal{L}, 1_\mathcal{L}\}$ coincide. However, when we allow for intermediate truth values, a choice for logical operators opens up. Below we argue that certain choices are more "answer set behaved" than others.

Classical answer sets cannot contain both a and $\neg a$. If one wants to preserve this behaviour for fuzzy answer sets, i.e. such that a $1_\mathcal{L}$-consistent fuzzy answer set can not contain a and $\neg a$ simultaneously, not even to some degree, $\mathcal{T}_c$ should be chosen with care. E.g., on $\mathcal{L} = [0, 1]$, take $\mathcal{T}_c = \mathcal{T}_\mathrm{W}$ and consider the fuzzy interpretation $I = \{(a, 0.4), (\neg a, 0.4)\}$. Then, $\mathcal{T}_c(I(a), I(\neg a)) = max(0.4 + 0.4 - 1, 0) = 0$. For this t-norm it holds, in general, that $\mathcal{T}_c(I(a), I(\neg a)) = 0$ iff $I(a) + I(\neg a) \leq 1$, which certainly does not correspond to a classical answer set semantics. However, there exist some stronger versions for $\mathcal{T}_c$ that do not suffer from this problem, i.e. for which $\mathcal{T}_c(I(a), I(\neg a)) = 0$ iff $I(a) = 0$ or $I(\neg a) = 0$. Both $\mathcal{T}_M$ and $\mathcal{T}_P$ are such t-norms, and can be used to retrieve fuzzy answer sets with a classical ASP consistency notion.

Next, we consider the possible choices for the implicator. By definition, an implicator satisfies $\mathcal{I}(0_\mathcal{L}, 0_\mathcal{L}) = \mathcal{I}(0_\mathcal{L}, 1_\mathcal{L}) = \mathcal{I}(1_\mathcal{L}, 1_\mathcal{L}) = 1_\mathcal{L}$ and $\mathcal{I}(1_\mathcal{L}, 0_\mathcal{L}) = 0_\mathcal{L}$, which implies that any choice for $\mathcal{I}$ is sufficient to retrieve classical answer sets from the $1_\mathcal{L}$-consistent fuzzy $1_\mathcal{L}$-answer sets. However, when intermediate truth values are considered, certain choices for $\mathcal{I}$ are more answer set alike, as witnessed by the following example.

Example 10. Reconsider the program from Example 3 and let $J = \{(a, 0.6), (b, 0.4)\}$ be a fuzzy interpretation. When using $\mathcal{I}_{S_M}$, we get $J_\models(r_1) = max(1 - 0.6, 0.6) = 0.6$ and $J_\models(r_2) = max(1 - 0.4, 0.4) = 0.6$, yielding that J will be at most a 0.6-answer

set. However, in a classical answer set context this looks a bit unintuitive as the heads of both rules are satisfied to exactly the same degrees as to which their bodies are applicable, and thus intuitively the rules should be totally satisfied. Applying $\mathcal{I}_{T_M}$ on the program yields $J_{\models}(r_1) = J_{\models}(r_2) = 1$, which fits that intuition.

The implicator $\mathcal{I}_{T_M}$ belongs to the class of R-implicators, for which, in general, it holds that $\mathcal{I}(x, y) = 1_{\mathcal{L}}$ whenever $x \leq_{\mathcal{L}} y$. All R-implicators in Example 2 satisfy the residuation principle or adjoint condition, i.e. $\mathcal{T}(x, y) \leq_{\mathcal{L}} z$ iff $x \leq \mathcal{I}_{\mathcal{T}}(y, z)$ for all x, y, and z in $\mathcal{L}$. Note that for such implicators the degree of satisfaction of a rule $l \leftarrow \beta$ does not go down as long as the truth value of the head is greater than or equal to $\mathcal{T}(I_{\models}(l \leftarrow \beta), I_{\models}(\beta))$. In other words, in this case we obtain a more direct expression for y to be used in Definition 5.

Finally, to preserve classical answer set semantics, an interpretation should be said to satisfy a program to degree $1_{\mathcal{L}}$ iff it satisfies all rules of the program to degree $1_{\mathcal{L}}$. The aggregator $\mathcal{A}(P, I_{\models}) = \inf\{I_{\models}(s) \mid s \in P\}$ we introduced before satisfies this condition and can be used to retrieve classical answer sets.

5 Related work

Logic programming in the presence of uncertainty or imprecision has received a considerable amount of attention (see e.g. [1, 9] for overviews). It is however interesting to observe that the well known existing frameworks, including those that consider fuzzy interpretations, hold on to two valued concepts of rule satisfaction, model, etc. in the sense that a fuzzy interpretation satisfies a rule or not, it is a model or not, etc. This clearly sets them apart from the approach introduced in this paper.

The enrichment of ASP with concepts from fuzzy logic as well as from the closely related possibilistic logic [11] has been studied from various angles already. In annotated answer set programming [22], a rule is of the form $l\{f(z_1, z_2, \ldots, z_n)\} \leftarrow l_1\{z_1\}, l_2\{z_2\}, \ldots, l_n\{z_n\}$ where $l, l_1, l_2, \ldots, l_n$ denote literals and $z_1, z_2, \ldots, z_n$ are annotation terms that can be understood as truth degrees. Such a rule asserts that l is true at least to degree $f(z_1, z_2, \ldots, z_n)$ whenever l_i is true at least to degree z_i (for $i = 1 \ldots n$). Because of this early revertment to the two valued case, no fuzzy logical operators are needed in this approach.

The approach in [17] adheres closer to ours. A rule of the form $\alpha \overset{z}{\leftarrow} \beta$ is said to be satisfied by the interpretation iff (in our notation) $I_{\models}(\alpha) \geq_{\mathcal{L}} \mathcal{T}(I_{\models}(\beta), z)$. The residuation principle reveals a clear connection with our approach when committing to an R-implicator $\mathcal{I}_{\mathcal{T}}$: namely that I satisfies the rule $\alpha \overset{z}{\leftarrow} \beta$ according to [17] iff I satisfies this rule at least to degree z in our approach. This is also in accordance with [8] where the use of adjoint pairs $(\mathcal{T}, \mathcal{I}_{\mathcal{T}})$ is strongly advocated to preserve important theoretical results. Being able to impose specific satisfaction requirements for individual rules is in general an interesting feature, e.g., when rules and facts originate from different knowledge bases that are not all equally trusted. Note that this can be easily incorporated in our approach by choosing a suitable aggregator $\mathcal{A}$.

In a similar way, [25] can be seen as a special case of the FASP framework presented in this paper as [25] commits itself, with limited motivation, to very specific choices for

the user-selectable operators on the lattice $[0, 1]$. Some of these choices are at least questionable. E.g., using the Gödel negator $\mathcal{N}_g$ for interpreting naf yields that a rule $a \leftarrow$ *not* b will not be applied in any way although b is only true to a small degree, e.g. 0.1. Using the standard negator $\mathcal{N}_s$, as we do in our examples, this would yield a rule that is applicable to a degree 0.9, and, if a rule satisfaction of at least 0.8 is wanted with e.g. $\mathcal{I}_{S_M}$, we have $max(0.1, y) = 0.8$, which implies that a will be derived at degree 0.8 in a fuzzy answer set.

A possibilistic definite logic program [18] consists of rules annotated with certainty degrees. These degrees are used to establish a possibility distribution on the universe of atom sets, from which a possibilistic model is derived. The authors choose implicitly for the Gödel negator, as they first compute the classical answer sets, and afterwards compute, for an answer set S, the possibility to which each literal l is contained in S.

6 Conclusions and future research

There are many ways to increase the expressive power of answer set programming (ASP) by enriching it with mechanisms to deal with imprecision and uncertainty. In this paper we presented a general and elegant fuzzification of ASP, called fuzzy answer set programming (FASP). The generality is reflected in a high configurability by the user, which allows the system to be tailored to the application at hand. Among other things, the ability to choose an aggregator allows for future extensions of the semantics, e.g. incorporating rule preferences on fuzzy programs. The elegance is due to a close adherence to both the fuzzy logic and the answer set programming paradigm: as opposed to other approaches, FASP does not revert soon to the two valued case but instead allows to compute the actual degree to which a fuzzy interpretation is an answer set. Furthermore we have shown that FASP extends the traditional answer set semantics.

Clearly, there are a lot of topics that still need to be investigated, e.g. a fixpoint characterization, the complexity of the semantics, the use of disjunction etc., all parametrized by the choice of the (lattice) operations. In addition, we intend to explore natural FASP applications areas such as “web of trust”, diagnosis, and decisision support.

References

[1] T. Alsinet, L. Godo, and S. Sandri. Two formalisms of extended possibilistic logic programming with context-dependent fuzzy unification: a comparative description. *Electronic Notes in Theoretical Computer Science*, 66(5), 2002.

[2] M. Balduccini and M. Gelfond. Logic programs with consistency-restoring rules. In *Proceedings of the International Symposium on Logical Formalization of Commonsense Reasoning*, AAAI 2003 Spring Symposium Series, 2003.

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

[4] G. Birkhoff. Lattice theory. *American Mathematical Society Colloquium Publications*, 25(3), 1967.

[5] G. Brewka. Logic programming with ordered disjunction. In *Proceedings of the 18th National Conference on Artificial Intelligence and Fourteenth Conference on Innovative Applications of Artificial Intelligence*, pages 100–105. AAAI Press, July 2002.

[6] G. Brewka and T. Eiter. Preferred answer sets for extended logic programs. *Artificial Intelligence*, 109(1-2):297–356, April 1999.
[7] F. Buccafurri, N. Leone, and P. Rullo. Strong and weak constraints in disjunctive datalog. In *Proc. of the 4th Intl. Conf. on Logic Programming (LPNMR '97)*, pages 2–17, 1997.
[8] C. Damasio, J. Medina, and M. Ojeda-Aciego. Sorted multi-adjoint logic programs: termination results and applications. *Journal of Applied Logic*, page To appear, 2006.
[9] C. V. Damasio and L. M. Pereira. Sorted monotonic logic programs and their embedding. In *Proc. of the 10th Intl. Conf. on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU-04)*, pages 807–814, 2004.
[10] M. De Vos and D. Vermeir. On the Role of Negation in Choice Logic Programs. In *Logic Programming and Non-Monotonic Reasoning Conference (LPNMR'99)*, volume 1730 of *LNAI*, pages 236–246. Springer, 1999.
[11] D. Dubois and H. Prade. Possibilistic logic: a retrospective and prospective view. *Fuzzy Sets and Systems*, 144(1):3–23, 2004.
[12] T. Eiter, W. Faber, N. Leone, and G. Pfeifer. The diagnosis frontend of the dlv system. *AI Communications*, 12(1-2):99–111, 1999.
[13] D. Gabbay, E. Laenens, and D. Vermeir. Credulous vs. Sceptical Semantics for Ordered Logic Programs. In *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning*, pages 208–217. Morgan Kaufmann, 1991.
[14] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Logic Programming, Proceedings of the Fifth International Conference and Symposium*, pages 1070–1080, Seattle, Washington, August 1988. The MIT Press.
[15] M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9(3-4):365–386, 1991.
[16] D. N. Juergen Dix, Ugur Kuter. Planning in answer set programming using ordered task decomposition. In *Proc. of the 27th German Annual Conf. on Artificial Intelligence (KI '03)*, volume 2821 of *LNAI*, pages 490–504. Springer, 2003.
[17] C. Mateis. Extending disjunctive logic programming by t-norms. In *Proc. of the 5th Intl. Conf. on Logic Programming and Nonmonotonic Reasoning (LPNMR99)*, volume 1730 of *LNAI*, pages 290–304. Springer, 1999.
[18] P. Nicolas, L. Garcia, and I. Stéphan. Possibilistic stable models. In *Proc. of the 19th Intl. Joint Conf. on Artificial Intelligence*, pages 248–253, 2005.
[19] M. Nogueira, M. Balduccini, M. Gelfond, R. Watson, and M. Barry. An a-prolog decision support system for the space shuttle. In *Third International Symposium on Practical Aspects of Declarative Languages*, volume 1990 of *LNCS*, pages 169–183. Springer, 2001.
[20] V. Novák, I. Perfilieva, and J. Močkoř. *Mathematical Principles of Fuzzy Logic*. Kluwer Academic Publishers, 1999.
[21] T. Soininen and I. Niemelä. Developing a declarative rule language for applications in product configuration. In *Proc. of the 1st Intl. Workshop on Practical Aspects of Declarative Languages (PADL '99)*, volume 1551 of *LNCS*, pages 305–319. Springer, 1999.
[22] U. Straccia. Annotated answer set programming. In *Proc. of the 11th Intl. Conf. on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU-06)*, 2006.
[23] A. van Gelder, K. A. Ross, and J. S. Schlipf. The well-founded semantics for general logic programs. *Journal of the Association for Computing Machinery*, 38(3):620–650, 1991.
[24] D. Van Nieuwenborgh and D. Vermeir. Preferred answer sets for ordered logic programs. *Theory and Practice of Logic Programming*, 6(1-2):107–167, 2006.
[25] G. Wagner. A logical reconstruction of fuzzy inference in databases and logic programs. In *Proceedings of the International Fuzzy Set Association World Congress (IFSA'97)*, 1997.
[26] L. Zadeh. Fuzzy logic and approximate reasoning. *Synthese 30*, pages 407–428, 1975.