Fuzzy Answer Set Programming: An Introduction

Marjon Blondeel, Steven Schockaert, Martine De Cock, and Dirk Vermeir

Abstract In this chapter, we present a tutorial about fuzzy answer set programming (FASP); we give a gentle introduction to its basic ideas and definitions. FASP is a combination of answer set programming and fuzzy logics which has recently been proposed. From the answer set semantics, FASP inherits the declarative nonmonotonic reasoning capabilities, while fuzzy logic adds the power to model continuous problems. FASP can be tailored towards different applications since fuzzy logics gives a great flexibility, e.g. by the possibility to use different generalizations of the classical connectives. In this chapter, we consider a rather general form of FASP programs; the connectives can in principal be interpreted by arbitrary $[0,1]^n \to [0,1]$-mappings. Despite that very general connectives are allowed, the presented framework turns out to be an intuitive extension of answer set programming.

1 Introduction

In this chapter we will present and illustrate the basic definitions of fuzzy answer set programming (FASP). In recent years a variety of approaches to FASP have been

Marjon Blondeel
Vrije Universiteit Brussel, Department of Computer Science, Pleinlaan 2, 1050 Brussel, Belgium, e-mail: mblondee@vub.ac.be

Steven Schockaert
Cardiff University, School of Computer Science and Informatics, 5 The Parade, Cardiff, CF24 3AA, UK, e-mail: S.Schockaert@cs.cardiff.ac.uk

Martine De Cock
Universiteit Gent, Department of Applied Mathematics and Computer Science, Krijgslaan 281 (S9), 9000 Gent, Belgium, e-mail: Martine.DeCock@UGent.be

Dirk Vermeir
Vrije Universiteit Brussel, Department of Computer Science, Pleinlaan 2, 1050 Brussel, Belgium, e-mail: dvermeir@vub.ac.be

proposed (e.g. [5], [10], [12], [21]). This framework is a generalization of answer set programming using fuzzy logics, a class of logics whose semantics are based on truth values taken from the unit interval $[0,1]$ [9]. Answer set programming (ASP) [1] is a tool for modeling combinatorial search problems in a declarative way. It has its roots in logic programming and nonmonotonic reasoning.

Nonmonotonicity enables human-like reasoning; humans constantly revise their knowledge when they obtain new information. In contrast, classical logic works monotonically; when new knowledge is added, the set of conclusions that can be inferred increases. To overcome this limitation of classical logic when imitating human reasoning, several nonmonotonic logics, e.g. autoepistemic logic [14] and default logic [17], and logic programming with negation-as-failure such as ASP [7] have been proposed. In logic programming, and ASP in particular, nonmonotonicty is obtained by the negation-as-failure operator "not". The difference with classical negation $\neg$ is that $\neg a$ is true if we can derive $\neg a$, whereas $\text{not}\, a$ is true is we fail to derive a. Note that this means that ASP can deal with incomplete information; one can draw conclusions if information is absent.

The basic idea of ASP is that a search problem is translated into an ASP program, i.e. a set of rules of the form $\alpha \leftarrow \beta$. Such a rule indicates that whenever the body β holds, the head α holds as well. The expression α is a disjunction of literals and β a conjunction of extended literals. A literal is an atom or an expression of the form $\neg a$ with a an atom. An extended literal is a literal or an expression of the form $\text{not}\, l$ with l a literal. Given an ASP program, the idea is to find a minimal set of literals that can be derived from the program. A program can have several of such "answer sets" or none at all. The answer sets then correspond to the solutions of the original search problem. Let us consider a concrete example. Suppose one wants to color the vertices of a graph in either black or white but adjacent nodes must be colored differently. This search problem can be modeled by the program P:

$$\begin{aligned}
\text{black}(X) &\leftarrow \text{not}\,\text{white}(X)\\
\text{white}(X) &\leftarrow \text{not}\,\text{black}(X)\\
&\leftarrow \text{edge}(X,Y) \wedge \text{white}(X) \wedge \text{white}(Y)\\
&\leftarrow \text{edge}(X,Y) \wedge \text{black}(X) \wedge \text{black}(Y)
\end{aligned}$$

The first two rules express that each node should have exactly one of the two colors. The last two rules are constraints expressing that two nodes connected by an edge should have a different color. The empty head of a constraint can be thought of as always being "false". Hence, a constraint rule only holds if its body is false as well. Note that we use variables X and Y; this is to allow a compact description of the problem. By grounding the program, i.e. replacing the variables in all meaningful ways, one gets all the rules. For instance, for a graph with nodes a and b, the first rule from the program P above, gives rise to the grounded rules "$\text{black}(a) \leftarrow \text{not}\,\text{white}(a)$" and "$\text{black}(b) \leftarrow \text{not}\,\text{white}(b)$". In addition, a number of facts, rules of the form "$\text{edge}(a,b) \leftarrow$" with a and b nodes, are added to the program. The empty body of a fact can be thought of as always being "true". Hence, such a rule implies that there is an edge between node a and b, since a rule with a body that is true can only hold if the head is true as well. After grounding the program, the answer set semantics defines the solutions to the program. For instance, if there are three nodes a, b and c such that there is an edge between a and b and one between b and c, then there are two answer sets. One of these will contain the atoms black(a), white(b) and black(c) and the other the atoms white(a), black(b) and white(c).

Unfortunately, ASP is not suitable for expressing continuous optimization problems since it is limited to expressing problems in boolean logic. For example, suppose one wants to travel by car from one city to another in Winter. The driving time that is needed to do this depends on several factors; for instance the amount of snow, the distance and the traffic. These concepts are a matter of degree rather than boolean properties, thus we cannot directly use ASP to model this problem. One solution to this problem is to allow propositions to be true to a certain degree in $[0,1]$ and to generalize the syntax and semantics of ASP using fuzzy logics. We can then write the rule

$$\text{driving time } \leftarrow f(\text{snow}, \text{distance}, \text{traffic})$$

where “driving time”, “snow” and “traffic” now have to be seen as atoms that can be assigned a degree in $[0,1]$. The function f defines how these degrees have to be combined to give the driving time. Note that it is not realistic to assume that f is a linear function. For example, if it starts to snow, not even taking into account the other factors, then the driving time will increase very fast; after that the increase of driving time due to the snow will slow down. In practice, one can use statistical information to define f. Finally, remark that FASP is used to deal with partial truth and not with uncertainty or vagueness. See [6] for a discussion on the difference between these concepts. To deal with uncertainty, among others, ASP can be extended with possibility theory (e.g. [15]) or with probability theory (e.g. [3]).

The basic idea of FASP is to model search problems with continuous domains. A continuous search problem can then be translated into a FASP program, i.e. a set of rules of the form $\alpha \leftarrow \beta$ where α and β are built from atoms, expressions of the form not$a$ with a an atom, constants and connectives that can in principal be interpreted by arbitrary $[0,1]^n \to [0,1]$-mappings. Such a rule now intuitively means that the truth degree of α must be greater or equal to the truth degree of β. Reconsider our example about the driving time. The rule

$$\text{snow} \leftarrow 0.2$$

can be used to indicate that it snows to *at least* degree 0.2 which could mean, depending on how you define the degree of snow, that the snow melts immediately when it touches the ground. This rule is thus satisfied if it snows to degree 0.9. However this attaches a higher value to “snow” than the rule actually supports. If the degree of “snow” does not depend on other atoms, it is reasonable to attach the degree 0.2 to “snow”. This is in line with the idea of ASP which attempts to make as few literals true as possible to satisfy the rules of a program. Hence here we are interested in finding the lowest truth degrees that we can assign to each of the atoms, such that the rules are still satisfied. Although α and β may be built from very general connectives, FASP can model search problems entirely similar as ASP does for search problems with discrete domains.

Although it has been studied by several authors, FASP is by far not as developed as ASP. For example, very little is known about its computational complexity and few techniques are known to compute the answer sets of FASP programs. Also, many extensions proposed for ASP have not yet been considered in FASP. With the exceptions of e.g. [12], [18] and [20], most work is even restricted to FASP programs with exactly one atom in the head.

In the following section, we will give the necessary background on fuzzy logics followed by an introduction to FASP in Section 3. We present a motivating real life example in Section 4 and some remarks and open problems about FASP in Section 5.

2 Background on fuzzy logics

Fuzzy logics [9] form a class of logics whose semantics are based on truth degrees taken from the unit interval $[0,1]$. We will consider general fuzzy logics whose formulas are built from a set of atoms A, the truth constants in $[0,1]\cap\mathbb{Q}$ and arbitrary n-ary connectives for each $n\in\mathbb{N}$. A *fuzzy interpretation* is a mapping $I : A \to [0,1]$, also called a *fuzzy set* on A. The set of all fuzzy sets on A will be written as $\mathscr{F}(A)$. We can extend a fuzzy interpretation I as follows. Each n-ary connective f is interpreted by a function $\mathtt{f} : [0,1]^n \to [0,1]$. For instance, the n-ary connective "average" can correspond to the function $[0,1]^n \to [0,1] : (x_1,\dots,x_n) \mapsto \frac{1}{n}\sum_{i=1}^{n} x_i$. We define $[f(\alpha_1,\dots,\alpha_n)]_I = \mathtt{f}([\alpha_1]_I,\dots,[\alpha_n]_I)$ for formulas α_i $(1 \le i \le n)$. For $c \in [0,1]\cap\mathbb{Q}$ we have $[c]_I = c$. If C is a set of formulas we say that I is a *fuzzy model* of C iff $[\alpha]_I = 1$ for all $\alpha \in C$; we write this as $I \models C$. For fuzzy interpretations $I_1, I_2 \in \mathscr{F}(A)$ we write $I_1 \le I_2$ iff $I_1(a) \le I_2(a)$ for all $a \in A$. If $I_1 \le I_2$ and $I_1 \ne I_2$, we write $I_1 < I_2$. A fuzzy model I is a *minimal fuzzy model* of a set of formulas C if there does not exist a fuzzy model J of C such that $J < I$.

We will now recall some generalizations of the classical connectives. Specifically, *triangular norms* (short t-norm) are used to generalize classical conjunction. These are mappings $\mathtt{T} : [0,1]^2 \to [0,1]$ that are commutative, associative, increasing and for which it holds that $\mathtt{T}(x,1) = x$ for each $x \in [0,1]$. Disjunction can be generalized by a *triangular conorm* (short t-conorm). These are mappings $\mathtt{S} : [0,1]^2 \to [0,1]$ that are commutative, associative, increasing and for which it holds that $\mathtt{S}(x,0) = x$ for each $x \in [0,1]$. Logical implication can be generalized by an implicator, i.e. a function $\mathtt{I} : [0,1]^2 \to [0,1]$ such that $\mathtt{I}(0,0) = \mathtt{I}(0,1) = \mathtt{I}(1,1) = 1$ and $\mathtt{I}(1,0) = 0$ and $\mathtt{I}$ is decreasing in the first component and increasing in the second. Given a t-norm $\mathtt{T}$, the *residual implicator* $\mathtt{I}_\mathtt{T}$ of $\mathtt{T}$, defined as

$$\mathtt{I}_\mathtt{T}(x,y) = \sup\{\lambda \mid \lambda \in [0,1] \text{ and } \mathtt{T}(x,\lambda) \le y\}$$

satisfies all these conditions. If $\mathtt{T}$ is a left-continuous t-norm, then for all $x,y \in [0,1]$ it holds that $x \leq y$ iff $\mathtt{I}_{\mathtt{T}}(x,y) = 1$ ([9]). In general, residual implicators are a good choice to generalize classical implication since they satisfy a generalization of the modus ponens rule: $\mathtt{T}(x, \mathtt{I}_{\mathtt{T}}(x,y)) \leq \min(x,y)$. For continuous t-norms $\mathtt{T}$ there is an even stronger property: $\mathtt{T}(x, \mathtt{I}_{\mathtt{T}}(x,y)) = \min(x,y)$. Consider a residual implicator $\mathtt{I}$ and a t-norm $\mathtt{T}$. The *biresiduum* of $\mathtt{I}$ and $\mathtt{T}$ is defined as $\mathtt{E}_{\mathtt{T},\mathtt{I}}(x,y) = \mathtt{T}(\mathtt{I}(x,y), \mathtt{I}(y,x))$. This function is a generalization of the logical equivalence. Note that $\mathtt{I}$ does not need to be the residual implicator of $\mathtt{T}$; for an arbitrary residual implicator $\mathtt{I}$ it holds that either $\mathtt{I}(x,y) = 1$ or $\mathtt{I}(y,x) = 1$. Finally, negation can be generalized by a negator, i.e. a function $\mathtt{N} : [0,1] \to [0,1]$ such that $\mathtt{N}$ is decreasing, $\mathtt{N}(1) = 0$ and $\mathtt{N}(0) = 1$. Every implicator $\mathtt{I}$ induces a negator $\mathtt{N}_{\mathtt{I}}$ defined as $\mathtt{N}_{\mathtt{I}}(x) = \mathtt{I}(x,0)$.

Logics whose semantics are based on (left-)continuous triangular norms form an important subclass of fuzzy logics; they generalize the classical logical connectives in a natural way. In examples we will often use the fuzzy logic based on the Łukasiewicz t-norm. For the connectives conjunction $\otimes$, disjunction $\oplus$, implication $\to$ and negation $\neg$, and a fuzzy interpretation $I \in \mathscr{F}(A)$ we then have

- $[\alpha \otimes \beta]_I = \max([\alpha]_I + [\beta]_I - 1, 0)$
- $[\alpha \oplus \beta]_I = \min([\alpha]_I + [\beta]_I, 1)$
- $[\alpha \to \beta]_I = \min(1 - [\alpha]_I + [\beta]_I, 1)$
- $[\neg \alpha]_I = 1 - [\alpha]_I$

Łukasiewicz logic is often used in applications because it preserves many nice properties from classical logic. Moreover, among the t-norm based logics, Łukasiewicz logic is the only one with a continuous residual implicator. This means that a set of formulas in Łukasiewicz logic can be seen as a set of constraints on continuous functions. This logic is also closely related to mixed integer programming. McNaughton [13] showed this in a non-constructive way and Hähnle [8] gave a concrete translation from a set of formulas in Łukasiewicz logic into a mixed integer program. Finally, Łukasiewicz logic is also very close to linear logic, see e.g. [**?**].

More general, truth values do not need to be values in $[0,1]$. An arbitrary complete lattice will do the trick as well. We restrict to the unit interval because it is intuitive and convenient in practice.

3 Fuzzy answer set programming (FASP)

Recall the example from the introduction of the chapter. Suppose you want to travel by car from one city to another and you want to have an idea about the time needed to do this. The driving time can depend on several factors, for instance the amount of snow, the distance and the traffic. These concepts are a matter of degree rather than boolean properties, thus we cannot directly use ASP to model this problem. One solution is to allow propositions to be true to a certain degree in $[0,1]$ and to generalize the syntax and semantics of ASP using fuzzy logics. We now see "driving time", "snow" and "traffic" as atoms that can be assigned a truth value in $[0,1]$. To do this, an appropriate rescaling is needed. For instance, "snow" will have truth value 0 if there is no snow at all and it will have a truth degree $x > 0$ if there is snow, but it will be given a different value depending on how much snow falls from the sky and if it melts or not. The degree of "driving time" then depends on $f(\text{snow}, \text{distance}, \text{traffic})$ with f corresponding to a $[0,1]^3 \to [0,1]$-mapping that is increasing in each of its arguments. In practice, this function f can be defined by using statistical information. We can then write the rule

$$\text{driving time} \leftarrow f(\text{snow}, \text{distance}, \text{traffic}).$$

The syntax and semantics of FASP, as we will define below, can deal with such general functions f. A lower bound on the driving time can then be found in an answer set that corresponds to a solution of the FASP program.

3.1 Programs and fuzzy models

Consider a set of atoms A. An atom corresponds to a property that may have a certain truth degree in $[0,1]$, not restricted to only 1 (true) or 0 (false).

Definition 1. A *FASP program* is a finite set of rules of the form

$$r : g(a_1, \ldots, a_n) \leftarrow f(b_1, \ldots, b_m, \text{not}_1\, c_1, \ldots, \text{not}_k\, c_k),$$

with $a_i, b_j, c_l \in A \cup ([0,1] \cap \mathbb{Q})$ ($i \in \{1, \ldots, n\}$, $j \in \{1, \ldots, m\}$ and $l \in \{1, \ldots, k\}$), f and g resp. $(m+k)$-ary and n-ary connectives. We assume f resp. g is interpreted by a function $\mathtt{f} : [0,1]^{m+k} \to [0,1]$ resp. $\mathtt{g} : [0,1]^n \to [0,1]$ such that $\mathtt{f}$ and $\mathtt{g}$ are increasing in all their arguments. We also assume that $\leftarrow$ is interpreted by a residual implicator and each negation-as-failure operator not_j corresponds to a negator $\mathtt{N_j}$. We refer to the rule by its label r.

Note that in line with the tradition in logic programming we write a rule as $\alpha \leftarrow \beta$ where $\leftarrow$ is actually an implication $\to$. Also remark that in the definition of rules, we restrict to rational numbers to ensure that the language remains recursively enumerable. The expression $g(a_1, \ldots, a_n)$ is called the *head* r_h of rule r and $f(b_1, \ldots, b_m,$ $\text{not}_1\, c_1, \ldots, \text{not}_k\, c_k)$ is called the *body* r_b. Typically the connectives correspond to the connectives from a given fuzzy logic (see Section 2), but other choices, e.g. averaging operators, can be useful as well. A rule of the form "$0 \leftarrow a$" is usually written as "$\leftarrow a$" and a rule of the form "$a \leftarrow 1$" as "$a \leftarrow$".

If the connectives in Definition 1 are restricted to compositions of the classical connectives, i.e. conjunctions in the body and disjunctions in the head, and the truth values are restricted to 0 and 1, we obtain the same syntax as classical ASP. Note that in classical ASP, it is for example not needed to consider disjunctions in the body of rules since a rule $a \leftarrow b \vee c$ can be expressed by the two rules $a \leftarrow b$ and $a \leftarrow c$. As will become clear, for FASP this is not the case.

By using fuzzy interpretations (see Section 2), one can assign truth degrees to atoms and rules. For instance, in a FASP program, a rule

$$\text{open} \leftarrow 0.5$$

is modeled by a fuzzy interpretation I iff $I(\text{open}) \geq 0.5$.

Definition 2. A *fuzzy interpretation* I of a FASP program P is an element of $\mathscr{F}(\mathscr{B}_P)$, with $\mathscr{B}_P$ the set of atoms occurring in P. A fuzzy interpretation I is called a *fuzzy model* of P iff $[r]_I = 1$ for all $r \in P$.

If we restrict the fuzzy interpretations in Definition 2 to mappings $\mathscr{B}_P \to \{0,1\}$, we get classical interpretations of ASP programs.

Recall that we are interested in the "minimal" knowledge that can be derived from a program: from a single rule "open $\leftarrow$ 0.5", we want to derive that the truth degree of "open" is 0.5. One can use minimal fuzzy models to deal with this.

Example 1. Consider the following program P:

$$\begin{aligned} r_1 : \ \text{open} &\leftarrow \text{not}\,\text{closed} \\ r_2 : \text{closed} &\leftarrow \ \text{not}\,\text{open} \end{aligned}$$

We assume that "$\leftarrow$" and "not" correspond to resp. the Łukasiewicz implicator and the Łukasiewicz negator. The properties "open" and "closed" can be given a value $[0,1]$ depending on the extent, e.g. the angle, to which a door is opened resp. closed. Each combination of degrees of "open" and "closed", not necessarily meaningful, is represented by a fuzzy interpretation. The rule r_1 intuitively means that the door is open to a degree greater or equal than the extent to which the door is not closed. Rule r_2 implies the opposite property. Specifically, a fuzzy interpretation I models the program P iff

$$\begin{aligned} I(\text{open}) &\geq 1 - I(\text{closed}) \\ I(\text{closed}) &\geq 1 - I(\text{open}). \end{aligned}$$

By considering for example the rule

$$r_3 : \text{open} \leftarrow 0.5$$

we add the information that the door must be open to at least degree 0.6. The minimal fuzzy model of the program only containing rule r_3 is the fuzzy interpretation I such that $I(\text{open}) = 0.6$. As will become clear later, the fuzzy interpretation I with $I(\text{open}) = 0.6$ and $I(\text{closed}) = 0.4$ is an answer set of the program consisting of rule r_1, r_2 and r_3.

One can consider different types of programs, depending on the rules they contain. Programs without negation-as-failure are called *positive*, programs with exactly one atom in the head are called *normal* and normal programs that are positive are called *simple*. Let us discuss these programs more in detail. We start with generalizing the idea of forward chaining from ASP to simple programs.

3.2 Simple programs and answer sets

For simple programs, the minimal fuzzy model exists and is unique [**?**]. Similar to ASP, minimal fuzzy models of simple FASP programs can be characterized by forward chaining, as illustrated below and subsequently defined more formally.

Example 2. Consider the program P:

$$\begin{aligned} a &\leftarrow 0.1\\ b &\leftarrow 0.8\\ c &\leftarrow a \oplus b\\ a &\leftarrow b \otimes c \end{aligned}$$

First, consider the fuzzy interpretation $I_0 : \mathscr{B}_P \rightarrow [0,1] : x \mapsto 0$ by which every atom has truth degree 0. However, I_0 is not a fuzzy model: for example the first rule imposes that the truth degree of a must be greater or equal to 0.1. Let us increase the truth values by defining $I_1 : \mathscr{B}_P \rightarrow [0,1]$. To model $a \leftarrow 0.1$ and $b \leftarrow 0.8$, we put $I_1(a) = 0.1$ and $I_1(b) = 0.8$. To model the third rule, we put $I_1(c) = \max(I_1(a) \oplus I_1(b), 0) = 0.9$. We check if I_1 models the last rule. This is not the case since $I_1(a) = 0.1$ and $I_1(b) \otimes I_1(c) = 0.7$. We need to adjust I_1 once more: we define a new fuzzy interpretation I_2 with $I_2(a) = 0.7$, $I_2(b) = I_1(b)$ and $I_2(c) = I_1(c)$ The fuzzy interpretation I_2 is the unique minimal fuzzy model of P.

Definition 3. Consider a simple FASP program P. A fuzzy interpretation I is the *answer set* of P if it is the minimal fuzzy model of P.

For simple programs the minimal fuzzy model, i.e. the answer set, coincides with the least fixpoint of the immediate consequence operator Π_P [5]. This operator maps fuzzy interpretations to fuzzy interpretations and is defined as

$$\Pi_P(I)(a) = \sup\{[r_b]_I \mid (a \leftarrow r_b) \in P\},$$

for $a \in \mathscr{B}_P$ and a fuzzy interpretation I. Intuitively, the minimal fuzzy model of a simple FASP program corresponds to the maximal information one can derive by forward chaining until no new knowledge can be discovered anymore.

Note that, unlike ASP, it is not always possible to compute this fixpoint in a finite number of steps. Consider for instance the program containing the single rule "$a \leftarrow \frac{a+1}{2}$". It will take infinitely many steps taken by the immediate consequence operator to find the least fixpoint $I(a) = 1$ [19].

If constraints, i.e. rules in which the head is a constant, are allowed in simple programs, the least fixpoint of the immediate consequence will exist, but there may be no fuzzy model at all.

Example 3. Consider the program P:

$$\begin{aligned} a &\leftarrow 1\\ 0 &\leftarrow a \end{aligned}$$

The least fixpoint of Π_P is the fuzzy interpretation I with $I(a) = 1$. However, P has no fuzzy model since there cannot exist a fuzzy interpretation M such that $M(a) \geq 1$ and $0 \geq M(a)$. To deal with constraints one can use the fact that a fuzzy interpretation I is an answer set of $P \cup C$, with P a simple FASP program and C a set of constraints iff I is an answer set of P and a fuzzy model of C.

3.3 Positive programs and answer sets

If the heads of the rules in a positive program P can be more general formulas than only single atoms, P can have several minimal fuzzy models, or none at all. In any case, they present the minimal knowledge that can be derived from P.

Example 4. Consider the program P:

$$\begin{array}{rcl} a \oplus b & \leftarrow & 0.3 \\ a & \leftarrow & b \\ b & \leftarrow & 0.1 \end{array}$$

A minimal fuzzy model of P is the fuzzy interpretation I such that $I(a) = 0.2$ and $I(b) = 0.1$. But, for example I' such that $I'(a) = 0.15$ and $I'(b) = 0.15$ is a minimal fuzzy model as well. On the other hand the program

$$\begin{array}{rcl} \min(a, 0.5) & \leftarrow & b \\ b & \leftarrow & 0.1 \oplus a \end{array}$$

has no (minimal) fuzzy models. Indeed, for a fuzzy interpretation I to model this program it must hold that $\min(0.1 + I(a), 1) \leq \min(I(a), 0.5)$. If $0.1 + I(a) \leq 1$, this would imply that $0.1 + I(a) < I(a)$ and if $0.1 + I(a) \geq 1$, then it would follow that $1 < 0.5$.

Definition 4. Consider a positive FASP program P. A fuzzy interpretation I is an *answer set* of P if it is a minimal fuzzy model of P.

Note that the immediate consequence operator cannot be used for programs with more than one atom in the heads of rules; the truth value of the body of a rule does not necessarily have an equal impact on the truth values of each of the atoms in the head.

3.4 General programs and answer sets

In this section, we will generalize the definitions of ASP to arbitrary FASP programs. For FASP programs that are not positive, answer sets can no longer be defined directly in terms of minimal fuzzy models.

Example 5. Consider for example the program

$$\begin{aligned} a &\leftarrow a \\ 0 &\leftarrow \text{not}\, a \end{aligned}$$

with "not" interpreted by the Łukasiewicz negator. The only minimal fuzzy model is I such that $I(a) = 1$. However, the justification for deriving a truth value for a only depends on itself, so this fuzzy model is not in line with the intuition of forward chaining. To solve this problem, we will reduce a general FASP program to a positive FASP program.

Intuitively, we "guess" an answer set I and replace all negation-as-failure literals $\text{not}\, c$ by their fuzzy interpretation $[\text{not}\, c]_I$. For the program from Example 1

$$\begin{aligned} \text{open} &\leftarrow \text{not}\,\text{closed} \\ \text{closed} &\leftarrow \text{not}\,\text{open} \end{aligned}$$

a suitable guess would be I with $I(\text{open}) = 0.6$ and $I(\text{closed}) = 0.4$; a door is closed to the degree 0.4 if it is opened to the degree $1 - 0.4 = 0.6$. Let us now consider the same program, but we replace "not closed" and "not open" by their fuzzy interpretations under I:

$$\begin{aligned} \text{closed} &\leftarrow 0.4 \\ \text{open} &\leftarrow 0.6 \end{aligned}$$

The minimal fuzzy model of this program is exactly I. Hence, I was a stable guess and we say that it is answer set of the program.

Note that I_x with $I_x(\text{open}) = x$ and $I_x(\text{closed}) = 1 - x$ with $x \in [0,1]$ are stable guesses as well.

Definition 5. Consider a FASP program P and a fuzzy interpretation I. The *reduct* r^I of a rule r in P w.r.t. I is obtained by replacing all expressions of the form $\text{not}_j\, a$ by the fuzzy interpretation $[\text{not}_j\, a]_I$. The *reduct* P^I of P w.r.t. I is the set of rules

$$P^I = \{r^I \mid r \in P\}.$$

Note that this definition generalizes the well-known Gelfond-Lifschitz transformation [7], used to transform general ASP programs to positive ASP programs.

Formally, we have the following definition.

Definition 6. Consider a FASP program P and a fuzzy interpretation I. I is called an *answer set* of P iff I is an answer set of P^I.

A FASP program can have several answer sets as in Example 5, or none at all, as in Example 6.

Example 6. Consider the program P consisting of the one rule

$$p \leftarrow \text{not}\, p$$

with "not" interpreted by the negator $\mathtt{N} : [0,1] \to [0,1]$ with $\mathtt{N}(x) = 0$ if $x > 0$ and $\mathtt{N}(0) = 1$. For each fuzzy interpretation I with $I(p) > 0$ we have that P^I is is the positive program consisting of the rule

$$p \leftarrow 0.$$

The unique minimal model of P^I is J with $J(p) = 0$, hence our original guess I is clearly not a minimal model of P^I. If, on the other hand, we start with this fuzzy interpretation $J(p) = 0$, then we obtain for P^J the rule

$$p \leftarrow 1.$$

J is not a fuzzy model of P^J, let alone a minimal fuzzy model. We conclude that P has no answer sets.

However, if a different negator is used, this program can have an answer set for instance if not is interpreted by the Łukasiewicz negator. For a guess $M(p) = x$ with $x \in [0,1]$, we now obtain for P^M the rule

$$p \leftarrow 1 - x.$$

Hence, M is the minimal fuzzy model of P^M if $x = 1 - x$ or $x = 0.5$.

By Definition 6 it follows that an answer set of a FASP program P is also a fuzzy model of P. One can even prove that it must be a minimal fuzzy model of P [**?**]. The converse however does not hold.

Example 7. Recall the program from Example 5

$$\begin{aligned} a &\leftarrow a \\ 0 &\leftarrow \text{not}\, a \end{aligned}$$

with "not" interpreted by the Łukasiewicz negator. The only minimal fuzzy model is the fuzzy interpretation I such that $I(a) = 1$. However it is not an answer set since I is not a minimal fuzzy model of

$$\begin{aligned} a &\leftarrow a \\ 0 &\leftarrow 0 \end{aligned}$$

4 Motivating example

Forest fires cause massive loss of vegetation and animal life. If a fire is detected on time, suppression units are able to reach the fire in its initial stages which is important to avoid huge losses. Moreover suppression costs will be considerably reduced. Wireless sensor networks can be effectively used for this purpose [22]. These networks consist of a number of devices that can sense their environment and communicate wirelessly. We will use FASP to determine, given measurements made by the sensors about the temperature, if there are sensors that are not working optimally and if so, within what range we can reasonably assume the temperature to be.

Suppose we have n sensors. By assuming an appropriate linear rescaling, we can see temperature as a value in $[0,1]$. Although we will not be able to derive an exact temperature, we will try to find a subinterval of $[0,1]$ in which we may assume the temperature to be. More specifically, for each sensor $i \in \{1,\ldots,n\}$, we denote the upper bound on the exact temperature at its location as t_i^{up} and the lower bound as t_i^{low}. The measured temperature is t_i'. The sensor network defines a weighted graph G as follows. The vertices are the sensors and there is an edge with weight $w_{ij} \in [0,1]$ between the vertices corresponding to sensor i and sensor j. The value w_{ij} is such that we can reasonably assume, based on the locations of sensors i and j that the temperature difference between these locations must be less than $1 - w_{ij}$. Finally, we suppose that b_i represents the error on the temperature measured by sensor i.

We can write the following program P. For $i, j \in \{1,\ldots,n\}$ we have the rules

1. $t_i^{low} \leftarrow t_i' \otimes \text{not}\, b_i$
2. $t_i^{up} \leftarrow t_i' \oplus b_i$
3. $b_i \oplus b_j \leftarrow (t_i' \nleftrightarrow t_j') \otimes w_{ij}$

where we use the connectives from Łukasiewicz logic and assume that the negation-as-failure operator is interpreted by the Łukasiewicz negation. The constants $t_i' \nleftrightarrow t_j'$, which define the degree to which t_i' and t_j' are different are defined as $1-(t_i' \leftrightarrow t_j')$ with $\leftrightarrow$ the Łukasiewicz biresiduum.

Rules 1 and 2 define the relationship between the actual and the measured temperature. For a fuzzy interpretation I to model these rules, it most hold for each sensor i that $I(t_i') - I(b_i) \leq I(t_i^{low})$ and $I(t_i') + I(b_i) \leq I(t_i^{up})$. An answer set I is such that $I(t_i^{up})$ and $I(t_i^{low})$ will be chosen minimal. Rule 3 imposes that if the difference between t_i' and t_j' is too large with respect to $1 - w_{ij}$, then there must be something wrong with sensors i and/or j. The semantics of FASP makes sure that the "total error" is distributed over sensor i and j in a minimal way.

Consider as a concrete example a forest with three sensors. Suppose we have $t_1' = 0.4$, $t_2' = 0.9$ and $t_3' = 0.5$ and $w_{1,2} = 0.8$, $w_{1,3} = 0.8$ and $w_{2,3} = 0.8$. Hence, we have $I(t_1' \nleftrightarrow t_2') = 0.5$, $I(t_1' \nleftrightarrow t_3') = 0.1$ and $I(t_2' \nleftrightarrow t_3') = 0.4$.

For I to be a fuzzy model of rule 3, it most hold, for each sensor i that

$$\max(I(t_i' \nleftrightarrow t_j') + I(w_{ij}) - 1, 0) \leq I(b_i) + I(b_j),$$

or more specifically,

1. $0.3 \leq I(b_1) + I(b_2)$
2. $0 \leq I(b_1) + I(b_3)$
3. $0.2 \leq I(b_2) + I(b_3)$

A possible "guess" for an answer set I could be such that $I(b_1) = 0.29$, $I(b_2) = 0.01$ and $I(b_3) = 0.19$. From rules 1 and 2, we obtain that for I to be an answer set of P, i.e. a minimal fuzzy model of P^I, we must have $I(t_1^{low}) = 0.11$ and $I(t_1^{up}) = 0.69$, $I(t_2^{low}) = 0.89$ and $I(t_2^{up}) = 0.91$, $I(t_3^{low}) = 0.31$ and $I(t_3^{up}) = 0.69$.

Another possibility is a fuzzy interpretation J with $J(b_1) = 0.15$, $J(b_2) = 0.15$ and $J(b_3) = 0.05$.

The answer sets provide us with all possible ways in which the "total error" can be "divided" over the sensors. For each such setting, we also obtain an upper and lower bound on the actual temperature at the locations of the sensors.

5 Some remarks about FASP

In ASP, there are two types of negation: negation-as-failure and strong negation. When an atom a and its negation $\neg a$ both appear in the head of rules, there is the possibility of inconsistency. In a fuzzy context, the classical definition of consistency must be modified since a literal a and its negation $\neg a$ can be both true in a consistent way. One solution could be to define fuzzy interpretations of a program P as elements in $\mathscr{L}_P$, i.e. the set of literals in P and to add the rules $1 \leftarrow a \oplus \neg a$ for each $a \in \mathscr{L}_P$. In [21], degrees of consistency of fuzzy interpretations are discussed.

Logic programming, which contains ASP as a special case, has had a significant impact on the development of nonmonotonic logics and vice versa [2]. It is closely related to e.g. autoepistemic, equilibrium and default logic. Equilibrium logic [16] is one of the most general approaches to ASP. Programs, seen as sets of formulas in equilibrium logic, can be arbitrary propositional theories without restrictions on where the two types of negation may occur. When restricting to the syntax of ASP, there is a one-to-one correspondence between the equilibrium models of a program and its answer sets. This result is generalized to FASP in [18] by introducing a fuzzy equilibrium logic. Even when very general constructs such as in this chapter are allowed, the answer sets of a FASP program correspond to its fuzzy equilibrium models. Furthermore, a reduction from the problem of finding fuzzy equilibrium models to the problem of solving a particular bilevel mixed integer program is proposed, allowing to implement reasoners already existing for these types of problems for FASP. By using the complexity of fuzzy equilibrium logic, it was also shown that computational complexity of FASP is equal to that of ASP in the general case. This means that in general, adding fuzziness to ASP, does not imply an increase in the computational complexity. In [**?**] some results about the computational complexity of FASP with Łukasiewicz semantics are presented. For simple programs a correspondence to an open problem was shown, which indicates that setting the complexity may not be easy. However, there is P-membership for several interesting subclasses. The correspondence between autoepistemic logic and ASP [11] was generalized in [4].

As was illustrated by examples, some FASP programs have no solutions. In practice however, it might be suitable to opt for an imperfect solution. One strategy is to add weights to the rules in a FASP program: rules do not have to be satisfied to degree 1. In [**?**], aggregated FASP is proposed. The idea is not to immediately state the extent to which each rule should be satisfied, but to let an aggregator function determine an overall score of suitability of a solution. Contrary to the case of classical ASP, there is not yet an efficient FASP solver, even without considering aggregators. A well-known technique for ASP consists of translating a program to a propositional theory whose models correspond to the answer sets of the program. In [**?**] the first steps towards an efficient FASP solver are taken by generalizing the ideas of translating an ASP program to a SAT instance. A FASP solver could then use existing techniques for solving fuzzy satisfiability problems such as mixed integer programming. The completion of a FASP program is introduced and it was shown that in case the program has no cyclic dependencies, which induce so called loops, the models of this completion are the answer sets of the original program. For programs that contain loops, a reduction to fuzzy SAT is proposed using a generalization of the notion of loop formulas in ASP to FASP [**?**]. One of the most important issues that still need to be tackled for building a good solver is optimizing the grounding of FASP programs.

6 Conclusions

We presented an introduction to fuzzy answer set programming, a recently developed framework that is suitable for modeling search problems with continuous domains. The syntax and semantics make FASP highly configurable and applicable in different domains. However, a lot of topics still need to be investigated, FASP is by far not as developed as ASP. Little is known about the computational complexity of specific fragments and there are almost no techniques available to actually calculate answer sets.

References

1. BARAL, C. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, 2003.
2. BARAL, C., AND GELFOND, M. Logic programming and knowledge representation. *Journal of Logic Programming 19* (1994), 73–148.
3. BARAL, C., GELFOND, M., AND RUSHTON, J. Probabilistic reasoning in computer science. In *Logic Programming and Nonmonotonic Reasoning, 7th International Conference* (2004), pp. 21–33.
4. BLONDEEL, M., SCHOCKAERT, S., DE COCK, M., AND VERMEIR, D. Fuzzy autoepistemic logic: Reflecting about knowledge of truth degrees. In *Symbolic and Quantitative Approaches to Reasoning with Uncertainty* (2011), pp. 616–627.
5. DAMÁSIO, C., AND PEREIRA, L. Antitonic logic programs. In *Proceedings of the 6th International Conference on Logic Programming and Nonmonotonic Reasoning* (2001), pp. 379–392.
6. DUBOIS, D., AND PRADE, H. Possibility theory, probability theory and multiple-valued logics: a clarification. *Annals of Mathematics and Artificial Intelligence 32*, 1-4 (2001), 35–66.
7. GELFOND, M., AND LIFSCHITZ, V. The stable model semantics for logic programming. In *Proceedings of the Fifth International Conference and Symposium on Logic Programming* (1988), pp. 1070–1080.

8. HÄHNLE, R. Proof theory of many-valued logic - linear optimization - logic design: connections and interactions. *Soft Computing 1* (1997), 107–119.
9. HÁJEK, P. *Metamathematics of Fuzzy Logic*. Trends in Logic, 1998.
10. JANSSEN, J., SCHOCKAERT, S., VERMEIR, D., AND DE COCK, M. General fuzzy answer set programs. In *Proceedings of the International Workshop on Fuzzy Logic and Applications* (2009), pp. 353–359.
11. LIFSCHITZ, V., AND SCHWARZ, G. Extended logic programs as autoepistemic theories. In *Proceedings of the Second International Workshop on Logic Programming and Nonmonotonic Reasoning* (1993), pp. 101–114.
12. ŁUKASIEWICZ, T., AND STRACCIA, U. Tightly integrated fuzzy description logic programs under the answer set semantics for the semantic web. In *Proceedings of the 1st International Conference on Web Reasoning and Rule Systems* (2007), pp. 289–298.
13. MCNAUGHTON, R. A theorem about infinite-valued sentential logic. *The Journal of Symbolic Logic 16*, 1 (1951).
14. MOORE, R. Semantical considerations on nonmonotonic logic. In *Proceedings of the Eight International Joint Conference on Artificial Intelligence* (1983), pp. 272–279.
15. NICOLAS, P., GARCIA, L., STÉPHAN, I., AND LEFÈVRE, C. Possibilistic uncertainty handling for answer set programming. *Annals of Mathematics and Artificial Intelligence 47*, 1-2 (2006), 139–181.
16. PEARCE, D. A new logical characterisation of stable models and answer sets. In *NMELP '96: Selected papers from the Non-Monotonic Extensions of Logic Programming* (1997), pp. 57–70.
17. REITER, R. A logic for default reasoning. *Artificial Intelligence 13*, 1-2 (1980), 81–132.
18. SCHOCKAERT, S., JANSSEN, J., VERMEIR, D., AND DE COCK, M. Answer sets in a fuzzy equilibrium logic. In *Proceedings of the 3rd International Conference in Web Reasoning and Rule Systems* (2009), pp. 135–149.
19. STRACCIA, U. Annotated answer set programming. In *Proceedings of the 11th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems* (2006).
20. STRACCIA, U., OJEDA-ACIEGO, M., AND DAMÁSIO, C. V. On fixed-points of multi-valued functions on complete lattices and their application to generalized logic programs. *SIAM Journal on Computing*, 5 (2009), 1881–1911.
21. VAN NIEUWENBORGH, D., DE COCK, M., AND VERMEIR, D. An introduction to fuzzy answer set programming. *Annals of Mathematics and Artificial Intelligence 50*, 3-4 (2007), 363–388.
22. YU, L., WANG, N., AND MENG, X. Real-time forest fire detection with wireless sensor networks. In *Proceedings of the Wireless Communication Networking and Mobile Computing International Conference* (2005), pp. 1214–1217.