Computing Fuzzy Answer Sets using DLVHEX

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. Fuzzy answer set programming has been introduced as a framework that successfully combines the concepts of answer set programming and fuzzy logic. In this paper, we show how the fuzzy answer set semantics can be mapped onto the semantics for HEX-programs, which are nonmonotonic logic programs under the answer set semantics that support the use of external function calls. By using the DLVHEX reasoning engine, we so devise a vehicle for effectively computing fuzzy answer sets.

1 Introduction

The answer set programming (ASP) paradigm [9] has gained a lot of popularity in the last years, due to its truly declarative nonmonotonic semantics, which has been proven useful in a number of interesting applications, e.g. [15, 2, 12, 10]. The idea behind the answer set semantics, a generalization of the stable model semantics [8], 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".

An alternative characterization of answer sets is given in [16, 14] in terms of unfounded sets. Intuitively, an unfounded set is a set of literals for which there is no motivation to suppose that these literals have to be true. The reason herefore is that these literals either depend on each other, or the rules that can motivate them are not applicable. A candidate answer set M is then an answer set of P iff it is a model of P and it does not contain such unfounded sets.

On the other hand, fuzzy logic is a suitable framework for dealing with degrees of truth and satisfaction [19]. 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

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

ones in the top and bottom elements of the lattice. One of the strengths of fuzzy logic is that a user can freely choose, depending on the type of application under consideration, which specific definition she uses for these operations.

In [18, 17] both formalisms, i.e. ASP and fuzzy logic, have been combined into the single framework of fuzzy answer set programming (FASP) to increase the flexibility and hence the application potential of ASP. In the FASP framework, fuzzy answer sets are considered, meaning that literals can belong to an answer set to a certain extent, as opposed to either belonging to the answer set or not. Further, the semantics also allows that, if desired, both a and $\neg a$ can be true to a certain degree at the same time without necessarily loosing consistency. Similarly, a more flexible interpretation of negation as failure is allowed and a rule can be satisfied to a certain degree. While [18] defines the semantics in a direct fashion, similar to the normal ASP semantics, [17] adapts the notion of an unfounded set to the setting of fuzzy interpretations.

HEX-programs are introduced in [4] as nonmonotonic logic programs with higher order atoms and external function calls under the fully declarative ASP semantics. While higher order features are useful in the context of meta-reasoning, external function calls allow to exchange knowledge with external sources. Although the main area of application for HEX-programs is the Semantic Web, we show in this paper that some of its features, especially external function calls, can also be fruitfully applied in other interesting contexts, i.e. to compute fuzzy answer sets.

The mapping we propose in this work to compute fuzzy answer sets using HEX-programs is based on the characterisation of the FASP semantics with unfounded sets from [17]. The idea behind the mapping comes from the guess-and-check methodology that is often used in ASP applications. The guess part of our encoding computes a consistent fuzzy model for a given program using a meta-programming technique. To handle the fuzzy operations w.r.t. the lattice under consideration, we use a number of external function calls. The more difficult checking part of our encoding also uses meta-programming, but combined with a saturation technique that exploits the minimality criterion of answer sets for disjunctive programs, which is e.g. used in [3] to show Σ_2^P-hardness of disjunctive logic programs under the ASP semantics.

The rest of this paper is organized as follows. In Section 2 we give some preliminaries on fuzzy answer set programming and HEX-programs, while Section 3 presents the mapping from fuzzy answer sets to HEX-programs. In Section 4 we outline how the translation from Section 3 could be used to approximate answer sets for FASP programs that use an infinite truth lattice. We end our paper with some conclusions and directions for future research in Section 5.

2 Preliminaries

2.1 Truth Lattices

Throughout 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 [1]. 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[3] of $\mathcal{L}$. If the lattice $\mathcal{L}$ under consideration is clear from the context, we omit the subscripts and use $\leq$, 0 and 1 to denote the ordering, the smallest and the greatest element respectively.

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. [13]).

A *negator* on $\mathcal{L}$ is any $\mathcal{L} \to \mathcal{L}$ mapping $\mathcal{N}$ satisfying $\mathcal{N}(0) = 1$ and $\mathcal{N}(1) = 0$. Moreover we require $\mathcal{N}$ to be decreasing, i.e. $x_1 \leq x_2$ implies $\mathcal{N}(x_1) \geq \mathcal{N}(x_2)$ for all x_1 and x_2 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 $\mathcal{T}(1, x) = x$, for all x in $\mathcal{L}$. Moreover we require $\mathcal{T}$ to be increasing in both of its components, i.e.[4] $x_1 \leq x_2$ implies $\mathcal{T}(x_1, y) \leq \mathcal{T}(x_2, y)$ for all x_1, x_2 and y in $\mathcal{L}$. 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 $\mathcal{S}(0, x) = x$, 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 $\mathcal{I}(0, 0) = 1$, and $\mathcal{I}(1, x) = x$, for all x in $\mathcal{L}$. Moreover we require $\mathcal{I}$ to be decreasing in its first, and increasing in its second component, i.e. $x_1 \leq x_2$ implies $\mathcal{I}(x_1, y) \geq \mathcal{I}(x_2, y)$ as well as $\mathcal{I}(y, x_1) \leq \mathcal{I}(y, x_2)$ for all x_1, x_2 and y in $\mathcal{L}$.

A fuzzy set in U is a $U \mapsto \mathcal{L}$ mapping. The set of all elements that have a non-zero membership degree in a fuzzy set A in U is called the *support*, i.e. $supp(A) = \{x | A(x) >_{\mathcal{L}} 0_{\mathcal{L}}\}$. For fuzzy sets A and B in U, A is said to be included in B, denoted by $A \subseteq_{\mathcal{L}} B$, iff $A(u) \leq_{\mathcal{L}} B(u)$ for all u in U. As usual, we have $A \subset_{\mathcal{L}} B$ iff $A \subseteq_{\mathcal{L}} B$ and not $B \subseteq_{\mathcal{L}} A$. Again, when the lattice under consideration is clear from the context, we will omit the subscripts.

2.2 Fuzzy Answer Set Programming

We give some preliminaries concerning the fuzzy answer set semantics along the lines of [17]. 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 $\neg\neg a = a$. 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. $Y^- = \{l \mid \textit{not } l \in Y\}$. Further, we use *not* X to denote the set $\{\textit{not } l \mid l \in X\}$.

A *rule* is of the form $a \leftarrow \beta$, where[5] a is either a literal or the symbol $\bot$ and β is a finite set of extended literals. To denote the head a of the rule, we use $H(a \leftarrow \beta)$, while $B(a \leftarrow \beta)$ is used to denote the body β. When the *head* of a rule r is the symbol $\bot$, i.e. $H(r) = \bot$, the rule is called a *constraint*, while rules with an empty body, i.e.

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

[4] Note that the monotonicity of the second component immediately follows from that of the first component due to the commutativity.

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

$B(r) = \emptyset$ are called *facts*. For constraints, we normally omit the head symbol, i.e. we use $\leftarrow \beta$ instead of $\bot \leftarrow \beta$.

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 $Elit_P = Lit_P \cup \mathit{not}\ Lit_P$.

An important feature of the FASP framework, which it inherits from fuzzy logic, is its high configurability: it allows a user to choose, in function of the application at hand, how the different classical operations need to be interpreted. More specifically, a user must select a complete lattice $\mathcal{L}$ first. Then, she has to choose two negators $\mathcal{N}_c$ and $\mathcal{N}_n$, for defining consistency and the semantics of negation as failure respectively. Further, two t-norms need to be fixed: $\mathcal{T}_c$ is used for defining consistency and $\mathcal{T}_a$ determines the applicability of rules. Also an implicator $\mathcal{I}$ is needed to obtain the degree of satisfaction of a rule. Finally, two aggregators $\mathcal{A}_c$ and $\mathcal{A}_p$ are needed, where $\mathcal{A}_c$ combines the consistency degrees of the different atoms and their negation into a consistency degree for the interpretation, while $\mathcal{A}_p$ 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.

To allow, in a fuzzy context, for a literal l to be true to a certain degree, or to allow that both l and $\neg l$ can be a bit true in a consistent way, an interpretation has to be a fuzzy set, which makes 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. With I we associate a consistency function $I_c : \mathcal{B}_P \mapsto \mathcal{L}$, defined as $I_c(a) = \mathcal{N}_c(\mathcal{T}_c(I(a), I(\neg a)))$ for each $a \in \mathcal{B}_P$. Further, I is called x-**consistent**, $x \in \mathcal{L}$, iff $\mathcal{A}_c(\mathcal{B}_P, I_c) \geq x$.*

Intuitively, the consistency function computes the degree of consistency of a particular atom and its negation in an interpretation, while the definition of x-consistency allows a user to choose how the individual consistencies of the atoms and their negations have to be combined into a degree of consistency for the whole interpretation. By choosing an appropriate lower bound x, the user can then fix the point where an interpretation is no longer considered consistent. Note that by using the aggregator $\mathcal{A}_c$ the user can choose to ignore certain inconsistencies, or she can allow certain literals to be more inconsistent than others. However, it is demanded that an aggregator is increasing whenever the degrees of the individual consistencies are increasing.

As fuzzy interpretations only assign truth values to ordinary literals explicitly, a mechanism to retrieve truth values for naf-literals is needed. While complementary literals l and $\neg l$ are only weakly related to each other using $\mathcal{N}_c$, $\mathcal{T}_c$, $\mathcal{A}_c$ 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 $I(\mathit{not}\ l) = \mathcal{N}_n(I(l))$ for each $l \in Lit_P$.

Having fuzzy interpretations and x-consistency, we need to redefine the satisfaction of rules. While a rule in classical ASP is either satisfied or not, FASP allows a more flexible setting where rules can be partially (to a certain degree) satisfied. Further, rules need not be satisfied to the same degree. These degrees are obtained using $\mathcal{T}_a$ 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.

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*

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

Note that $I_{\models}(\{l\}) = I(l)$ and $I_{\models}(\mathit{not}\ l) = \mathcal{N}_n(I(l))$, as $\{l\} = \{l\} \cup \emptyset$ and $\mathcal{T}_a(I(l), 1) = I(l)$. 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 $\mathcal{A}_p$, 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. It is demanded that the 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 $\mathcal{A}_p(P, I_{\models}) \geq y$.*

Example 1. Consider the lattice $\mathcal{L} = \{0, 0.1, \ldots, 0.9, 1\}$ and the program

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

and consider the fuzzy interpretations[6] $K = \{(a, 0.9), (b, 0.2), (c, 0.9), (d, 0.9)\}$ and $L = \{(a, 0.9), (b, 0.9), (d, 0.1)\}$. Taking $\mathcal{A}_c(\mathcal{B}_P, I_c) = \inf\{I_c(a) \mid a \in \mathcal{B}_P\}$, it is clear that both K and L are 1-consistent, independently of the choices for $\mathcal{N}_c$ and $\mathcal{T}_c$. For negation as failure, we use the negator $\mathcal{N}_n(x) = 1 - x$. To evaluate the body of r_2 we use $\mathcal{T}_a(x, y) = min(x, y)$, while the implicator that we use is $\mathcal{I}(x, y) = 1$, if $x \leq y$ and $\mathcal{I}(x, y) = y$ otherwise. Finally, as an aggregator for the rules, we use $\mathcal{A}_p(P, I_{\models}) = \inf\{I_{\models}(s) \mid s \in P\}$, i.e. the weakest rule dominates the solution.

Now, one can check that we have

$$L_{\models}(B(r_2)) = \mathcal{T}_a(L(a), L(\mathit{not}\ b)) = \mathcal{T}_a(0.9, 1 - 0) = min(0.9, 1) = 0.9\ ,$$

and thus

$$L_{\models}(r_2) = \mathcal{I}(L_{\models}(B(r_2)), L(b)) = \mathcal{I}(0.9, 0.9) = 1\ .$$

Similarly, we have $L_{\models}(r_1) = \mathcal{I}(1, 0.9) = 0.9$ and $L_{\models}(r_3) = \mathcal{I}(0.1, 0.1) = 1$. As a result, L is a fuzzy 0.9-model of P.

[6] 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 $(l, 0)$.

For the above example, one can check in a similar way that also K is a fuzzy 0.9-model. However, intuitively, K seems less acceptable than L. E.g., when the rule r_2 is considered w.r.t. K, we see that $K(b) = 0.2$, while $K(b) = 0.1$ would suffice to obtain the same degree of satisfaction for r_2, i.e. $K_{\models}(r_2) = \mathcal{I}(\mathcal{T}_a(0.9, 1-0.9), 0.2) = \mathcal{I}(0.1, 0.2) = 1 = \mathcal{I}(0.1, 0.1)$. Further, there is no support for accepting c at degree 0.9, as there is no applicable rule with c in the head. On the other hand, L does not suffer from these problems as each literal in L appears in the head of a rule and none of the truth degrees of the head literals can be lowered without lowering the degree of satisfaction of the corresponding rule.

To trace such intuitively incorrect fuzzy models, [17] defines a fuzzy version of the classical unfounded set characterisation [16, 14] of ASP. To obtain this fuzzy variant, one needs to define for each rule r in the program a subset Y of $\mathcal{L}$ such that assigning any value from Y to the head of r does not lower r's degree of satisfaction. However, when the lower values in the range are chosen, the rule is said to *support* its head literal better. Formally, the support of a rule $l \leftarrow \beta \in P$ w.r.t. a fuzzy interpretation I is defined as $I_s(l \leftarrow \beta) = \inf\{y \mid \mathcal{I}(I_{\models}(\beta), y) \geq I_{\models}(l \leftarrow \beta)\}$.

Definition 4. *Let P be a program and let I be a fuzzy interpretation. A set of literals X is an unfounded set w.r.t. I iff for each literal $l \in X$ and each rule $l \leftarrow \beta \in P$:*

1. *$\beta \cap X \neq \emptyset$; or*
2. *$I(l) > I_s(l \leftarrow \beta)$; or*
3. *$I_{\models}(\beta) = 0$ holds.*

*A fuzzy interpretation I is **unfounded-free** iff $supp(I) \cap X = \emptyset$ for every unfounded set X w.r.t. I. An x-consistent unfounded-free fuzzy y-model of P is called an x-consistent **fuzzy y-answer set** of P.*

Example 2. Reconsider Example 1. One can check that $X = \{b, c, d\}$ is an unfounded set w.r.t. K, as

- for $b \in X$, we have $K_s(r_2) = \inf\{y \mid \mathcal{I}(0.1, y) \geq= 1\} = \inf[0.1, 1]$, yielding that $K(b) = 0.2 > K_s(r_2) = 0.1$;
- for $c \in X$, we have no rules in P with c in the head of the rule, vacuously satisfying the conditions in Definition 4; and
- for $d \in X$, we have $K_{\models}(r_3) = \mathcal{I}(1 - 0.2, 0.9) = \mathcal{I}(0.8, 0.9) = 1$ and thus $K_s(r_3) = \inf\{y \mid \mathcal{I}(0.8, y) \geq= 1\} = \inf[0.8, 1]$, yielding that $K(d) = 0.9 > K_s(r_3) = 0.8$.

On the other hand, for L and the rules in P, we have

$$L(a) = \inf\{y \mid \mathcal{I}(1, y) \geq 0.9\} = \inf[0.9, 1]$$
$$L(b) = \inf\{y \mid \mathcal{I}(0.9, y) \geq 1\} = \inf[0.9, 1]$$
$$L(d) = \inf\{y \mid \mathcal{I}(0.1, y) \geq 1\} = \inf[0.1, 1]$$

As a result, L is clearly unfounded-free and thus a fuzzy 0.9-answer set of P.

2.3 HEX-programs

In this subsection we briefly introduce the semantics of HEX-programs, and refer the reader to [4] for a detailed in-depth introduction. To write HEX-programs, one needs three mutually disjoint sets $\mathcal{C}$, $\mathcal{X}$, and $\mathcal{G}$ of respectively constant names, variable names, and external predicate names. As usual, variables start with an uppercase letter, while constants start with a lowercase letter. Further, external predicate names are prefixed with "&". As the semantics supports higher-order atoms, constants serve both as individual and predicate names.

A term in a HEX-program is any element of $\mathcal{C} \cup \mathcal{X}$, while a higher-order atom is of the form $Y_0(Y_1, \ldots, Y_n)$, $n \geq 0$, where $Y_0, Y_1, \ldots, Y_n$ are terms. When Y_0 is a constant, we have a classical ordinary atom. An external atom is of the form $\&g[I_1, \ldots, I_m](O_1, \ldots, O_n)$, $g \in \mathcal{G}$, where $I_1, \ldots, I_m$ and $O_1, \ldots, O_n$ are two lists of terms called respectively the input and output of the external atom. A HEX-program is a set of rules of the form

$$\alpha_1 \vee \cdots \vee \alpha_k \leftarrow \beta_1, \ldots, \beta_m, \mathit{not}\ \beta_{m+1}, \ldots, \mathit{not}\ \beta_n$$

where $n, k \geq 0$, $\alpha_1, \ldots, \alpha_k$ are higher-order atoms, and $\beta_1, \ldots, \beta_n$ are either higher-order atoms or external atoms. Again, we use $H(r)$ and $B(r)$ to denote the head (i.e. $\{\alpha_1, \ldots, \alpha_k\}$) and body (i.e. $\{\beta_1, \ldots, \beta_n\}$) of a rule r respectively.

As usual, the Herbrand base of a HEX-program P, denoted $\mathcal{B}_P$, is the set containing all possible ground versions of higher-order and external atoms occurring in P, which are obtained by replacing variables with constants from $\mathcal{C}$. Similarly, one can define the grounding[7] of rules and the grounding of programs, where $gr(P)$ is used to denote the grounded version of a HEX-program P.

Any set $I \subseteq \mathcal{B}_P$ is called an interpretation for P, and we use $I \models a$ iff $a \in I$. With every external atom $\&g[I_1, \ldots, I_m](O_1, \ldots, O_n)$, an $(m+n+1)$-ary Boolean function $f_{\&g}$ is associated such that every tuple $(I, i_1, \ldots, i_m, o_1, \ldots, o_n)$, with $I \subseteq \mathcal{B}_P$ and $i_k, o_k \in \mathcal{C}$, is assigned either 0 (false) or 1 (true). For an interpretation I and a ground external atom $a = \&g[i_1, \ldots, i_m](o_1, \ldots, o_n)$, we use $I \models a$ iff $f_{\&g}(I, i_1, \ldots, i_m, o_1, \ldots, o_n) = 1$. For a ground rule r of a HEX-program P, we define

- $I \models H(r)$ iff $\exists a \in H(r) \cdot I \models a$;
- $I \models B(r)$ iff $\forall a \in B(r) \backslash \mathit{not}\ B(r)^- \cdot I \models a$ and $\forall a \in B(r)^- \cdot I \not\models a$; and
- $I \models r$ iff $I \models H(r)$ whenever $I \models B(r)$.

An interpretation I is a model for a HEX-program P iff $I \models r$ for all rules $r \in gr(P)$.

While the classical answer set semantics is defined by using the Gelfond-Lifschitz reduct [9], HEX-programs use the more recent notion of the FLP-reduct [7]. For a HEX-program P and an interpretation I, we define the FLP-reduct of P w.r.t. I, denoted fP^I, as the set of all rules $r \in gr(P)$ such that $I \models B(r)$. Then, I is an answer set of P iff I is a minimal model of fP^I[8].

[7] Note that an ungrounded atom a in a negation as failure construction *not* a is also grounded.

[8] Note that fP^I may still contain negation-as-failure and, therefore, it may have several minimal models.

Example 3. Consider[9] the following HEX-program P inspired by an example from [4]. Besides the facts $\{node(a) \leftarrow ; node(b) \leftarrow ; node(c) \leftarrow ; graph(g) \leftarrow ; graph(h) \leftarrow\}$, we have the following rules in our program

$$G(N, M) \vee no\_edge(G, N, M) \leftarrow graph(G), node(N), node(M)$$
$$reached(G, X) \leftarrow graph(G), \&reachable[G, a](X)$$

Intuitively, this program will generate edges for two graphs g and h using the first rule, and then, using the second rule, compute for those graphs which nodes are reachable from a.

For the first rule, the grounded program $gr(P)$ will contain rules like

$$g(a, c) \vee no\_edge(g, a, c) \leftarrow graph(g), node(a), node(c)$$
$$h(b, a) \vee no\_edge(h, b, a) \leftarrow graph(h), node(b), node(a)$$

while the second rule will generate grounded rules like

$$reached(g, b) \leftarrow graph(g), \&reachable[g, a](b)$$
$$reached(h, c) \leftarrow graph(h), \&reachable[h, a](c)$$

Now, we associate with the external atom $\&reachable$ a function $f_{\&reachable}$ such that $f_{\&reachable}(I, E, A, B) = 1$ iff B is reachable in the graph E from A. E.g., if $I = \{g(a, b), g(b, c)\}$, then $I \models \&reachable[g, a](c)$ as $f_{\&reachable}(I, g, a, c) = 1$.

Finally, one can check that, e.g.,

$$J = \{g(a, b), g(b, c), h(a, c), reached(g, b), reached(g, c), reached(h, c)\} \ ,$$

is an answer set for the above program, where, for clarity, the no_edge atoms are ommitted, i.e. for every $x, y \in \{a, b, c\}$ and $z \in \{g, h\}$, we have $no\_edge(z, x, y) \in J$ iff $z(x, y) \notin J$.

Indeed, the FPL-reduct fP^J contains all grounded versions of the disjunctive rule, as these are always applicable w.r.t. J. Further, fP^J contains the rules

$$reached(g, b) \leftarrow graph(g), \&reachable[g, a](b)$$
$$reached(g, c) \leftarrow graph(g), \&reachable[g, a](c)$$
$$reached(h, c) \leftarrow graph(h), \&reachable[h, a](c)$$

and thus, J is clearly a minimal model of fP^J, yielding that I is an answer set.

The semantics of HEX-programs has been implemented in the DLVHEX system [5], which is a frontend to the DLV answer set solver for disjunctive programs [11]. The external atoms are supported in DLVHEX by a simple and elegant plugin framework that allows a user to write her own plugins in C++.

[9] For simplicity, we assume that $\mathcal{C}$, $\mathcal{X}$, and $\mathcal{G}$ are implicitly defined by the rules in the program.

3 From Fuzzy Answer Sets to HEX-programs

In this section we present a translation from the FASP semantics to HEX-programs such that the fuzzy answer sets can be retrieved from the answer sets of the HEX-programs. The mapping we present is based on the well-known guess-and-check methodology in answer set programming [6]. Intuitively, the guess part of our encoding will be responsible for computing x-consistent fuzzy y-answer sets, while the checking part of the encoding will check the foundedness condition of fuzzy models.

In what follows, we assume that the lattice $\mathcal{L}$ under consideration is finite[10], i.e. $\mathcal{L} = \{v_1, \ldots, v_n\}$, where $n > 0$. To handle the lattice operations, e.g. negation or implication, we will use external atoms. Our translation needs, among others, the following external functions:

- $\&nlt[X,Y]()$ which is defined as $f_{\&nlt}(I,X,Y) = 1$ iff $X \not<_{\mathcal{L}} Y$, with $X, Y \in \mathcal{L}$;
- $\&nlte[X,Y]()$ which is defined as $f_{\&nlte}(I,X,Y) = 1$ iff $X \not\leq_{\mathcal{L}} Y$, with $X, Y \in \mathcal{L}$;
- $\&negator[V](NV)$ which is defined as $f_{\&negator}(I,V,NV) = 1$ iff $NV = \mathcal{N}_n(V)$, with $V, NV \in \mathcal{L}$;
- $\&t\_norm[X_1,\ldots,X_k](X)$ which is defined, for $k = 0$, as $f_{\&t\_norm}(I,X) = 1$ iff $X = 1_{\mathcal{L}}$; and, for[11] $k > 0$, as $f_{\&t\_norm}(I,X_1,\ldots,X_k,X) = 1$ iff

$$X = \mathcal{T}_a(X_k, \mathcal{T}_a(X_{k-1}, \mathcal{T}_a(\ldots \mathcal{T}_a(X_1, 1_{\mathcal{L}})))) \ ;$$

- $\&implicator[X,Y](Z)$ which is defined as $f_{\&implicator}(I,X,Y,Z) = 1$ iff $Z = \mathcal{I}(X,Y)$, with $X, Y, Z \in \mathcal{L}$;
- $\&support[X,Z](Y)$ which is defined as $f_{\&support}(I,X,Z,Y) = 1$ iff $Y = \inf\{y \mid \mathcal{I}(X,y) \geq Z\}$, with $X, Y, Z \in \mathcal{L}$.

Note that we use $\&nlt$ and $\&nlte$ to handle $\not<_{\mathcal{L}}$ and $\not\leq_{\mathcal{L}}$, as the properties $x \not< y \Leftrightarrow x \geq y$ and $x \not\leq y \Leftrightarrow x > y$ do not hold for non-total lattices.

3.1 The Guess Program P_{guess}

The guess program we present in this subsection will compute, for a program P, the x-consistent fuzzy y-models, with $x, y \in \mathcal{L}$. To represent a fuzzy interpretation (and thus a fuzzy model), we use the following set of rules that introduces the binary metapredicate fi (**f**uzzy **i**nterpretation):

$$\{fi(l,v_1) \vee \cdots \vee fi(l,v_n) \leftarrow \ \mid l \in Lit_P\} \ .$$

Intuitively, these rules encode a guess for a fuzzy interpretation, i.e. a single truth value for each literal in the program. The minimality criterion of the answer set semantics for

[10] Later on, we will discuss how infinite lattices could be handled, to a certain extent, in our approach.

[11] Although we define for each $k \geq 0$ an external atom $\&t\_norm$, we only need a finite number of those external atoms for a given program P, i.e. one for each different body size of a rule in P.

disjunctive programs ensures that only one truth value is selected for each literal. To handle the special literal $\bot$ in constraint rules, we use the fact

$$fi(\bot, 0_{\mathcal{L}}) \leftarrow$$

Next, we need rules that introduce the negation as failure variants of each literal using the truth values that are assigned to the literals in the fuzzy interpretation. Again, we will use a binary metapredicate to represent this knowledge, i.e. nfi:

$$nfi(L, NV) \leftarrow fi(L, V), \&negator[V](NV)$$

To compute the degree of consistency of a fuzzy interpretation, we use the external function $\&consistency[F](X)$ which is defined as $f_{\&consistency}(I, F, X) = 1$ iff the fuzzy interpretation J^F, i.e. the fuzzy interpretation that is represented by the predicate F (and disregarding $\bot$), satisfies $\mathcal{A}_c(\mathcal{B}_F, J^F_c) = X$ (see Definition 1), where $\mathcal{B}_F$ is the set of all atoms (except $\bot$) that are encoded in F. Note that $\mathcal{B}_F = \mathcal{B}_P$ as each literal $l \in Lit_P$ is present in F. This allows to implement the external function $\&consistency[F](X)$ independently of a particular program. The external atom is then used in the first rule below to compute the exact degree of consistency of the fuzzy interpretation, while the second rule, i.e. a constraint, is used to allow all degrees of consistency that are higher than the demanded boundary $x \in \mathcal{L}$.

$$\begin{aligned} consistent(X) &\leftarrow \&consistency[fi](X) \\ &\leftarrow consistent(X), \&nlte[x, X]() \end{aligned}$$

To compute fuzzy models one needs to obtain the degrees in which rules are applicable, i.e. the degree of truth of their bodies. This is accomplished by the following rule for each rule $r = a \leftarrow b_1, \ldots, b_k, \mathit{not}\ b_{k+1}, \ldots, \mathit{not}\ b_m \in P$:

$$\begin{aligned} body(r, X, a) \leftarrow\ & fi(b_1, X_1), \ldots, fi(b_k, X_k), nfi(b_{k+1}, X_{k+1}), \ldots, nfi(b_m, X_m), \\ & \&t\_norm[X_1, \ldots, X_m](X) \end{aligned}$$

To make some of the rules below easier to read, we opted for $body$ to be a ternary predicate, where the third argument corresponds to the head literal of the rule denoted by the first argument. Note that constraint rules are also handled in this way as constraints have $\bot$ as their head literal.

Using the applicability of rules, we are now able to compute the degree of satisfaction of the rules in a program, using the rule

$$rule(R, Z) \leftarrow body(R, X, H), fi(H, Y), \&implicator[X, Y](Z)$$

Finally, a fuzzy interpretation I is sanctioned a fuzzy y-model, $y \in \mathcal{L}$, iff the aggregation of the rule satisfactions w.r.t. I is not below the threshold y. To handle the aggregation, we define an external atom $\&aggregate[R](X)$ as $f_{\&aggregate}(I, R, X) = 1$ iff the aggregation of the rule satisfactions encoded in the predicate R equals the truth value X, i.e. $\mathcal{A}_p(P^R, J^R_{\models}) = X$ (see Definition 3), where P^R corresponds to the program represented by the R-predicate and $J^R_{\models}$ corresponds to the rule satisfaction degrees

encoded by the R-predicate. In the first rule below this external atom is used to compute the aggregation degree of a fuzzy interpretation, while the second rule, a constraint, is used to assure that no degree lower than $y \in \mathcal{L}$ is accepted

$$\begin{aligned} aggregation(X) &\leftarrow \&aggregate[rule](X) \\ &\leftarrow aggregation(X), \&nlte[y, X]() \end{aligned}$$

Now, for a program P, we use $P^{x,y}_{guess}$ to denote the HEX-program obtained by combining all the rules described above. It is not difficult to see that this program $P^{x,y}_{guess}$ exactly computes the x-consistent fuzzy y-models of P.

3.2 The Check Program P_{check}

To check, for a program P, whether an x-consistent fuzzy y-model M of P computed by $P^{x,y}_{guess}$ is also an x-consistent fuzzy y-answer set of P, we need additional rules, denoted by P_{check}, that check M for unfounded sets. To accomplish this, we will use the saturation technique that is used in some complexity proofs of the answer set semantics for disjunctive programs [3] and in the work of automated integration of guess and check programs in answer set programming [6].

Intuitively, we will compute a set of literals X. If X is such that it has no literal in common with the fuzzy interpretation contained in fi, it cannot be used to show that fi is not unfounded-free. Therefore, we consider X always as founded in this case and we will saturate the answer set accordingly. On the other hand, when X has at least one literal in common with the fuzzy interpretation contained in fi, then, if X is founded, we will saturate the answer set. Thus, because of the minimality of answer sets, if a set X exists that is unfounded w.r.t. fi, no saturation will occur and we will have a minimal answer set for the HEX-program. As a result, saturated answer sets of the HEX-program $P^{x,y}_{guess} \cup P_{check}$ will correspond to x-consistent fuzzy y-answer sets of P.

First, we need a rule in P_{check} that computes the support for each rule in the program, i.e.

$$sup(R, Y) \leftarrow body(R, X, H), rule(R, Z), \&support[X, Z](Y)$$

Next, we need to guess a set X of literals, that will be checked for (un)foundedness. To represent the set of literals X, we will use a unary metapredicate x and to guess the set, we use the rules

$$\{x(l) \vee x(l') \leftarrow \ \mid l \in Lit_P\} \ .$$

Intuitively, the above rules encode that each literal $l \in Lit_P$ can be either in X, i.e. $x(l)$ is chosen, or not be in X, i.e. $x(l')$ is chosen, where l' is a (unique) mirror version of l used solely for encoding that l is either in X or not in X.

Now we check whether the set X of literals represented by the predicate x is unfounded or not. In case it is not unfounded, we will saturate the answer set such that, if there exists an unfounded set X', an answer set that guesses X' is always smaller (w.r.t. subset inclusion) than the answer set that guesses X. To this end, we will define when a set of literals X is founded, i.e. we will use the negated version of Definition 4. A set of literals X is not unfounded, i.e. is founded, w.r.t. a fuzzy interpretation I iff there exists a literal $l \in X$ and a rule $r = l \leftarrow \beta \in P$ such that:

– $\beta \cap X = \emptyset$; and
– $I(l) \not> I_s(r)$; and
– $I_{\models}(\beta) > 0$.

To encode the above definition, we need an additional external atom, denoted $\&no\_body\_intersection[R, X]()$, that is able to check the condition $\beta \cap X = \emptyset$. The foundedness condition for a set X, encoded by the predicate x can then be represented by the rule

$$\begin{aligned} \mathit{founded} \leftarrow\ & x(L), \mathit{body}(R, B, L), \&no\_body\_intersection[R, x](), \\ & \mathit{fi}(L, V), \mathit{sup}(R, S), \&nlt[S, V](), B \neq 0_{\mathcal{L}} \end{aligned} \tag{1}$$

The external function $\&no\_body\_intersection[R, X]()$ that is used in the above rule is defined as $f_{\&no\_body\_intersection} = 1$ iff the rule R has no literal represented by the predicate X in its body. However, this definition is not sufficient, as it will always result in 0 for non-fact rules in case the answer set was saturated, i.e. in such an answer set, the intersection of the body of a non-fact rule with the literals encoded in the predicate X is never empty. As a result, the above rules for checking foundedness would be removed in the FPL reduct, which is undesirable for a correct translation, as will be shown later on. To remedy this, we add the following condition to the definition: $f_{\&no\_body\_intersection} = 1$ also whenever the predicate X holds for both l and l' for every literal $l \in Lit_P$. Thus, in a saturated answer set the rules for checking foundedness are always contained in the FPL reduct.

It is clear from Definition 4 that for a set of literals X such that $\mathit{supp}(J^{\mathit{fi}}) \cap X = \emptyset$, where J^{fi} denotes the fuzzy interpretation represented by the predicate fi, it does not matter whether X is unfounded or not to determine whether J^{fi} is a fuzzy answer set. For this reason we introduce a rule that derives $\mathit{founded}$ whenever $\mathit{supp}(J^{\mathit{fi}}) \cap X = \emptyset$, which is checked by the external function $\&no\_intersection[F, X]()$, i.e.

$$\mathit{founded} \leftarrow \&no\_intersection[\mathit{fi}, x]() \tag{2}$$

The external function $\&no\_intersection[F, X]()$ that is used in the above rule is defined as $f_{\&no\_intersection} = 1$ iff for the fuzzy interpretation J^F encoded in F and the set of literals encoded in X it holds that $\mathit{supp}(J^{\mathit{fi}}) \cap X = \emptyset$. Again there is a problem with this definition if the answer set under consideration is saturated, as the above rule (2) would be removed in the FPL reduct because the external atom would return 0 in such an answer set. The solution is similar to the one adopted above: $f_{\&no\_intersection} = 1$ should also hold whenever the predicate X holds for both l and l' for every literal $l \in Lit_P$. Thus, the rule is kept in the FPL reduct for a saturated answer set.

Finally, we introduce some rules that ensure saturation whenever $\mathit{founded}$ holds.

$$\begin{gathered} \mathit{founded} \vee \mathit{unfounded} \leftarrow \\ \mathit{unfounded} \leftarrow \mathit{founded} \\ \{x(l) \leftarrow \mathit{founded}\ ;\ x(l') \leftarrow \mathit{founded} \mid l \in Lit_P\} \end{gathered}$$

Intuitively, the first rule states that a set of literals X is either founded or unfounded. The second rule and the third set of rules are responsible for the saturation of the answer

set in case $founded$ is derived, i.e. both $unfounded$ and $\{x(l), x(l')\}$ for every literal $l \in Lit_P$ are added to the answer set in this case.

The following theorem confirms that the combination of $P^{x,y}_{guess}$ and P_{check} can be used to retrieve the x-consistent fuzzy y-answer sets of P.

Theorem 1. *Let P be a program, let I be a fuzzy interpretation and take $x, y \in \mathcal{L}$.*

I is an x-consistent fuzzy y-answer set of P iff there exists an answer set I' of $P^{x,y}_{guess} \cup P_{check}$ such that

- $\{fi(l, I(l)) \mid l \in Lit_P\} \subseteq I'$*; and*
- $\{founded, unfounded\} \subseteq I'$.

Proof. (sketch)

$\boxed{\Rightarrow}$ Let I be an x-consistent fuzzy y-answer set of P and consider $I' = I'_{guess} \cup I'_{check}$, where I'_{guess} is the answer set of $P^{x,y}_{guess}$ that corresponds to I. Note that, by construction of $P^{x,y}_{guess}$, there is exactly one such I'_{guess}. Further, I'_{check} contains the atoms $\{founded, unfounded\} \cup \{x(l), x(l') \mid l \in Lit_P\}$ and $\forall r \in P \cdot \exists x \in \mathcal{L} \cdot \forall y \in \mathcal{L} \backslash x \cdot sup(r, x) \in I'_{check} \wedge sup(r, y) \notin I'_{check}$.

Clearly, I' is a model of $P^{x,y}_{guess} \cup P_{check}$. The only thing left to show is that I' is minimal w.r.t. the FPL reduct. To see this, one can show that the FPL reduct contains all rules in P_{check} except some of the support computing rules (which play no role in the saturation). Suppose I' is not minimal w.r.t. the FPL reduct, i.e. there exists an interpretation $I'' \subset I'$ such that I'' is a model of the FPL reduct. The only way this would be possible is to have at least $founded \notin I''$, which implies that none of the rules with $founded$ in the head should be applicable w.r.t. I''. Thus, I'' has to make a choice for a set X encoded in the predicate x such that the rule (2), $founded \leftarrow \&no\_intersection[fi, x]()$, remains unapplicable, i.e. the condition $supp(I) \cap X \neq \emptyset$ holds; and such that none of the rules (1) $founded \leftarrow x(L), \ldots, B \neq 0_{\mathcal{L}}$ are applicable. As a result, X is an unfounded set w.r.t. I, which yields a contradiction.

$\boxed{\Leftarrow}$ Let I' be an answer set of $P^{x,y}_{guess} \cup P_{check}$ such that $\{founded, unfounded\} \subseteq I'$, i.e. I' is a saturated answer set. As in the other direction of the proof, we can write I' as $I'_{guess} \cup I'_{check}$, where I'_{guess} corresponds to an answer set of $P^{x,y}_{guess}$. Let I be the fuzzy interpretation encoded by I'_{guess}. Clearly, I is an x-consistent fuzzy y-model of P.

Now, suppose I is not a fuzzy answer set, i.e. there exists a set of literals X such that $supp(I) \cap X \neq \emptyset$ and X is unfounded w.r.t. I. Consider

$$I'' = I' \backslash (\{founded\} \cup \{x(l) \mid l \notin X\} \cup \{x(l') \mid l \in X\}) \ .$$

Clearly, I'' is a model of the FPL reduct w.r.t. I' as none of the rules with only $founded$ in the head become applicable with this choice for X, and thus $unfounded$ will be chosen in the disjunctive rule. However, $I'' \subset I'$, yielding that I' is not an answer set, a contradiction.

4 Approximating Infinite Lattices

While the mapping from the previous section assumes a finite lattice, it can also be used to approximate solutions in cases where $\mathcal{L}$ is infinite.

This is useful as fuzzy operations are commonly defined on the infinite lattice $[0, 1]$ while, on the other hand, approximate answers are often satisfactory. E.g. it is unlikely that one would be interested in an $0.85372\ldots$-consistent fuzzy $0.94563\ldots$-answer set.

In such infinite cases one could start by using a finite sublattice of $\mathcal{L}$, e.g. $\mathcal{L}_{0.1} = \{0, 0.1, \ldots, 0.9, 1\}$ and modify the fuzzy operations accordingly. E.g., one could modify the product t-norm on $[0, 1]$ by rounding up the result to the nearest element from $\mathcal{L}_{.1}$.

In fact, one could start with a "small" finite sublattice to obtain a rough approximation of an answer set and successively employ larger (finer grained) sublattices to obtain more accurate approximations. Furthermore, the computation of successive approximations could be made more efficient by altering the rules in P_{guess} to make good use of the approximation obtained so far.

E.g., assume that using $\mathcal{L}_{0.1} = \{0, 0.1, \ldots, 0.9, 1\}$ yields an approximate solution $\{(a, 0.9), (b, 0.7)\}$. When refining to $\mathcal{L}_{0.05} = \{0, 0.05, 0.1, \ldots, 0.9, 0.95, 1\}$, one could restrict the rules in P_{guess} to

$$fi(a, 0.85) \vee fi(a, 0.9) \vee fi(a, 0.95) \leftarrow$$
$$fi(b, 0.75) \vee fi(b, 0.7) \vee fi(b, 0.75) \leftarrow$$

to find a better approximation. Obviously, since fewer guesses are possible, finding solutions using the modified P_{guess} will be faster than if a solution in $\mathcal{L}_{0.05}$ would have to be found from scratch.

5 Conclusions and Future Research

We have presented a translation from the fuzzy answer set programming framework to (the semantics of) HEX-programs using a guess-and-check methodology where external atoms in the target HEX-programs handle the fuzzy operations and a saturation technique is used to check candidate fuzzy answer set solutions for (being free of) unfounded sets.

The translation directly integrates the guess and check parts of the resulting program and does not need the ASP meta-interpreter technique described in [6].

An important added benefit of the translation to HEX-programs is that it remains usable in case the FASP semantics is extended with external atoms to handle e.g. fuzzy knowledge bases in a Semantic Web context.

In future work, we intend to implement an automated version of the translation and integrate it with the DLVHEX system. Further, we will develop a set of plugin modules for DLVHEX that support the most common fuzzy operations on some lattices, as well as the approximation strategy outlined in Section 4. Naturally, the system will need intensive benchmarking to illustrate its behavior on different kinds of applications for the FASP semantics. Finally, we also plan to look into specific optimizations of the DLVHEX system for handling FASP applications.

References

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

2. T. Eiter, W. Faber, N. Leone, and G. Pfeifer. The diagnosis frontend of the dlv system. *AI Communications*, 12(1-2):99–111, 1999.
3. T. Eiter and G. Gottlob. On the computational cost of disjunctive logic programming: Propositional case. *Annals of Mathematics and Artificial Intelligence*, 15(3-4):289–323, 1995.
4. T. Eiter, G. Ianni, R. Schindlauer, and H. Tompits. A uniform integration of higher-order reasoning and external evaluations in answer-set programming. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI-05)*, pages 90–96, 2005.
5. T. Eiter, G. Ianni, R. Schindlauer, and H. Tompits. dlvhex: A system for integrating multiple semantics in an answer-set programming framework. In *Proc. 20th Workshop on Logic Programming and Constraint Systems (WLP 06)*, pages 206–210, 2006.
6. T. Eiter and A. Polleres. Towards automated integration of guess and check programs in answer set programming: a meta-interpreter and applications. *Theory and Practice of Logic Programming*, 6(1-2):23–60, 2006.
7. W. Faber, N. Leone, and G. Pfeifer. Recursive aggregates in disjunctive logic programs: Semantics and complexity. In *Proc. of the 9th European Conference on Logics in Artificial Intelligence (JELIA 2004)*, volume 3229 of *LNAI*, pages 200–212. Springer, 2004.
8. 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.
9. M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9(3-4):365–386, 1991.
10. 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.
11. N. Leone, G. Pfeifer, W. Faber, F. Calimeri, T. Dell'Armi, T. Eiter, G. Gottlob, G. Ianni, G. Ielpa, C. Koch, S. Perri, and A. Polleres. The dlv system. In *Proc. of the Eur. Conf. on Logics in AI (JELIA2002)*, volume 2424 of *LNCS*, pages 537–540. Springer, 2002.
12. 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.
13. V. Novák, I. Perfilieva, and J. Močkoř. *Mathematical Principles of Fuzzy Logic*. Kluwer Academic Publishers, 1999.
14. D. Sacca and C. Zaniolo. Stable models and non-determinism in logic programs with negation. In *PODS '90: Proceedings of the ninth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 205–217. ACM Press, 1990.
15. 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.
16. A. van Gelder, K. A. Ross, and J. S. Schlipf. Unfounded sets and well-founded semantics for general logic programs. In *Proceedings of the 7th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 221–230. ACM Press, 1988.
17. D. Van Nieuwenborgh, M. De Cock, and D. Vermeir. An introduction to fuzzy answer set programming. Submitted.
18. D. Van Nieuwenborgh, M. De Cock, and D. Vermeir. Fuzzy answer set programming. In *Proc. of the 10th European Conference on Logics in Artificial Intelligence (JELIA 2006)*, volume 4160 of *LNAI*, pages 359–372. Springer, 2006.
19. L. Zadeh. Fuzzy logic and approximate reasoning. *Synthese 30*, pages 407–428, 1975.