TLP **13** (4–5): Online Supplement, July 2013.
URL: http://dx.doi.org/10.1017/S1471068413000471

Fuzzy answer sets approximations

MARIO ALVIANO*

Department of Mathematics and Computer Science, University of Calabria, 87036 Rende (CS), Italy
(*e-mail:* alviano@mat.unical.it)

RAFAEL PEÑALOZA†

Dresden University of Technology, 01062 Dresden, Germany
Center for Advancing Electronics Dresden
(*e-mail:* penaloza@tcs.inf.tu-dresden.de)

submitted 10 April 2013; revised 23 June 2013; accepted 5 July 2013

Abstract

Fuzzy answer set programming (FASP) is a recent formalism for knowledge representation that enriches the declarativity of answer set programming by allowing propositions to be graded. To now, no implementations of FASP solvers are available and all current proposals are based on compilations of logic programs into different paradigms, like mixed integer programs or bilevel programs. These approaches introduce many auxiliary variables which might affect the performance of a solver negatively. To limit this downside, operators for approximating fuzzy answer sets can be introduced: Given a FASP program, these operators compute lower and upper bounds for all atoms in the program such that all answer sets are between these bounds. This paper analyzes several operators of this kind which are based on linear programming, fuzzy unfounded sets and source pointers. Furthermore, the paper reports on a prototypical implementation, also describing strategies for avoiding computations of these operators when they are guaranteed to not improve current bounds. The operators and their implementation can be used to obtain more constrained mixed integer or bilevel programs, or even for providing a basis for implementing a native FASP solver. Interestingly, the semantics of relevant classes of programs with unique answer sets, like positive programs and programs with stratified negation, can be already computed by the prototype without the need for an external tool.

KEYWORDS: fuzzy logic, answer set programming, search-space pruning operators

1 Introduction

Answer Set Programming (ASP), i.e., logic programming under stable model semantics (Gelfond and Lifschitz 1991), is a declarative language for knowledge representation (Niemelä 1999; Marek and Truszczyński 1999; Lifschitz 2002). In ASP, problems are modeled by specifying a set of requirements that all solutions,

* Partially supported by Regione Calabria within the PIA project KnowRex POR FESR 2007–2013.
† Partially supported by DFG under grant BA 1122/17-1 and within the Cluster of Excellence 'cfAED'.

called answer sets, have to satisfy. One of the strengths of ASP is its capability to model non-monotonic knowledge, overcoming a well-known limitation of classical logic, which can only deal with monotonic inferences. While monotonicity is desired in mathematics, it is widely considered a weakness for knowledge representation (Baral 2003), where non-monotonicity arises in common reasoning tasks such as reasoning by default, abductive reasoning and belief revision. ASP can handle these tasks naturally (Marek and Remmel 2004; Lin and You 2002; Delgrande *et al.* 2008), allowing for modeling and reasoning on incomplete information, and possibly retracting some conclusions as new knowledge on the application domain is acquired. Since complete knowledge can only be achieved in mathematical abstraction, it can be stated that ASP makes logic closer to the real world.

However, ASP is still based on precise information, which cannot always be assumed in the real world. For example, measures provided by any instrument or sensor always come with some degree of tolerance, and information expressed in natural language are often vague: there is no precise way to distinguish persons that are tall from those who are not. Fuzzy logic (Dubois *et al.* 1991) can handle vague information of this kind by interpreting propositions with a truth degree in the interval of real numbers $[0, 1]$. Intuitively, the higher the degree assigned to a proposition, the *more true* it is, with the extreme elements 0 and 1 denoting *totally false* and *totally true*, respectively. Consider for example the *Barber of Seville* paradox: In the small town of Seville, all and only those men who do not shave themselves are shaved by the barber. Classical set theory can neither prove nor disprove that the barber shaves himself, hence a fuzzy interpretation of this proposition should be $1/2$, the most undetermined truth degree.

Fuzzy Answer Set Programming (FASP) aims at combining ASP and fuzzy logic. For example, a FASP encoding of the Barber of Seville paradox is

$$shaves(barber, X) \leftarrow not\ shaves(X, X) \quad shaves(X, X) \leftarrow not\ shaves(barber, X)$$

from which $shaves(barber, barber)$ gets the expected truth degree $1/2$. FASP has been defined for a very general framework (Nieuwenborgh *et al.* 2007b), allowing several connectors to be combined in the same program. With the aim of providing an indication for implementing a FASP solver, more constrained frameworks have been considered by Lukasiewicz (2006) and Janssen *et al.* (2012). However, current proposals are based on compilations into different paradigms and introduce many auxiliary variables which could affect performance negatively. The focus of this paper is on operators for approximating fuzzy answer sets. These operators can be used for limiting the search space of an external tool, such as the linear or bilevel program approaches proposed by Lukasiewicz (2006) and Janssen *et al.* (2012), or as the basis for implementing a native FASP solver.

More precisely, in this paper we describe operators for computing two fuzzy interpretations, called lower and upper bound, such that every answer set is between these bounds. We introduce fuzzy unfounded sets, which generalize the notion of unfounded sets of classical ASP to deal with fuzzy semantics. We also define a well-founded operator that combines the fuzzy T_P operator with the complement of the greatest unfounded set to improve on previously computed bounds. We show

https://doi.org/10.1017/S1471068413000471 Published online by Cambridge University Press

that these operators yield answer sets for positive and stratified FASP programs in polynomial time, while in general produce the well-founded semantics by Damásio and Pereira (2001). For dealing with normal FASP programs, we introduce the new minimal satisfiability operator S_P. For semantics based on the Łukasiewicz t-norm, this operator is polynomial-time computable and can further tighten the approximation. Finally, we describe a prototypical implementation which combines optimization ideas from classical ASP solvers (Lierler and Maratea 2004; Gebser *et al.* 2007; Alviano *et al.* 2011), and report on an experiment assessing the potential performance gain provided by our operators to a bilevel program solver.

2 Syntax and semantics

Let $\mathscr{B}$ be a fixed, finite set of propositional atoms. A *fuzzy atom* (or *atom* for short) is either a propositional atom from $\mathscr{B}$ or a numeric constant in the range $[0, 1]$, where numeric constants are overlined, e.g. $\overline{1}$, to distinguish them from propositional atoms. A *literal* is either a fuzzy atom or a fuzzy atom preceded by the default negation symbol *not*. A *normal FASP program* is a finite set of rules of the form

$$a \leftarrow b_1 \otimes \cdots \otimes b_m \otimes not\ b_{m+1} \otimes \cdots \otimes not\ b_n \tag{1}$$

where $n >= m >= 0$, $a, b_1, \ldots, b_n$ are atoms, and $\otimes$ denotes the fuzzy conjunction. For a rule r of the form (1), the atom a is called the *head* of r, denoted $H(r)$, and the conjunction $b_1 \otimes \cdots \otimes b_m \otimes not\ b_{m+1} \otimes \cdots \otimes not\ b_n$ is called the *body* of r, denoted $B(r)$. The expressions $B^+(r) = \{b_1, \ldots, b_m\}$ and $B^-(r) = \{b_{m+1}, \ldots, b_n\}$ denote the multiset of positive and negative body literals of r, respectively. Multisets thus represent conjunctions: a multiset $A = \{a_1, \ldots, a_k\}$ $(k >= 0)$ of literals represents the conjunction $\bigotimes_{i=1}^{k} a_i$. Moreover, *not* A denotes the multiset $\{not\ a_1, \ldots, not\ a_k\}$, i.e., $\bigotimes_{i=1}^{k} not\ a_i$. A rule r is *positive* if $B^-(r) = \emptyset$, and a *fact* if $B^+(r) = B^-(r) = \emptyset$. Relevant subclasses of normal FASP programs are the positive and stratified programs. A program is *positive* if all of its rules are positive. The notion of stratified program requires the introduction of the dependency graph $G_P = (\mathscr{B}, A)$ of a program P, where A contains arcs $a \rightarrow^+ b_i$ and $a \rightarrow^- b_j$ $(1 \leqslant i \leqslant m < j \leqslant n)$ for each rule $r \in P$ of the form (1). P is *stratified* if no cycle in G_P contains $\rightarrow^-$ arcs.

The semantics of FASP programs generalizes that of ASP by interpreting propositional atoms with a truth degree from the interval $[0, 1]$. An additional degree of liberty arises from the choice of the operator used to interpret fuzzy conjunctions. We focus on semantics based on t-norms (Klement *et al.* 2000). A *t-norm* is a binary, associative and commutative operator $\otimes : [0, 1] \times [0, 1] \rightarrow [0, 1]$ that is monotonic and has unit 1, i.e., for every $x, y, z \in [0, 1]$, $x \leqslant y$ implies $x \otimes z \leqslant y \otimes z$, and $x \otimes 1 = x$. There are three fundamental t-norms, called the *Gödel*, *Łukasiewicz* , and *product* t-norms, where $x \otimes y$ is defined as $\min\{x, y\}$, $\max\{x + y - 1, 0\}$, and $x \cdot y$, respectively. In the following, FASP programs are assumed to be associated with a fixed t-norm computable in polynomial time. A *fuzzy interpretation* I for a FASP program P is a fuzzy set in $\mathscr{B}$, i.e., a function $I : \mathscr{B} \rightarrow [0, 1]$ mapping each propositional atom of $\mathscr{B}$ into a truth degree in $[0, 1]$. The interpretation I is extended

https://doi.org/10.1017/S1471068413000471 Published online by Cambridge University Press

to numeric constants, negative literals and multisets as follows. For a constant $\overline{c}$, $I(\overline{c}) = c$; for a negative literal *not* b, $I(not\ b) = 1 - I(b)$; for a multiset of literals $A = \{l_1, \cdots, l_k\}$, $I(A) = \bigotimes_{i=1}^{k} I(l_i)$.

Let $\mathscr{I}$ be the set of all interpretations and $I, J \in \mathscr{I}$. I is a *subset* of J ($I \subseteq J$) if $I(a) \leqslant J(a)$ for each $a \in \mathscr{B}$. I is a strict subset of J ($I \subset J$) if $I \subseteq J$ and $I \neq J$. Fuzzy set intersection ($I \cap J$), union ($I \cup J$), and difference ($I \setminus J$) are defined as follows: for every $a \in \mathscr{B}$, $[I \cap J](a) := \min\{I(a), J(a)\}$, $[I \cup J](a) := \max\{I(a), J(a)\}$, and $[I \setminus J](a) := \max\{I(a) - J(a), 0\}$. An interpretation I *models* a rule r with head a, denoted $I \models r$, if $I(a) >= I(B(r))$. I models a FASP program P ($I \models P$) if $I \models r$ for each $r \in P$. An interpretation M is an answer set of a positive FASP program P if M is a minimal model of P, i.e., $M \models P$ and there is no interpretation $I \subset M$ such that $I \models P$. M is an answer set of a normal FASP program P if M is an answer set of the reduct P^M obtained from P by replacing each negative literal *not* b by the constant $\overline{I(not\ b)}$.

Example 1
The FASP program $P_{\mathsf{ex1}} = \{a \leftarrow c \otimes not\ b,\ b \leftarrow not\ c,\ c \leftarrow \overline{0.1}\}$ is a stratified program, containing one positive rule. Consider the interpretation I such that $I(a) = 0, I(b) = 0.9$, and $I(c) = 0.1$. The reduct P^I_{ex1} is $\{a \leftarrow c \otimes \overline{0.1},\ b \leftarrow \overline{0.9},\ c \leftarrow \overline{0.1}\}$. Under the Łukasiewicz t-norm semantics, I is a minimal model of P^I_{ex1}, and hence it is an answer set for P_{ex1}. It can be seen that this is in fact the only answer set of P_{ex1}.

In general, a FASP program may have infinitely many answer sets. Rather than trying to enumerate them all, we are interested in approximating them by removing interpretations that cannot be answer sets.

3 Search space pruning operators

To approximate the answer sets of a normal FASP program P, we will construct two interpretations $L, U \in \mathscr{I}$, called *lower* and *upper bound*, respectively, such that $L \subseteq U$. The idea is to find the tightest interpretations L and U such that $L \subseteq M \subseteq U$ holds for every answer set M of P. Let $\mathbf{0}$ and $\mathbf{1}$ be the interpretations that map every propositional atom to 0 and 1, respectively. Obviously, $L = \mathbf{0}$ and $U = \mathbf{1}$ satisfy the desired condition of bounding all answer sets for any program P. Before defining operators for improving these bounds, we introduce a convenient notion for the *partial evaluation* of a rule r w.r.t. $\langle I, J\rangle$, where $I, J \in \mathscr{I}$ are fuzzy interpretations:

$$\langle I, J\rangle(r) := I(B^+(r)) \otimes J(not\, B^-(r)). \tag{2}$$

Lower bounds can be improved by the well-known immediate consequence operator.

Definition 1 (*Immediate consequence operator*)
The *immediate consequence operator* of a program P w.r.t. an upper bound U is the function $T_P^U : \mathscr{I} \to \mathscr{I}$ where $[T_P^U(L)](a) := \max\{\langle L, U\rangle(r) \mid r \in P,\ H(r) = a\}$ for each atom $a \in \mathscr{B}$.

https://doi.org/10.1017/S1471068413000471 Published online by Cambridge University Press

The operator T_P^U is monotonic, and thus it has a least fixpoint $T_P^U \Uparrow \mathbf{0}$, i.e., the sequence $L_0 := \mathbf{0}$, $L_{i+1} := T_P^U(L_i)$ $(i \geqslant 0)$ converges to $T_P^U \Uparrow \mathbf{0}$.

Theorem 1
Let $U \in \mathscr{I}$, and P be a program. The fixpoint $T_P^U \Uparrow \mathbf{0}$ is reached after a linear number of iterations, measured on the number of atoms appearing in P.

If P is positive, $T_P^{\mathbf{1}} \Uparrow \mathbf{0}$ coincides with the unique answer set of P (Lukasiewicz 2006). Hence, by Theorem 1, this answer set is computable in polynomial time. For normal programs, instead, the lower bound can be improved replacing L by $T_P^U \Uparrow \mathbf{0}$. To improve the upper bound, we use an idea from classical ASP: the truth of an atom a in an answer set M must be supported by some rule, i.e., $M(a)$ cannot be larger than the maximum of $M(B(r))$ over all rules $r \in P$ with $H(r) = a$. Support must also be acyclic, or *founded*.

Definition 2 (*Fuzzy unfounded set*)
Let $L, U \in \mathscr{I}$, $L \subseteq U$, and P be a program. A fuzzy set $X \in \mathscr{I}$ is a *fuzzy unfounded set* for P w.r.t. (L, U) if for each $r \in P$ such that $X(H(r)) > 0$, the following inequality is satisfied: $[U \cap (\mathbf{1} \setminus X)](H(r)) >= \langle U \cap (\mathbf{1} \setminus X), L \rangle (r)$.

Intuitively, fuzzy unfounded sets evidence lack of (acyclic) support.

Theorem 2
For FASP programs without numeric constants and crisp sets, Definition 2 coincides with the original notion of unfounded set by Van Gelder *et al.* (1991).

As in the crisp case, the union of two (fuzzy) unfounded sets is an unfounded set.

Theorem 3
Let X_1, X_2 be two fuzzy unfounded sets for P w.r.t. (L, U). Then also $X_1 \cup X_2$ is a fuzzy unfounded set for P w.r.t. (L, U).

We can thus define the greatest fuzzy unfounded set, denoted $GUS_P^{L,U}$, as the union of all fuzzy unfounded sets. We also highlight a relationship with fuzzy answer sets.

Theorem 4
M is a fuzzy answer set of a program P if and only if $GUS_P^{M,M} = \mathbf{1} \setminus M$.

In order to find unfounded sets, we can employ the operator R_P, defined next.

Definition 3 (*Operator* R_P)
Let $L, U \in \mathscr{I}$. The *operator* R_P for a program P w.r.t. (L, U) is the function $R_P^{L,U} : \mathscr{I} \to \mathscr{I}$ such that $[R_P^{L,U}(X)](a) := \min\{X(a), 1 - \max\{\langle U \cap (\mathbf{1} \setminus X), L \rangle (r) \mid r \in P,\ H(r) = a\}\}$ for every $a \in \mathscr{B}$.

The operator $R_P^{L,U}$ is antitonic; thus it has a greatest fixpoint $R_P^{L,U} \Downarrow \mathbf{1}$, which is the limit of the sequence $X_0 := \mathbf{1}$, $X_{i+1} := R_P^{L,U}(X_i), i >= 0$. There is a strong relationship between fuzzy unfounded sets and the operator R_P, which allows for replacing the upper bound U by $\mathbf{1} \setminus (R_P^{L,U} \Downarrow \mathbf{1})$.

https://doi.org/10.1017/S1471068413000471 Published online by Cambridge University Press

Theorem 5
Let $L, U \in \mathscr{I}$, $L \subseteq U$, and P be a program. If $\mathbf{1} \setminus R_P^{L,U}\Downarrow\mathbf{1} \subseteq U$, then $R_P^{L,U}\Downarrow\mathbf{1} = GUS_P^{L,U}$.

It is easy to see that one application of the R_P operator requires linear time in the number of rules P. Moreover, as in Theorem 1, the greatest fixpoint $R_P^{L,U}\Downarrow\mathbf{1}$ is obtained after at most as many applications of this operator as there are atoms in P. In total, this means that this fixpoint can be computed in polynomial time on the size of P.

Theorem 6
Let $L, U \in \mathscr{I}$ and P be a program. The fixpoint $R_P^{L,U} \Downarrow \mathbf{1}$ can be computed in polynomial time on the size of P.

To sum up, the T_P operator can be used to improve the lower bound L, while R_P is useful for decreasing the upper bound U. These two operators complement each other, as a tighter lower bound may help to further decrease the upper bound, and dually, a tighter upper bound may increase the lower bound. In fact, if $L \subseteq L'$ and $U' \subseteq U$, then (i) $T_P^U(I) \subseteq T_P^{U'}(I)$ and (ii) $R_P^{L',U}(I) \subseteq R_P^{L,U}(I)$, thus $\mathbf{1} \setminus R_P^{L,U}(I) \subseteq \mathbf{1} \setminus R_P^{L',U}(I)$ hold for every interpretation I. We can then combine T_P and R_P to obtain a new operator.

Definition 4 (*Well-founded operator*)
The *well-founded operator* is the function $W_P : \mathscr{I} \times \mathscr{I} \to \mathscr{I} \times \mathscr{I}$ defined as

$$W_P(L, U) := \left(T_P^U(L), \mathbf{1} \setminus R_P^{L,U}\Downarrow\mathbf{1}\right). \tag{3}$$

The well-founded operator is monotonic in the lattice $(\mathscr{I} \times \mathscr{I}, \leqslant)$, where $(L, U) \leqslant \left(L', U'\right)$ if and only if $L \subseteq L'$ and $U' \subseteq U$. Hence, W_P has a least fixpoint $W_P\Uparrow(\mathbf{0}, \mathbf{1})$. Moreover, every pair $W_i = (L_i, U_i)$ in the sequence $W_0 := (\mathbf{0}, \mathbf{1})$, $W_{i+1} := W_P(W_i)$ $(i \geqslant 0)$ satisfies that $L_i \subseteq M \subseteq U_i$ for every answer set M of P. We show a stronger result.

Theorem 7
Let P be a program, L, U two interpretations, $(L', U') = W_P(L, U)$, and M an answer set for P. If $L \subseteq M \subseteq U$, then $L' \subseteq M \subseteq U'$.

Just as T_P does for positive programs, an iterative application of the well-founded operator yields the unique answer set of stratified programs in polynomial time.

Theorem 8
Let P be a stratified program. The least fixpoint of W_P coincides with the unique answer set of P and is computable in polynomial time.

Example 2
Consider the stratified program P_{ex1} from Example 1. The application of the well-founded operator to P_{ex1} are shown in Table 1. After three iterations, a fixpoint is reached, stating that every answer set M for P_{ex1} must be such that $M(a) = 0$, $M(b) = 0.9$, and $M(c) = 0.1$. As seen before, this is indeed the only answer set of P_{ex1}.

https://doi.org/10.1017/S1471068413000471 Published online by Cambridge University Press

Table 1. *Iterations of the combined operator on* P_{ex1} *and* P_{ex2}

	P_{ex1}			P_{ex2}		
	a	b	c	a	b	c
$L_0 := \mathbf{0} \quad ; U_0 := \mathbf{1}$	0; 1	0 ; 1	0 ; 1	0 ; 1	0 ; 1	0 ; 1
$L_1 := T_P^{U_0}(L_0); U_1 := \mathbf{1} \setminus R_P^{L_0,U_0}\Downarrow\mathbf{1}$	0; 0.1	0 ; 1	0.1; 0.1	0 ; 0.1	0 ; 1	0.1; 0.1
$L_2 := T_P^{U_1}(L_1); U_2 := \mathbf{1} \setminus R_P^{L_1,U_1}\Downarrow\mathbf{1}$	0; 0.1	0.9; 0.9	0.1; 0.1	0 ; 0.1	0.9; 1	0.1; 0.1
$L_3 := T_P^{U_2}(L_2); U_3 := \mathbf{1} \setminus R_P^{L_2,U_2}\Downarrow\mathbf{1}$	0; 0	0.9; 0.9	0.1; 0.1	0 ; 0	0.9; 1	0.1; 0.1
$L_4 := T_P^{U_3}(L_3); U_4 := \mathbf{1} \setminus R_P^{L_3,U_3}\Downarrow\mathbf{1}$	0; 0	0.9; 0.9	0.1; 0.1	0 ; 0	1 ; 1	0.1; 0.1
$L_5 := T_P^{U_4}(L_4); U_5 := \mathbf{1} \setminus R_P^{L_4,U_4}\Downarrow\mathbf{1}$	0; 0	0.9; 0.9	0.1; 0.1	0 ; 0	1 ; 1	0.1; 0.1

Although it is only possible to guarantee that the iterative application of the combined operator computes an answer set if the program is stratified, they can also produce the answer sets of cyclic programs, as shown by the following example.

Example 3
For the program $P_{\mathsf{ex2}} = P_{\mathsf{ex1}} \cup \{b \leftarrow not\ a\}$, Table 1 shows the applications of W_P. A fixpoint is reached after four iterations, stating that the only candidate for an answer set is the interpretation M with $M(a) = 0$, $M(b) = 1$, and $M(c) = 0.1$. The reduct of P_{ex2} w.r.t. M is $\{a \leftarrow c \otimes \overline{0},\ b \leftarrow \overline{0.9},\ c \leftarrow \overline{0.1},\ b \leftarrow \overline{1}\}$, for which M is a minimal model. However, the iterative application of W_P might not terminate, as for example for program $P_{\mathsf{inf}} = \{a \leftarrow \overline{0.9} \otimes not\ a\}$ over the product t-norm. The least fixpoint of $W_{P_{\mathsf{inf}}}$ assigns to a degree $9/19$, but ω applications are required.

Even if W_P often provides good bounds, there is still room for improvement. Consider the program $P_{odd} = \{a \leftarrow not\ a\}$. P_{odd} has exactly one answer set M with $M(a) = 1/2$. However, W_P yields the bounds $L = \mathbf{0}$ and $U = \mathbf{1}$. Observe that the rule of P_{odd} states the implicit restriction that a must be evaluated to at least $1/2$ in every model I of P because $I \models P$ implies $I(a) \geqslant I(not\ a) = 1 - I(a)$. Implicit restrictions of this kind might be used to further improve lower bounds.

Definition 5 (*Minimal satisfiability*)
The *minimal satisfiability operator* of a program P w.r.t. an upper bound U is the function $S_P^U : \mathscr{I} \to \mathscr{I}$ where $[S_P^U(L)](a) := \inf\{I(a) \mid I \models P, L \subseteq I \subseteq U\}$ for each $a \in \mathscr{B}$.

Theorem 9
Let $L, U, M \in \mathscr{I}$ and P be a program. If $M \models P$ and $L \subseteq M \subseteq U$, then $S_P^U(L) \subseteq M \subseteq U$.

It is thus possible to improve the bounds by an iterative application of the S_P and R_P operators. It is also easy to see that $T_P^U(L) \subseteq S_P^U(L)$. This in particular means that the lower bound obtained by S_P is always at least as good as the one given by T_P, and in some cases strictly better. This translates not only in better bounds being computed, but also in a lower number of iterations needed to obtain them. Unfortunately, in general it is not clear how to compute the minimal satisfiability operator, as it requires finding optimal values for a possibly complex system of constraints, depending on the t-norm. If we restrict to the Łukasiewicz t-norm, then

https://doi.org/10.1017/S1471068413000471 Published online by Cambridge University Press

S_P reduces to solving a series of linear programming problems. More precisely, for a program P we define a finite system of inequalities $Ł_P$ having, for each rule $r \in P$ of the form (1), one inequality

$$a \geqslant b_1 + \ldots + b_m - b_{m+1} - \ldots - b_n + 1 - m \tag{4}$$

such that all variables $a, b_1, \ldots, b_n$ are restricted to the interval $[0, 1]$. All models of P must satisfy $Ł_P$ and *vice versa*. Thus, we obtain the following result.

Theorem 10
Let P be a program over the Łukasiewicz t-norm, and $L, U \in \mathscr{I}$. For every atom $a \in \mathscr{B}$ it holds that $[S_P^U(L)](a) = \min\{I(a) \mid I satisfies Ł_P \cup \{L(b) \leqslant I(b) \leqslant U(b) \mid b \in \mathscr{B}\}\}$.

In this case, the minimal satisfiability operator can be computed efficiently.

Theorem 11
Let $L, U \in \mathscr{I}$, and P be a program over the Łukasiewicz t-norm. $S_P^U(L)$ is computable in polynomial time w.r.t. the number of rules.

Consider again program P_{odd}. S_P computes the minimal value for a with $a \geqslant 1 - a$, i.e., $1/2$. The lower bound is updated to $1/2$. Then, R_P yields $R_P(L) = 1/2$, and updates the upper bound to $1/2$. Further applications of the operators do not modify L or U, hence $M(a) = 1/2$ is our candidate answer set, which in this case is the correct solution.

4 Implementation and experiment

We developed `fasp`, a prototype handling propositional FASP programs. Programs with variables can be transformed into equivalent propositional FASP programs by means of an almost standard grounding procedure, for example by using `gringo` (Gebser *et al.* 2007), which we extended to deal with numeric constants. The prototype is available at `https://github.com/alviano/fasp.git`. In the input program, numeric constants are specified by writing a decimal or fractional number preceded by a # character. The output of `gringo` is a numeric format which constitutes the input of `fasp`. The output of `fasp` is the fuzzy answer sets approximation obtained by the operators described in Section 3 w.r.t. the t-norm specified by the command-line option `--tnorm=TNORM`, where the currently implemented `TNORM`s are `lukasiewicz` (default), `godel`, and `product`. For the Łukasiewicz t-norm, `fasp` can also produce the bilevel program defined by Blondeel *et al.* (2012), which is encoded for the library `YALMIP` (`http://users.isy.liu.se/johanl/yalmip`) and can be solved by invoking `octave` (`http://www.gnu.org/software/octave/`). The bilevel program is produced after computing the approximating operators if `fasp` is run with `--mode=answer-set`, while `--mode=answer-set-unoptimized` can be used for producing the bilevel program without applying any operator. Hence, if the encoding of program P_{ex2}:

```
a :- c, #0.1.    b :- #0.9.    c :- #0.1.    b :- not a.
```

is written in a file `test.lp`, the fuzzy answer set `a[0], b[1], c[0.1]` is the

https://doi.org/10.1017/S1471068413000471 Published online by Cambridge University Press

Algorithm 1: Fuzzy answer sets approximation

Input : a FASP program P
Output: lower and upper bounds L, U

```
1  begin
2      L := 0;   U := 0;
3      foreach r ∈ P do
4          UpperBoundIncrease(r);
5      foreach constant c̄ occurring in P do
6          L(c̄) := c;   U(c̄) := c;
7          foreach r ∈ P such that c̄ ∈ B⁺(r) do
8              UpperBoundIncrease(r);
9              LowerBoundIncrease(H(r), L(B⁺(r)) ⊗ U(not B⁻(r)));
10     MinimalSatisfiability();
11     return (L, U)
```

Procedure UpperBoundIncrease(r: **rule**)

```
1  begin
2      if U(B⁺(r)) ⊗ L(not B⁻(r)) > U(H(r)) then
3          U(H(r)) := U(B⁺(r)) ⊗ L(not B⁻(r));
4          sp(H(r)) := r;
5          foreach r′ ∈ P such that H(r) ∈ B⁺(r′) do
6              UpperBoundIncrease(r′);
```

output of

```
gringo test.lp | fasp --tnorm=lukasiewicz --mode=answer-set
```

The prototype implements Algorithm 1, where for simplicity we assume that all empty rule bodies are replaced by constant $\overline{1}$. Initially, L and U are set to $\mathbf{0}$. Any numeric constant $\overline{c}$ is treated as a propositional atom whose lower and upper bounds are set to c (line 6). These bounds are propagated as described below (lines 5–9), but first `fasp` computes suitable upper bounds for all atoms (line 4). One could argue that processing constants before other rules could be more reasonable; however, Algorithm 1 runs on the stream provided by `gringo`, which first outputs rules where fuzzy atoms are represented by ids, and only at the end are these ids associated to atom names and numeric constants.

Upper bounds are determined by the fixpoint of R_P. To achieve an efficient implementation of this operator, `fasp` takes advantage of source pointers (Simons *et al.* 2002), a technique largely used in crisp ASP solvers that we adapted to the fuzzy case. A source pointer for an atom a is a rule witnessing the upper bound of a. Numeric constants do not need source pointers, while propositional atoms do. Initially, all source pointers are unset. Procedure UpperBoundIncrease is invoked for each rule $r \in P$ (lines 3–4 of Algorithm 1). The procedure computes the upper bound for $B(r)$ as $d = U(B^+(r)) \otimes L(not\ B^-(r))$. If d is strictly greater than $U(H(r))$, the upper bound of $H(r)$ is set to d and the source pointer of $H(r)$ is set to r

https://doi.org/10.1017/S1471068413000471 Published online by Cambridge University Press

Procedure LowerBoundIncrease(a: **atom**, d: **degree**)

```
1 begin
2     if d > L(a) then
3         L(a) := d;
4         foreach r ∈ P such that a ∈ B⁺(r) do
5             LowerBoundIncrease(H(r), L(B⁺(r)) ⊗ U(not B⁻(r)));
6         foreach r ∈ P such that a ∈ B⁻(r) do
7             UpperBoundDecrease(r);
```

Procedure UpperBoundDecrease(r: **rule**)

```
1 begin
2     if sp(H(r)) = r and U(H(r)) − U(B⁺(r)) ⊗ L(not B⁻(r)) > ϵ then
3         U(H(r)) := U(B⁺(r)) ⊗ L(not B⁻(r));
4         foreach r′ ∈ P such that H(r) ∈ B⁺(r′) do
5             UpperBoundDecrease(r′);
6         foreach r′ ∈ P such that H(r) ∈ B⁻(r′) do
7             LowerBoundIncrease(H(r′), L(B⁺(r′)) ⊗ U(not B⁻(r′)));
8         r′ := arg min_{r″∈P} U(B⁺(r″)) ⊗ L(not B⁻(r″));
9         UpperBoundIncrease(r′);
```

(lines 2–4). This new upper bound for $H(r)$ is propagated in each rule r' in which $H(r)$ occurs as a positive body literal (lines 5–6), possibly increasing the upper bound of $H(r')$ and changing its source pointer to r'. At the end of this process, atoms having upper bound different from 0 have source pointers set. Lower bounds, instead, are given by the fixpoint of T_P, obtained by first processing facts and then numeric constants. Each new lower bound d, say for atom a, can increase the lower bound of the head atom of any rule r in which a occurs as a positive body literal (lines 2–5 of LowerBoundIncrease). More specifically, $H(r)$ has a new lower bound set to $L(B^+(r)) \otimes U(not\ B^-(r))$ if this degree is strictly greater than $L(H(r))$.

Lower and upper bound updates can interact intensively to obtain better bounds, and our system handles these interactions as soon as possible. Whenever the lower bound of an atom a is increased, the system checks whether decreasing the upper bound of the head atom of any rule r in which a occurs as a negative literal is possible (lines 6–7 of LowerBoundIncrease). In particular, this might happen if r is the source pointer of $H(r)$, in which case the upper bound of $H(r)$ might be decreased to $U(B^+(r)) \otimes L(not\ B^-(r))$. (To ensure termination, the upper bound is updated only if the decrease is greater than a fixed constant ϵ.) This propagation is handled by procedure UpperBoundDecrease, which first checks whether other upper bounds have to be decreased (lines 4–5). To this end, only rules in which $H(r)$ occurs as a positive body literal have to be checked, and source pointers allow to skip most of these rules. Once upper bounds have been decreased, the procedure possibly increases the lower bounds of the head atom of any rule r' in which $H(r)$ occurs as a negative body literal (lines 6–7). Finally the procedure determines the best source

https://doi.org/10.1017/S1471068413000471 Published online by Cambridge University Press

Procedure MinimalSatisfiability

```
1 begin
2     while bounds changed do
3         D := ∅;
4         foreach a ∈ 𝔅 do
5             if a ∉ D then
6                 s := solution for linear program for a;
7                 LowerBoundIncrease(a, s(a));
8                 D := D ∪ {b | s(b) = L(b)};
```

Table 2. *Experimental result on* `fasp`*: solved instances and average execution time*

		Timeouts		Average execution time*		
	Tested instances	Unopt.	Optimized	Unopt.	Optimized	Average perc. gain*
Graph Coloring	60	34	0	247.44	34.45 (2.68)	76.43%
Hamiltonian Path	40	33	9	120.51	6.41 (0.02)	81.49%
Stratified	90	10	0	190.07	1.80 (0.02)	96.71%
Odd Cycle	90	33	0	186.94	1.95 (0.03)	97.18%

* Computed on the instances solved by both the approaches.

pointer for $H(r)$ (line 8) and in case increases its upper bound, propagating this information by means of the procedure UpperBoundIncrease (line 9).

When these procedures terminate, all atoms have proper lower and upper bounds. For the Łukasiewicz t-norm, bounds can be further improved by the minimal satisfiability operator S_P implemented by procedure MinimalSatisfiability, which takes advantage of the GLPK library (`http://www.gnu.org/software/glpk/`) for solving linear programs. However, operator S_P could be resource demanding, and thus `fasp` limits its computation as follows: The operator is computed w.r.t. an atom a, thus obtaining a proper lower bound for a which is possibly propagated (lines 6–7). The procedure then skips all atoms for which the solution provided by the simplex algorithm already coincides with their lower bounds and thus witnesses that no improvement is possible for these bounds (line 8). Moreover, as the system of inequalities is almost fixed for the input program, it is computed after reading the program and updated when bounds are changed.

Table 2 reports the result of an experiment on `fasp`. *Graph Coloring* and *Hamiltonian Path* are variants of well-known NP-complete problems. Instances of Graph Coloring are those used in the third ASP Competition (Calimeri *et al.* 2011), while random instances were tested for Hamiltonian Path. For both domains, random constants were added in the body of facts. *Stratified* and *Odd Cycle* are very simple programs consisting of rules $a_{i+1} \leftarrow a_i$, $i = 0, \ldots, n-1$, where n is given by the test cases, and atom a_0 is defined by $a_0 \leftarrow \overline{0.9}$ and by $a_0 \leftarrow not\ a_n$, respectively. The experiment was performed on an Intel Xeon CPU X5365 3.00 GHz with

https://doi.org/10.1017/S1471068413000471 Published online by Cambridge University Press

4 GB of central memory and running Debian 6 with kernel Linux 2.6.32. Memory was limited to 3 GB and execution time to 600 seconds. In this benchmark there is a sensible performance gain due to the approximating operators implemented in `fasp`. In fact, "unoptimized" bilevel programs showed a poor performance, timing out 110 times, while the "optimizied" approach timed out only 9 times (on which also the "unoptimized" approach timed out). Even restricting to the 170 test cases solved by the unoptimized apprach, there is a significant advantage of the optimized approach, evidenced by an average percentage gain of at least 76%. Note that in Table 2 the time required for computing the approximation operators is reported in parentheses and included in the execution time of the optimized approach.

5 Related work

The study of fuzzy extensions of logic programs can be traced back more than two decades (see e.g. Dubois *et al.* 1991). Moreover, compared to classical logic, fuzzy logics offer several additional levels of liberty for the definition of their semantics; namely, the choice of the space of truth degrees, the interpretation of the conjunction, the negation, and even the implication. Hence, we describe only the work that is closest related to ours.

The first generalization of the immediate consequence operator to deal with fuzzy semantics was due to Achs and Kiss (1995) and Achs (1997), albeit exclusively for the Gödel t-norm semantics. Fuzzy answer set semantics were introduced by Lukasiewicz (2006) based on a generalization of the Gelfond-Lifschitz transformation (Damásio and Pereira 2001). It was then shown that positive and stratified programs have a unique answer set, and that it can be obtained by a finite iteration of the T_P operator. However, no further analysis on the number of iterations needed to obtain that answer set was made. It is worth noting that the semantics described by Lukasiewicz (2006) are based on a finite set of truth values, rather than the whole interval $[0, 1]$, as in our case. Nonetheless, the proof ideas can be generalized to arbitrary t-norms over $[0, 1]$ without difficulty. General fuzzy answer set programs have been studied in detail in the last years (Janssen 2011), considering not only general t-norm semantics, but also arbitrary connectives to be used in the head and the body of the rules.

While the complexity of finding fuzzy answer sets is now relatively well understood, there were to-date no solvers available. In an effort to compute answer sets for general programs, a completion method was proposed by Janssen *et al.* (2012). The idea is to transform the program P into a set of fuzzy logic formulas, whose models correspond precisely to the answer sets of P. However, due to the lack of (optimized) fuzzy logic reasoners, this reduction does not allow for an effective implementation. A different approach, specifically designed for the Łukasiewicz t-norm, is to reduce the program P to a bilevel linear programming problem (Blondeel *et al.* 2012). This method, which is also implemented by our system, has important theoretical repercussions, e.g., it can be used to show that disjunctions on rule heads do not add expressivity under this semantics.

https://doi.org/10.1017/S1471068413000471 Published online by Cambridge University Press

Unfounded sets for FASP programs were first defined by Nieuwenborgh *et al.* (2007a); these unfounded sets are defined w.r.t. total interpretations and are actually crisp sets used for characterizing fuzzy answer sets. We will refer to this notion as *crisp unfounded sets*. Our definition is more general: it is given w.r.t. partial interpretations and for fuzzy sets; it is suitable for pruning the search space but also characterizes answer sets. A relationship between the two notions follows by Theorem 4.

Corollary 1
For every fuzzy unfounded set X w.r.t. (I,I), set $\{a \mid X(a) + I(a) > 1\}$ is a crisp unfounded set. For every crisp unfounded set Y there is a fuzzy unfounded set X such that $Y = \{a \mid X(a) + I(a) > 1\}$.

To the best of our knowledge, there had been no previous attempts to generalize the R_P operator as a complementation of unfounded sets. This operator allows for a better approximation of answer sets without spending too many resources. A similar idea was studied by Loyer and Straccia (2009), where a well-founded semantics is used for querying fuzzy logic programs over the Gödel t-norm. A well-founded semantics was also defined by Damásio and Pereira (2001), for which we can prove the following result.

Theorem 12
The fixpoint of W_P gives the well-founded semantics by Damásio and Pereira (2001).

6 Conclusions

We studied the problem of finding answer sets for normal FASP programs with t-norm based semantics. We studied fuzzy variants of the operators T_P and R_P, which bound the class of all answer sets of a FASP program P. These operators, as well as the combined well-founded operator W_P, extend well-known operators from classical ASP to handle fuzzy semantics. As such, our operators preserve many of the properties that make them suitable for practical implementations. In particular, we have shown that one application of W_P requires only polynomial time, measured on the size of P. Moreover, for positive and stratified programs, an iterative application of this operator yields the unique answer set in polynomial time, independently of the t-norm used. For normal FASP programs, which may have none or infinitely many answer sets, this operator is only guaranteed to provide lower and upper approximations for the class of answer sets. Depending on the program and the t-norm used, better bounds can be achieved by combining R_P with a new operator S_P. In particular, S_P is computable in polynomial time for the Łukasiewicz t-norm by solving at most one linear program for each atoms in the program.

We implemented a prototype which applies the operators in an optimized manner, using S_P only when no information can be obtained from W_P. In particular, if the program is stratified, then S_P will never be triggered. The system also keeps track of previous solutions of the set of inequalities introduced by S_P, to avoid trying to optimize atoms whose current bounds are already known to be optimal. It also

https://doi.org/10.1017/S1471068413000471 Published online by Cambridge University Press

takes advantage of other optimizations developed for classical ASP, such as source pointers, to reduce the number of computations needed. The approximation provided by our prototype could aid in the computation of fuzzy answer sets, as evidenced by our experiment.

There are several lines for future work. From the theoretical point of view, we plan to investigate further conditions and operators that allow a precise computation of answer sets. In particular, as finding answer sets for normal FASP programs is NP-hard, we need to develop methods for efficiently dealing with a choice operator. We believe that the completion approach by Janssen *et al.* (2012) can be improved through the introduction of binary selection variables. From the practical point of view, we intend to improve the prototype, which currently relies on an external tool for computing answer sets when the bounds cannot be further improved. One idea for this point would be to implement a completion-based method extended with learning techniques.

References

Achs, Á. 1997. Evaluation strategies of fuzzy datalog. *Acta Cybernetica 13,* 1, 85–102.

Achs, Á. and Kiss, A. 1995. Fuzzy extension of datalog. *Acta Cybernetica 12,* 2, 153–166.

Alviano, M., Faber, W., Leone, N., Perri, S., Pfeifer, G. and Terracina, G. 2011. The disjunctive datalog system DLV. In *Datalog 2.0*, G. Gottlob, Ed. Vol. 6702, Springer Berlin/Heidelberg, 282–301.

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

Blondeel, M., Schockaert, S., De Cock, M. and Vermeir, D. 2012. NP-completeness of fuzzy answer set programming under Łukasiewicz semantics. In *Working Papers of the ECAI-2012 Workshop in Weighted Logics for Artificial Intelligence WL4AI*, L. Godo and H. Prade, Eds. 43–50.

Calimeri, F., Ianni, G., Ricca, F., Alviano, M., Bria, A., Catalano, G., Cozza, S., Faber, W., Febbraro, O., Leone, N., Manna, M., Martello, A., Panetta, C., Perri, S., Reale, K., Santoro, M. C., Sirianni, M., Terracina, G. and Veltri, P. 2011. The third answer set programming competition: Preliminary report of the system competition track. In *11th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2011)*, J. Delgrande and W. Faber, Eds. Lecture Notes in Computer Science, vol. 6645, Springer Berlin/Heidelberg, 388–403.

Damásio, C. V. and Pereira, L. M. 2001. Antitonic logic programs. In *Proceedings of the 6th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'01)*. Springer-Verlag, London, UK, 379–392.

Delgrande, J. P., Schaub, T., Tompits, H. and Woltran, S. 2008. Belief revision of logic programs under answer set semantics. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Eleventh International Conference, KR 2008* Sydney, Australia, September 16-19, 2008, G. Brewka and J. Lang, Eds. 411–421.

Dubois, D., Lang, J. and Prade, H. 1991. Fuzzy sets in approximate reasoning, part 2: Logical approaches. *Fuzzy Sets and Systems 40,* 1, 203–244.

Gebser, M., Kaufmann, B., Neumann, A. and Schaub, T. 2007. Conflict-driven answer set solving. In *Twentieth International Joint Conference on Artificial Intelligence (IJCAI-07)*. Morgan Kaufmann Publishers, 386–392.

Gebser, M., Schaub, T. and Thiele, S. 2007. Gringo : A new grounder for answer set programming. In *Logic Programming and Nonmonotonic Reasoning — 9th International Conference, LPNMR'07*, C. Baral, G. Brewka, and J. Schlipf, Eds. Lecture Notes in Computer Science, vol. 4483, Springer Verlag, Tempe, Arizona, 266–271.

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

Janssen, J. 2011. *Foundations of Fuzzy Answer Set Programming*. PhD thesis, Ghent University.

Janssen, J., Vermeir, D., Schockaert, S. and Cock, M. D. 2012. Reducing fuzzy answer set programming to model finding in fuzzy logics. *Theory and Practice of Logic Programming 12,* 6, 811–842.

Klement, E. P., Mesiar, R. and Pap, E. 2000. *Triangular Norms.* Trends in Logic, Studia Logica Library. Springer-Verlag.

Lierler, Y. and Maratea, M. 2004. Cmodels-2: SAT-based answer set solver enhanced to non-tight programs. In *Proceedings of the 7th International Conference on Logic Programming and Non-Monotonic Reasoning (LPNMR-7)*, V. Lifschitz and I. Niemelä, Eds. LNAI, vol. 2923, Springer, 346–350.

Lifschitz, V. 2002. Answer set programming and plan generation. *Artificial Intelligence 138*, 39–54.

Lin, F. and You, J.-H. 2002. Abduction in logic programming: A new definition and an abductive procedure based on rewriting. *Artificial Intelligence 140,* 1/2, 175–205.

Loyer, Y. and Straccia, U. 2009. Approximate well-founded semantics, query answering and generalized normal logic programs over lattices. *Annals Mathematics and Artificial Intelligence 55,* 3-4, 389–417.

Lukasiewicz, T. 2006. Fuzzy description logic programs under the answer set semantics for the semantic web. In *Proc. 2nd International Conference on Rules and Rule Markup Languages for the Semantic Web (RuleML 2006)*, T. Eiter, E. Franconi, R. Hodgson and S. Stephens, Eds. IEEE Computer Society, 89–96.

Marek, V. W. and Remmel, J. B. 2004. Answer set programming with default logic. In , *Proceedings of the 10th International Workshop on Non-Monotonic Reasoning (NMR 2004)*, Whistler, Canada, June 6-8, 2004, J. P. Delgrande and T. Schaub, Eds. 276–284.

Marek, V. W. and Truszczyński, M. 1999. Stable models and an alternative logic programming paradigm. In *The Logic Programming Paradigm – A 25-Year Perspective*, K. R. Apt, V. W. Marek, M. Truszczyński and D. S. Warren, Eds. Springer Verlag, 375–398.

Niemelä, I. 1999. Logic programming with stable model semantics as constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence 25,* 3–4, 241–273.

Nieuwenborgh, D. V., Cock, M. D. and Vermeir, D. 2007a. Computing fuzzy answer sets using dlvhex. In *Proceedings of the 23rd International Conference on Logic Programming (ICLP 2007)*, Porto, Portugal, September 8-13, 2007, Lecture Notes in Computer Science, vol. 4670, 449–450.

Nieuwenborgh, D. V., Cock, M. D. and Vermeir, D. 2007b. An introduction to fuzzy answer set programming. *Annals Mathematics and Artificial Intelligence 50,* 3-4, 363–388.

Simons, P., Niemelä, I. and Soininen, T. 2002. Extending and implementing the stable model semantics. *Artificial Intelligence 138*, 181–234.

Van Gelder, A., Ross, K. A. and Schlipf, J. S. 1991. The Well-founded semantics for general logic programs. *Journal of the ACM 38,* 3, 620–650.