TLP **12** (6): 811–842, 2012. © Cambridge University Press 2011. This is a work of the U.S. Government and is not subject to copyright protection in the United States.

doi:10.1017/S1471068411000093 First published online 21 June 2011

Reducing fuzzy answer set programming to model finding in fuzzy logics

JEROEN JANSSEN* and DIRK VERMEIR
Department of Computer Science, Vrije Universiteit Brussel Pleinlaan 2, 1050 Brussels, Belgium
(*e-mail:* jeroen.janssen, dirk.vermeir}@vub.ac.be)

STEVEN SCHOCKAERT†
Department of Applied Mathematics and Computer Science, Universiteit Gent Krijgslaan 281, 9000 Ghent, Belgium
(*e-mail:* steven.schockaert@ugent.be)

MARTINE DE COCK‡
Institute of Technology, University of Washington 1900 Commerce Street, Tacoma, WA 98402, USA
(*e-mail:* mdecock@u.washington.edu)

submitted 16 August 2010; revised 17 January 2011; accepted 26 March 2011

Abstract

In recent years, answer set programming (ASP) has been extended to deal with multivalued predicates. The resulting formalisms allow for the modeling of continuous problems as elegantly as ASP allows for the modeling of discrete problems, by combining the stable model semantics underlying ASP with fuzzy logics. However, contrary to the case of classical ASP where many efficient solvers have been constructed, to date there is no efficient fuzzy ASP solver. A well-known technique for classical ASP consists of translating an ASP program P to a propositional theory whose models exactly correspond to the answer sets of P. In this paper, we show how this idea can be extended to fuzzy ASP, paving the way to implement efficient fuzzy ASP solvers that can take advantage of existing fuzzy logic reasoners.

KEYWORDS: fuzzy logic, answer set programming, ASSAT

1 Introduction

Answer set programming (ASP), see, e.g., (Baral 2003) is a form of nonmonotonic reasoning based on the stable model semantics for logic programming (Gelfond and Lifschitz 1988). Intuitively, in ASP, one writes a set of rules (the program) such that certain minimal models (the answer sets) of this program correspond to solutions of the problem of interest.

* Funded by a joint Research Foundation-Flanders (FWO) project.
† Postdoctoral fellow of the Research Foundation-Flanders (FWO).
‡ On leave of absence from Ghent University.

https://doi.org/10.1017/S1471068411000093 Published online by Cambridge University Press

In recent work, logic programming has been extended to handle many different facets of imperfect information. Most notably are the probabilistic (Ng and Subrahmanian 1993; Ng and Subrahmanian 1994; Lukasiewicz 1998; Lukasiewicz 1999; Damásio and Pereira 2000; Fuhr 2000; Baral *et al.* 2007; Straccia 2008) and possibilistic (Alsinet *et al.* 2002; Nicolas *et al.* 2005; Nicolas *et al.* 2006; Bauters *et al.* 2010) extensions to handle uncertainty, the fuzzy extensions (Ishizuka and Kanai 1985; Wagner 1998; Cao 2000; Vojtás 2001; Lukasiewicz 2006; Lukasiewicz and Straccia 2007a; Lukasiewicz and Straccia 2007b; Van Nieuwenborgh *et al.* 2007b; Madrid and Ojeda-Aciego 2008; Straccia 2008; Madrid and Ojeda-Aciego 2009; Saad 2009a), which allow to encode the intensity to which the predicates are satisfied, and, more generally, many-valued extensions (Shapiro 1983; Emden 1986; Kifer and Li 1988; Fitting 1991; Kifer and Subrahmanian 1992; Lakshmanan 1994; Lakshmanan and Sadri 1994; Subrahmanian 1994; Lakshmanan 1997; Lakshmanan and Sadri 1997; Nerode *et al.* 1997; Damásio and Pereira 2001a; Damásio and Pereira 2001b; Lakshmanan and Shiri 2001; Loyer and Straccia 2002; Loyer and Straccia 2003; Damásio and Pereira 2004; Damásio *et al.* 2004; Straccia 2005; Straccia 2006; Damásio *et al.* 2007; Straccia *et al.* 2009). In this paper, we focus on a fuzzy extension of ASP, called fuzzy ASP (FASP), which combines the stable model semantics for logic programming with fuzzy logics. More generally, FASP provides a semantics for logic programs in which the truth of predicates (or propositions) may be graded. Such grades may mean different things in different applications, but often they are related to the intensity to which a given property is satisfied. From an application point of view, this is important because it allows to describe continuous phenomena in a logical setting. Thus, a formalism is obtained in which problems with continuous domains can be modeled with the same ease by which discrete problems can be modeled in classical ASP.

In recent years, efficient solvers for classical ASP have been developed. Some of these are based on the Davis–Putnam–Logemann–Loveland (DPLL) algorithm (Davis and Putnam 1960), such as Smodels (Simons 2000) and DLV (Leone *et al.* 2006), others use ideas from satisfiability (SAT) solving, such as clasp (Gebser *et al.* 2009), while still others directly use SAT solvers to find answer sets, e.g., ASSAT (Lin and Zhao 2004), cmodels (Giunchiglia *et al.* 2004), and pbmodels (Liu and Truszczyński 2005). The SAT-based approaches have been shown to be fast and have the advantage that they can use the high number of efficient SAT solvers that have been released in recent years. The DPLL-based solvers have the advantage that they allow a flexible modeling language, since they are not restricted to what can directly and efficiently be translated to SAT and that they can be optimized for specific types of programs.

Probabilistic ASP can be reduced to classical SAT (Saad 2009b), allowing implementations using regular SAT solvers. Likewise, possibilistic ASP can be reduced to classical ASP (Nicolas *et al.* 2006), which means ASP solvers can be used for solving possibilistic ASP programs.

In the case of fuzzy ASP programs with a finite number of truth values, it has been shown in Van Nieuwenborgh *et al.* (2007a) that FASP can be solved using regular ASP solvers. Unfortunately, to date, no fuzzy ASP solvers or solving methods

https://doi.org/10.1017/S1471068411000093 Published online by Cambridge University Press

have been constructed for programs with infinitely many truth values. Our goal in this paper is to take a first step toward creating such efficient solvers by showing how the idea of translating ASP programs to SAT instances can be generalized to fuzzy answer set programs. In this way, we can create fuzzy answer set solvers that use existing techniques for solving fuzzy satisfiability problems, e.g., based on mixed integer programming (MIP) or other forms of mathematical programming. Specifically, we focus on the ASSAT approach introduced in Lin and Zhao (2004). While translating ASP to SAT is straightforward when ASP programs do not contain cyclic dependencies, called loops, careful attention is needed to correctly cover the important case of programs with loops. The solution presented by ASSAT is based on constructing particular propositional formulas for any loop in the program. In this paper, we pursue a similar strategy where **fuzzy loop formulas** are used to correctly deal with loops. Our main contributions can be summarized as follows:

(1) We define the completion of a fuzzy answer set program in the sense of Van Nieuwenborgh *et al.* (2007b) and show that the answer sets of FASP programs without loops are exactly the models of its completion.
(2) By generalizing the loop formulas from Lin and Zhao (2004), we then show how the answer sets of arbitrary FASP programs can be found, provided that the fuzzy logical connectives are t-norms. We furthermore show how the ASSAT procedure, which attempts to overcome the problem with an exponential number of loops, can be generalized to the fuzzy case.

We furthermore show that the FASP semantics in terms of unfounded sets (Van Nieuwenborgh *et al.* 2007a) coincide with the FASP semantics in terms of fixpoints (see, e.g., Lukasiewicz 2006). This is necessary because the development of loop formulas can more easily be done using the unfounded semantics, while the generalization of the ASSAT procedure is based on the fixpoint semantics.

The structure of the paper is as follows. Section 2 recalls the basic fuzzy logic operators and Section 3 recalls the FASP framework from (Van Nieuwenborgh *et al.* 2007b). In Section 3, we furthermore show that the unfounded semantics and fixpoint semantics of FASP coincide. Next, we define the completion of a FASP program in Section 4 and discuss the problems that occur in programs with loops. Section 5 then shows how these problems can be solved by adding loop formulas to the completion. We illustrate our approach on the problem of placing a set of ATM machines on the roads connecting a set of cities such that each city has an ATM machine nearby in Section 6. The reason for restricting our approach to t-norms is discussed in Section 7. Afterward, in Section 8, we give an overview of related work and then present the conclusions in Section 9.

A preliminary version of this paper appeared in Janssen *et al.* (2008). This paper extends our earlier work by adding proofs, a detailed use case and a discussion on the problems that occur when programs are allowed to contain t-conorms in the body. Furthermore, we improved the presentation by removing the aggregation-based approach that was used in the aforementioned work.

https://doi.org/10.1017/S1471068411000093 Published online by Cambridge University Press

Table 1. *Common fuzzy t-norms and t-conorms over* $([0,1],\leqslant)$

t-norm	t-conorm
$\mathcal{T}_m(x,y) = \min(x,y)$	$\mathcal{S}_m(x,y) = \max(x,y)$
$\mathcal{T}_l(x,y) = \max(0, x+y-1)$	$\mathcal{S}_l(x,y) = \min(x+y,1)$
$\mathcal{T}_p(x,y) = x \cdot y$	$\mathcal{S}_p(x,y) = x+y-x \cdot y$

2 Preliminaries

In general, fuzzy logics are logics whose semantics are defined in terms of variables that can take a truth value from the unit interval $[0,1]$ instead of only the values 0 (false) and 1 (true). Different ways exist to extend the classical logic connectives, leading to different logics with different tautologies and axiomatizations (Novák *et al.* 1999; Hájek 2001). We briefly recall the most important concepts related to fuzzy logic connectives.

A **negator** is a decreasing $[0,1] \to [0,1]$ mapping $\mathcal{N}$ satisfying $\mathcal{N}(0) = 1$ and $\mathcal{N}(1) = 0$. A negator is called **involutive** iff $\forall x \in [0,1] : \mathcal{N}(\mathcal{N}(x)) = x$.

A **triangular norm** (t-norm) is an increasing, commutative, and associative $[0,1]^2 \to [0,1]$ operator $\mathcal{T}$ satisfying $\forall x \in [0,1] : \mathcal{T}(1,x) = x$. Intuitively, this operator corresponds to logical conjunction. In this paper, we restrict ourselves to left-continuous t-norms. As the most commonly used t-norms obey this restriction, this poses no practical constraint.

A **triangular conorm** (t-conorm) is an increasing, commutative, and associative $[0,1]^2 \to [0,1]$ operator $\mathcal{S}$ satisfying $\forall x \in [0,1] : \mathcal{S}(0,x) = x$. Intuitively, it corresponds to logical disjunction.

An **implicator** $\mathcal{I}$ is a $[0,1]^2 \to [0,1]$ operator that is decreasing in its first and increasing in its second argument and satisfies $\mathcal{I}(0,0) = \mathcal{I}(0,1) = \mathcal{I}(1,1) = 1$, $\mathcal{I}(1,0) = 0$, and $\forall x \in [0,1] : \mathcal{I}(x,1) = x$. Every left-continuous t-norm induces a **residual implicator** defined by $\mathcal{I}(x,y) = \sup\{\lambda \in [0,1] \mid \mathcal{T}(x,\lambda) \leqslant y\}$. Furthermore, a left-continuous t-norm $\mathcal{T}$ and its residual implicator $\mathcal{I}$ satisfy the **residuation principle**, i.e., for x, y, z in $[0,1]$, we have that $\mathcal{T}(x,y) \leqslant z$ iff $x \leqslant \mathcal{I}(y,z)$. For any left-continuous t-norm $\mathcal{T}$, its residual implicator $\mathcal{I}$ satisfies

$$\mathcal{I}(x,y) = 1 \text{ iff } x \leqslant y. \tag{1}$$

For a given implicator $\mathcal{I}$, its induced negator is the operator $\mathcal{N}$ defined by $\mathcal{N}(x) = \mathcal{I}(x,0)$. We summarize some common t-norms, t-conorms, residual implicators, and induced negators in Tables 1 and 2.

The biresiduum of a residual implicator $\mathcal{I}$ is denoted as $\approx$ and defined by $x \approx y = \mathcal{T}(\mathcal{I}(x,y), \mathcal{I}(y,x))$. Note that due to equation (1), it follows that $x \approx y$ is always equal to either $\mathcal{I}(x,y)$ or $\mathcal{I}(y,x)$. We denote the particular choice of t-norm and implicator using a subscript, when it is not clear from the context, as in $x \approx_m y = \mathcal{T}_m(\mathcal{I}_m(x,y), \mathcal{I}_m(y,x))$.

A **fuzzy set** A in a universe X is an $X \to [0,1]$ mapping. For $x \in X$, we call $A(x)$ the **membership degree** of x in A. For convenience, we denote with $A = \{a_1^{k_1}, \ldots, a_n^{k_n}\}$

https://doi.org/10.1017/S1471068411000093 Published online by Cambridge University Press

Table 2. *Common residual pairs and induced negators over* $([0,1],\leqslant)$

t-norm	residual implicator	induced negator
$\mathcal{T}_m$	$\mathcal{I}_m(x,y) = \begin{cases} y & \text{if } x > y \\ 1 & \text{otherwise} \end{cases}$	$\mathcal{N}_m(x) = \begin{cases} 0 & \text{if } x > 0 \\ 1 & \text{otherwise} \end{cases}$
$\mathcal{T}_l$	$\mathcal{I}_l(x,y) = \min(1, 1-x+y)$	$\mathcal{N}_l(x) = 1-x$
$\mathcal{T}_p$	$\mathcal{I}_p(x,y) = \begin{cases} y/x & \text{if } x > y \\ 1 & \text{otherwise} \end{cases}$	$\mathcal{N}_p(x) = \mathcal{N}_m(x)$

the fuzzy set A satisfying $A(a_i) = k_i$ for $1 \leqslant i \leqslant n$ and $A(a) = 0$ for $a \notin \{a_1, \ldots, a_n\}$. We use $\mathcal{F}(X)$ to denote the universe of all fuzzy sets in X. The **support** of a fuzzy set A is defined by $supp(A) = \{x \in X \mid A(x) > 0\}$. Inclusion of fuzzy sets in the sense of Zadeh is defined as $A \subseteq B$ iff $\forall x \in X : A(x) \leqslant B(x)$. Finally, in this paper, we will write the **difference** $A \ominus B$ of two fuzzy sets to denote the fuzzy set defined by $(A \ominus B)(x) = \max(0, A(x) - B(x))$.

A **signature** is a tuple $\mathbb{S} = \langle \mathbb{A}, \mathbb{T}, \mathbb{C}, \mathbb{I}, \mathbb{N} \rangle$, with $\mathbb{A}$ a set of atoms (i.e., propositional letters), $\mathbb{T}$ a set of t-norms, $\mathbb{C}$ a set of t-conorms, $\mathbb{I}$ a set of implicators, and $\mathbb{N}$ a set of negators. Additionally, we demand that max and min are in the signature. A **fuzzy formula** over a signature $\mathbb{S}$ then is either an atom, a value from $[0,1]$, or the application of a t-norm or t-conorm from $\mathbb{T}$, respectively, $\mathbb{C}$, to two formulas, the application of an implicator from $\mathbb{I}$ to two formulas, or the application of a negator from $\mathbb{N}$ to a single formula. A **fuzzy theory** over a signature $\mathbb{S}$ is a set of fuzzy formulas over $\mathbb{S}$. An **interpretation** I over a signature $\mathbb{S}$ is a mapping from $\mathbb{A}$ to $[0,1]$. It is extended to fuzzy formulas in a straightforward way, i.e., if F and G are fuzzy formulas, then $I(\mathcal{T}(F,G)) = \mathcal{T}(I(F), I(G))$ (with $\mathcal{T} \in \mathbb{T}$), $I(\mathcal{S}(F,G) = \mathcal{S}(I(F), I(G))$ (with $\mathcal{S} \in \mathbb{C}$), $I(\mathcal{I}(F,G)) = \mathcal{I}(I(F), I(G))$ (with $\mathcal{I} \in \mathbb{I}$), and $I(\mathcal{N}(F)) = \mathcal{N}(I(F))$ (with $\mathcal{N} \in \mathbb{N}$). An interpretation M is a **model** of a fuzzy formula F, denoted $M \models F$, iff $M(F) = 1$. An interpretation M is a model of a fuzzy theory Θ, denoted $M \models \Theta$, iff for each $F \in \Theta$ we have that $M \models F$.

A particular signature leads to a particular fuzzy logic (Hájek 2001). For example, $\mathbb{S} = \langle \mathbb{A}, \{\mathcal{T}_m\}, \{\mathcal{S}_m\}, \{\mathcal{I}_m\}, \{\mathcal{N}_m\} \rangle$ gives rise to Gödel logic, $\mathbb{S} = \langle \mathbb{A}, \{\mathcal{T}_l, \mathcal{T}_m\}, \{\mathcal{S}_l, \mathcal{S}_m\}, \{\mathcal{I}_l\}, \{\mathcal{N}_l\} \rangle$ gives rise to Łukasiewicz logic, and $\mathbb{S} = \langle \mathbb{A}, \{\mathcal{T}_p, \mathcal{T}_m\}, \{\mathcal{S}_p, \mathcal{S}_m\}, \{\mathcal{I}_p\}, \{\mathcal{N}_p\} \rangle$ gives rise to product logic. For example, Łukasiewicz logic is generally considered to be closest in spirit to classical logic, in the sense that many of its important properties are preserved. Another important advantage of Łukasiewicz logic is that the implicator is continuous, which is not the case for Gödel or product logic. Reasoning in this logic can be done using MIP, whereas reasoning in Gödel logic can be done with the help of Boolean SAT solvers. As in the Boolean case, satisfiability checking in these three particular logics is NP-complete.

Finally, we denote the infimum, respectively, supremum of two elements of $[0,1]$ as $a \sqcap b$, respectively, $a \sqcup b$.

https://doi.org/10.1017/S1471068411000093 Published online by Cambridge University Press

3 Fuzzy answer set programming

Over the years, many different FASP formalisms have been developed (Lukasiewicz 2006; Lukasiewicz and Straccia 2007a; Van Nieuwenborgh *et al.* 2007b; Madrid and Ojeda-Aciego 2008; Straccia 2008; Madrid and Ojeda-Aciego 2009; Saad 2009a). Most of these base their semantics on fixpoints or minimal models, in combination with a reduct operation. The approaches described in Loyer and Straccia (2006) and Van Nieuwenborgh *et al.* (2007b), however, are constructed from a generalization of unfounded sets. As the development of loop formulas can be done more elegantly when starting from unfounded sets, and we can show that the fixpoint semantics are equivalent to the semantics proposed in Van Nieuwenborgh *et al.* (2007b), we will base our semantics on the latter framework. However, because the generalization of the ASSAT procedure from Lin and Zhao (2004) is based on fixpoint semantics, in this section, we also show novel links between the unfounded and fixpoint semantics that ensure the correctness of our generalized procedure. First, we recall the main definitions from Van Nieuwenborgh *et al.* (2007b).

A **literal**[1] is either an atom a or a **constant** from $[0, 1]$. An **extended literal** is either a literal (called a **positive extended literal**) or an expression of the form $\mathcal{N}(a)$ (called a **negative extended literal**), with a an atom and $\mathcal{N}$ an arbitrary negator. A **rule** r is of the form

$$r : a \leftarrow \mathcal{T}(b_1, \ldots, b_n),$$

where $n > 0$, a is a literal, $\{b_1, \ldots, b_n\}$ is a set of extended literals, $\mathcal{T}$ is an arbitrary t-norm, and r is a rule label. Furthermore, for convenience, we define $\mathcal{T}(b) = b$ and define $\mathcal{T}(b_1, \ldots, b_n)$ as $\mathcal{T}(b_1, \mathcal{T}(b_2, \ldots))$. The literal a is called the **head**, $H(r)$, of r, while the set $\{b_1, \ldots, b_n\}$ is called the **body**, $B(r)$, of r. We use $Lit^+(B(r))$ to denote the set of positive extended literals from the set $B(r)$. Given a rule r, we denote the t-norm used in its body as $\mathcal{T}_r$; the residual implicator corresponding to $\mathcal{T}_r$ is denoted as $\mathcal{I}_r$. A **constraint** is a rule with a constant in its head, whereas a **fact** is a rule with a constant as its body (and no constant in its head). For convenience, we abbreviate a rule of the form $r : a \leftarrow \mathcal{T}(b, 1)$, with b an extended literal, as $r : a \leftarrow b$.

A **FASP program** P is a finite set of rules. We call a program **simple** if no rule contains negative extended literals. The set of all atoms occurring in P is called the **Herbrand Base** $\mathcal{B}_P$ of P. Note that the Herbrand Base is finite since we assume that no function symbols occur in the terms of the ungrounded program. For any $a \in \mathcal{B}_P$, we define the set P_a as the set of rules with atom a in their head. An **interpretation** I of P is a fuzzy set in $\mathcal{B}_P$. We extend interpretations to constants from $[0, 1]$, extended literals, and rules as follows:

(1) $I(c) = c$ if $c \in [0, 1]$,
(2) $I(\mathcal{N}(l)) = \mathcal{N}(I(l))$ if l is a literal,
(3) $I(a \leftarrow \mathcal{T}(b_1, \ldots, b_n)) = \mathcal{I}_r(\mathcal{T}(I(b_1), \ldots, I(b_n)), I(a))$.

A **model** of a program P is an interpretation I of P such that for each rule $r \in P$ we have $I(r) = 1$.

[1] As usual, we assume that programs have already been grounded.

https://doi.org/10.1017/S1471068411000093 Published online by Cambridge University Press

Note that, although each rule in a FASP program can only have a single t-norm in its body, a rule with mixed t-norms, such as $r : a \leftarrow \mathcal{T}_1(a, \mathcal{T}_2(b, c))$, can easily be simulated by introducing a polynomial number of new literals and rules. In the case of rule r, we need one new literal a' and two new rules $r_1 : a \leftarrow \mathcal{T}_1(a, a')$ and $r_2 : a' \leftarrow \mathcal{T}_2(b, c)$.

Example 1
Consider the program P, which consists of the following set of rules:

$$r_1 : a \leftarrow \mathcal{T}_m(b, c),$$
$$r_2 : b \leftarrow 0.8,$$
$$r_3 : c \leftarrow \mathcal{T}_m(a, \mathcal{N}_l(b)),$$
$$r_4 : 0 \leftarrow \mathcal{T}_l(a, b).$$

Note that rule r_2 is a fact, and rule r_4 is a constraint. The fuzzy sets $I_1 = \{a^0, b^{0.8}, c^0\}$ and $I_2 = \{a^{0.2}, b^{0.8}, c^{0.2}\}$ are interpretations of P. For both interpretations, we have that $I_1(r_1) = I_2(r_1) = \ldots = I_1(r_4) = I_2(r_4) = 1$, i.e., they both are models of the program.

The definition of fuzzy answer sets relies on the notion of **unfounded sets**, studied in (Van Nieuwenborgh *et al.* 2007b), which correspond to sets of "assumption atoms" that have no proper motivation from the program.

Definition 1 (*Unfounded sets (Van Nieuwenborgh et al. 2007b)*)
Let P be a FASP program and let I be an interpretation of P. A set $U \subseteq \mathcal{B}_P$ is called **unfounded** with respect to I iff for each atom $u \in U$ and rule $r \in P_u$ either

(1) $U \cap Lit^+(B(r)) \neq \emptyset$; or
(2) $I(u) > I(B(r))$; or
(3) $I(B(r)) = 0$.

An interpretation I of P is called **unfounded-free** iff $supp(I) \cap U = \emptyset$ for any set U that is unfounded with respect to I.

Intuitively, an unfounded set with respect to an interpretation I of a FASP program P is a set of atoms that obtain a value in I that is not motivated by the rules of the program. The first condition of Definition 1 ensures that the values of the literals in U are justified by the values of literals not in U. The second condition shows that the degree to which a rule can motivate an atom is bounded by the value of its body. The third condition is needed to obtain a proper generalization of the classical definition of unfounded sets (Baral 2003) (see Van Nieuwenborgh *et al.* 2007b for more details).

Example 2
Consider program P and interpretations I_1 and I_2 from Example 1. For I_2, we can see that $U_2 = \{a, c\}$ is an unfounded set, as for rule r_1 and r_3, the only rules with a or c in the head, we have that $Lit^+(B(r_1)) \cap U_2 \neq \emptyset$ and $Lit^+(B(r_3)) \cap U_2 \neq \emptyset$. Since $supp(I_2) \cap U_2 \neq \emptyset$, interpretation I_2 is not unfounded-free. Interpretation I_1, however, is unfounded-free.

https://doi.org/10.1017/S1471068411000093 Published online by Cambridge University Press

As shown in Van Nieuwenborgh *et al.* (2007b), **answer sets** of FASP programs can be defined as the unfounded-free interpretations, which reflects the intuition that each atom in an answer set should have a proper motivation from the program.

Definition 2 (*Answer Set Van Nieuwenborgh et al. 2007b*)
Let P be a FASP program. A model M of P is called an **answer set** iff M is unfounded-free.

Example 3
Consider program P and interpretation I_1 from Example 1. Since we know from Example 2 that I_1 is unfounded-free, it follows that I_1 is an answer set of P.

An alternative definition of answer sets, in terms of fixpoints, exists (see, for example, Lukasiewicz 2006). We will use this to generalize the ASSAT procedure described in Lin and Zhao (2004).

Definition 3 (*Immediate Consequence Operator Lukasiewicz 2006*)
Let P be a FASP program. The **immediate consequence operator** of P is the mapping $\Pi_P : \mathcal{F}(\mathcal{B}_P) \rightarrow \mathcal{F}(\mathcal{B}_P)$ defined by

$$\Pi_P(I)(l) = \sup\{I(B(r)) \mid r \in P_l\}.$$

As shown in Lukasiewicz (2006), for simple programs, this operator is monotonic and thus has a least fixpoint (Tarski 1955), denoted as Π_P^*. For these simple programs, Lukasiewicz (2006) then defines the answer sets of a program as the least fixpoints of this operator. Since this operator is monotonic, the least fixpoint is unique and can be found by iteratively applying Π_P from the interpretation $\emptyset$ until a fixpoint is encountered. For nonsimple programs, Lukasiewicz (2006) defines a reduct operation that transforms a nonsimple program into a simple program.

Definition 4 (*Reduct Lukasiewicz 2006*)
Let P be a FASP program and let $r : a \leftarrow \mathcal{T}(b_1, \ldots, b_m, \mathcal{N}(b_{m+1}), \ldots, \mathcal{N}(b_n))$ be a rule in P, where $(b_1, \ldots, b_m) = Lit^+(B(r))$. The **reduct** of rule r, with respect to an interpretation I, is denoted as r^I, and defined by

$$r^I : a \leftarrow \mathcal{T}(b_1, \ldots, b_m, I(\mathcal{N}(b_{m+1})), \ldots, I(\mathcal{N}(b_n))).$$

The reduct of a program P with respect to an interpretation I is defined as $P^I = \{r^I \mid r \in P\}$.

Example 4
Consider program P and interpretation I_1 from Example 1. The reduct of P with respect to I_1 then is the following program:

$$r_1^{I_1} : a \leftarrow \mathcal{T}_m(b, c),$$
$$r_2^{I_1} : b \leftarrow 0.8,$$
$$r_3^{I_1} : c \leftarrow \mathcal{T}_m(a, 0.2),$$
$$r_4^{I_1} : 0 \leftarrow \mathcal{T}_l(a, b).$$

https://doi.org/10.1017/S1471068411000093 Published online by Cambridge University Press

In the following, we show the novel result that the semantics in terms of fixpoints coincide with those in terms of unfounded sets. An important lemma regarding the immediate consequence operator and reduct is the following.

Lemma 1
Let P be a FASP program. For any interpretation I of P, it holds that $I = \Pi_P(I)$ iff $I = \Pi_{P^I}(I)$.

Proof
Follows trivially by the construction of P^I and Definition 4. □

We now show that any answer set is a fixpoint of the immediate consequence operator.

Lemma 2
Let P be a FASP program. Then, any answer set A of P is a fixpoint of Π_P.

Proof
Let A be an answer set of P. We show that $A(a) = \sup\{A(B(r)) \mid r \in P_a\} = \Pi_P(A)(a)$ for any $a \in \mathcal{B}_P$, from which the stated readily follows. The proof is split into the case for $a \in supp(A)$ and $a \notin supp(A)$. For any $a \in supp(A)$, it must hold that $\{a\}$ is not unfounded with respect to A, meaning that $P_a \neq \emptyset$ and there is some $r \in P_a$ such that $A(a) \leqslant A(B(r))$. Since $A(r) = 1$, it then follows from equation (1) that $A(a) = A(B(r))$. As for any $r' \in P_a$, we have $A(r') = 1$, from equation (1), it also follows that $A(B(r)) = A(a) \geqslant A(B(r'))$. Hence, $A(B(r)) = A(a)$ is the supremum of $\{A(B(r')) \mid r' \in P_a\}$.

The case for $a \notin supp(A)$ is as follows. First, remark that as $A(a) = 0$, it follows from equation (1) and the fact that $A(r) = 1$ for each $r \in P_a$, that $A(B(r)) = 0$. Hence, $A(a) = \sup\{A(B(r)) \mid r \in P_a\}$. □

Second, we show that answer sets can be characterized in terms of fixpoints of the immediate consequence operator.

Proposition 1
Let P be a FASP program. An interpretation A is an answer set of P iff $A = \Pi^*_{P^A}$.

Proof
Let M be a model of P. In Van Nieuwenborgh *et al.* (2007b), it was shown that the least fixpoint of Π_{P^M} must necessarily be unfounded-free (Proposition 4). As any fixpoint of Π_P is a model of P, we only need to show that if M is unfounded-free, it is the least fixpoint of Π_{P^M}.

Suppose $M \neq \Pi^*_{P^M}$. Then, since any unfounded-free model is a fixpoint of Π_{P^M} due to Lemmas 1 and 2, it holds that some set $M' \subset M$ exists such that $M' = \Pi^*_{P^M}$.

Consider then $U = \{u \in \mathcal{B}_P \mid M'(u) < M(u)\}$. Surely, $U \subseteq supp(M)$ and hence $U \cap supp(M) \neq \emptyset$. We now show that U is unfounded with respect to M, leading to a contradiction. First, we show that for any atom $u \in U$ and rule $r \in P_u$ it holds that

$$(Lit^+(B(r)) \cap U = \emptyset) \Rightarrow M(B(r)) < M(u), \tag{2}$$

https://doi.org/10.1017/S1471068411000093 Published online by Cambridge University Press

as follows

$$\begin{aligned}
& & & Lit^+(B(r)) \cap U = \emptyset \\
& \equiv \langle \text{Def. } \cap \rangle & & \nexists l \in Lit^+(B(r)) : l \in U \\
& \equiv \langle \text{Duality } \forall,\exists \rangle & & \forall l \in Lit^+(B(r)) : l \notin U \\
& \equiv \langle \text{Def. } U \rangle & & \forall l \in Lit^+(B(r)) : M(a) = M'(a) \\
& \Rightarrow \langle M(l) = M(l^M) \rangle & & M(B(r^M)) = M'(B(r^M)) \\
& \Rightarrow \langle \text{Monotonicity sup} \rangle & & M(B(r^M)) \leqslant \sup_{r' \in P_a^M} M'(B(r')) \\
& \equiv \langle M' = \Pi_{P^M}(M') \rangle & & M(B(r^M)) \leqslant M'(u) \\
& \Rightarrow \langle u \in U, \text{ Def. } U \rangle & & M(B(r^M)) < M(u).
\end{aligned}$$

Thus, since it follows from the Definition of r^M that $M(B(r^M)) = M(B(r))$, we have shown that equation (2) holds. From this equation, we obtain that

$$(Lit^+(B(r)) \cap U \neq \emptyset) \vee (M(B(r)) < M(u)).$$

Hence,

$$(Lit^+(B(r)) \cap U \neq \emptyset) \vee (M(B(r)) < M(u)) \vee (M(B(r)) = 0),$$

which means U is unfounded with respect to M, a contradiction. □

Example 5
Consider program P and interpretations I_1 and I_2 from Example 1. Computing the least fixpoint of $\Pi_{P^{I_1}}$ and $\Pi_{P^{I_2}}$ can be done by iteratively applying these operators, starting from $\emptyset$, until we find a fixpoint. Hence, for P^{I_1}, we obtain in the first iteration $J_1 = \Pi_{P^{I_1}}(\emptyset) = \{b^{0.8}\}$. The second iteration gives $J_2 = \Pi_{P^{I_1}}(J_1) = \{b^{0.8}\} = J_1$, hence a fixpoint, meaning $\{b^{0.8}\} = I_1$ is the least fixpoint of $\Pi_{P^{I_1}}$. Iteratively applying $\Pi_{P^{I_2}}$ brings us $J_1 = \Pi_{P^{I_2}}(\emptyset) = \{b^{0.8}\}$, which is also a fixpoint of $\Pi_{P^{I_2}}$.

4 Completion of FASP programs

In this section, we show how certain fuzzy answer set programs can be translated to fuzzy theories such that the models of these theories correspond to answer sets of the program and vice versa. Such a correspondence is important as it allows us to find answer sets using fuzzy SAT solvers.

Definition 5 (*Completion of a FASP program*)
Let P be a FASP program. The completion of P, denoted as $comp(P)$, is defined as the following set of fuzzy formulas:

$$\{a \approx (\max\{B(r) \mid r \in P_a\}) \mid a \in \mathcal{B}_P\} \cup \{\mathcal{I}_r(B(r), H(r)) \mid r \in P, H(r) \in [0,1]\},$$

where $\approx$ is the biresiduum of an arbitrary residual implicator and $\mathcal{I}_r$ is the residual implicator of the t-norm used in the body of rule r.

The completion of a program consists of two parts, viz., a part for the literals $\{a \approx (\max\{B(r) \mid r \in P_a\}) \mid a \in \mathcal{B}_P\}$ and a part for constraints $\{\mathcal{I}_r(B(r), H(r)) \mid r \in P, H(r) \in [0,1]\}$. The constraints part simply ensures that all constraints are satisfied. The literal part ensures two things. By definition of the biresiduum and the fact that $\mathcal{I}(a,b) = 1$ iff $I(a) \leqslant I(b)$ for any residual implicator, we have that

https://doi.org/10.1017/S1471068411000093 Published online by Cambridge University Press

$I(a \approx b) = 1$ iff $I(a) \leqslant I(b)$ and $I(b) \leqslant I(a)$. Hence, the literal part of the completion establishes that rules are satisfied and second that the value of the literal is not higher than what is supported by the rule bodies.

Example 6
Consider program P from Example 1. Its completion is the following set of fuzzy propositions:

$$a \approx \mathcal{T}_m(b, c),$$
$$b \approx 0.8),$$
$$c \approx \mathcal{T}_m(a, \mathcal{N}_m(b)),$$
$$\mathcal{I}_l(\mathcal{T}_l(a, b), 0).$$

Note that when applying Definition 5 for a literal l that does not appear in the head of any rule, we get $a \approx \max \emptyset$, where we define $\max \emptyset = 0$.

We can now show that any answer set of a program P is a model of its completion $comp(P)$.

Proposition 2
Let P be a FASP program and let $comp(P)$ be its completion. Then, any answer set of P is a model of $comp(P)$.

Proof
Suppose A is an answer set of P. By Lemma 2, it follows that A is a fixpoint of Π_P; hence, for each $a \in \mathcal{B}_P$, $A(a) = \sup\{A(B(r)) \mid r \in P_a\}$. By construction of $comp(P)$ and the fact that A is a model of P, it then easily follows that $A \models comp(P)$. □

Example 7
Consider program P and interpretation I_1 from Example 1. It is easy to see that I_1 is a model of $comp(P)$.

The reverse of Proposition 2 is not true, in general, which is unsurprising because it is already invalid for classical ASP. The problem occurs for programs with "loops." as shown in the following example.

Example 8
Consider program P and interpretation I_2 from Example 1. We can easily see that I_2 is a model of $comp(P)$, but, as we have seen in Example 2, it is not an answer set of P.

One might wonder whether taking the minimal models of the completion would solve the above problem. The following example shows that the answer is negative.

Example 9
Consider the following program P_{min}:

$$a \leftarrow a,$$
$$p \leftarrow \mathcal{T}_l(\mathcal{N}_l(p), \mathcal{N}_l(a)).$$

https://doi.org/10.1017/S1471068411000093 Published online by Cambridge University Press

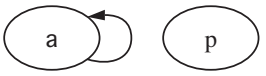


Fig. 1. Dependency graph of program P_{min} from Example 9.

The completion $comp(P_{min})$ is

$$a \approx a,$$
$$p \approx \mathcal{T}_l(\mathcal{N}_l(p), \mathcal{N}_l(a)).$$

Consider now the interpretation $I = \{a^{0.2}, p^{0.4}\}$. Since $I(a) = I(a)$ and

$$\mathcal{T}_l(\mathcal{N}_l(I(p)), \mathcal{N}_l(I(a))) = \max(0, 1 - I(p) + 1 - I(a) - 1) = 0.4,$$

we can see that I is a model of $comp(P_{min})$. We show that it is a minimal model as follows. Suppose some $I' \subset I$ exists. Then, we can consider three cases: (i) $I'(a) < I(a)$ and $I'(p) = I(p)$; (ii) $I'(a) = I(a)$ and $I'(p) < I(p)$; (iii) $I'(a) < I(a)$ and $I'(p) < I(p)$. In all three cases, we obtain that $\mathcal{T}_l(\mathcal{N}_l(I'(p)), \mathcal{N}_l(I'(a))) > 0.4 > I'(p)$, since $\mathcal{N}_l(I'(a)) = 1 - I'(a) > 0.8$ or $\mathcal{N}_l(I'(p)) > 0.6$. Hence, I' is not a model of $comp(P_{min})$ and I is thus a minimal model of $comp(P_{min})$. However, I' is not an answer set of P_{min}, since $\Pi^*_{P^I_{min},=}\{a^0, p^{0.4}\}$.

As in the crisp case however, when a program has no loops in its positive dependency graph, the models of the completion and the answer sets coincide. First, we define exactly what a loop of a FASP program is, and then, we show that this property indeed still holds for FASP.

Definition 6 (*Loop*)
Let P be a FASP program. The **positive dependency graph** of P is a directed graph $G_P = \langle \mathcal{B}_P, D \rangle$ where $(a, b) \in D$ iff $\exists r \in P_a : b \in Lit^+(B(r))$. For ease of notation, we also denote this relation with $(a, b) \in G_P$ for atoms a and b in the Herbrand base of P. We call a nonempty set $L \subseteq \mathcal{B}_P$ a **loop** of P iff for all literals a and b in L, there is a path (with length > 0) from a to b in G_P such that all vertices of this path are elements of L.

Example 10
Consider program P_{min} from Example 9. The dependency graph of P_{min} is pictured in Figure 1. We can see that $\{a\}$ is a loop. If this loop was not in the program, its completion would become

$$a \approx 0,$$
$$p \approx \mathcal{T}_m(\mathcal{N}_m(p), \mathcal{N}_m(a)).$$

This fuzzy theory has no models. Since program P_{min} has no answer sets, this means the answer sets coincide with the models of the completion when removing the loop.

Example 11
Consider program P from Example 1. The dependency graph of P is pictured in Figure 2. We can clearly see that there is a loop between nodes a and c. Due to this loop, the values of a and c are not sufficiently constrained in the completion.

https://doi.org/10.1017/S1471068411000093 Published online by Cambridge University Press

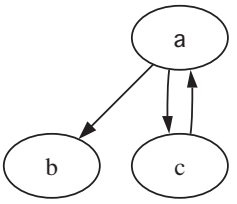


Fig. 2. Dependency graph of program P from Example 1.

From the preceding examples, one might think that removing the loops from the program would be sufficient to make the models of the completion and the answer sets coincide. However, this is not the case, as the semantics of the program then changes, as illustrated in the following example.

Example 12
Consider program P_{change} consisting of the following rules:

$$r_1 : a \leftarrow 0.3,$$
$$r_2 : a \leftarrow b,$$
$$r_3 : b \leftarrow a.$$

Its single answer set is $\{a^{0.3}, b^{0.3}\}$. If we remove rule r_2 or r_3, the answer set of the resulting program is $\{a^{0.3}\}$.

We can now show that for programs without loops the answer sets coincide with the models of their completion. We first introduce two lemmas.

Lemma 3
Let $G = \langle V, E \rangle$ be a directed graph with a finite set of vertices and $X \subseteq V$ with $X \neq \emptyset$. If every node in X has at least one outgoing edge to another node in X, there must be a loop in X.

Proof
From the assumptions, it holds that each $x \in X$ has an outgoing edge to another node in X. This means that there is an infinite sequence of nodes $x_1, x_2, \ldots$ such that $(x_i, x_{i+1}) \in E$ for $i \geqslant 1$. Since X is finite, it follows that some vertex occurs twice in this sequence and hence that there is a loop in X. □

Lemma 4
Let P be a FASP program, I be an interpretation of P and $U \subseteq \mathcal{B}_P$. Then, if $I \models comp(P)$ and U is unfounded with respect to I, it holds that for each u in $U \cap supp(I)$ there is some r in P_u such that $Lit^+(B(r)) \cap U \cap supp(I) \neq \emptyset$.

Proof
Assume that $u \in U \cap supp(I)$, in other words $u \in U$ and $I(u) > 0$. As U is unfounded with respect to I, for each $r \in P_u$, it holds that either

(1) $Lit^+(B(r)) \cap U \neq \emptyset$; or
(2) $I(B(r)) < I(u)$; or
(3) $I(B(r)) = 0$.

https://doi.org/10.1017/S1471068411000093 Published online by Cambridge University Press

We can now show that there is at least one rule $r \in P_u$ that violates the second and third of these conditions, meaning it must satisfy the first.

From $I \models comp(P)$, we know by construction of $comp(P)$ that for each $u \in U$, $I(u) = \sup\{I(B(r)) \mid r \in P_u\}$. Hence, for each $u \in U$, there is a rule $r \in P_u$ such that $I(u) = I(B(r))$; thus, the second condition is violated. Since $I(u) > 0$, it then also follows that the third condition is violated.

In other words, there must be some $r \in P_u$ such that $I(B(r)) = I(u)$ and $I(B(r)) \neq 0$. Since U is unfounded with respect to I, this means that $Lit^+(B(r)) \cap U \neq \emptyset$. Since $I(B(r)) \neq 0$ implies that $Lit^+(B(r)) \subseteq supp(I)$ due to the fact that $\mathcal{T}(0, x) = 0$ for any t-norm $\mathcal{T}$, we can conclude that there is some $r \in P_u$ such that $Lit^+(B(r)) \cap U \cap supp(I) \neq \emptyset$. □

Using these lemmas, we can now show that the answer sets of any program without loops in its dependency graph coincide with the models of its completion. This resembles Fages' theorem on tight programs in classical ASP (Fages 1994).

Proposition 3
Let P be a FASP program. If P has no loops in its positive dependency graph, it holds that an interpretation I of P is an answer set of P iff $I \models comp(P)$.

Proof
We already know from Proposition 2 that any answer set of P is necessarily a model of $comp(P)$; hence, we only need to show that every model of $comp(P)$ is an answer set of P under the conditions of this proposition. As $I \models comp(P)$, it holds that I is a model of P. We show by contradiction that I is unfounded-free. Assume that there is a set $U \subseteq \mathcal{B}_P$ such that U is unfounded with respect to I and $U \cap supp(I) \neq \emptyset$. From Lemma 4, we know that for each $u \in U \cap supp(I)$ it holds that there is some rule $r \in P_u$ such that $Lit^+(B(r)) \cap U \cap supp(I) \neq \emptyset$. Using the definition of G_P, this means that for each such u there is some $u' \in U \cap supp(I)$ such that $G_P(u, u')$. This however means that there is a loop in G_P by Lemma 3, contradicting the assumption. □

Hence, finding the answer sets of a program with no loops in its positive dependency graph can be done by finding models of its completion.

5 Loop formulas

As mentioned in the previous section, sometimes the models of the completion are not answer sets. In this section, we investigate how the solution that has been proposed for Boolean ASP, viz., adding loop formulas to the completion (Lin and Zhao 2004), can be extended to FASP.

For this extension, we start from a partition of the rules whose heads are in some particular loop L. Based upon this partition, for every loop L, we define a formula in fuzzy logic such that any model of the completion satisfying these formulas is an answer set.

https://doi.org/10.1017/S1471068411000093 Published online by Cambridge University Press

For any program P and loop L, we consider the following partition of the rules in P whose head belongs to the set L (due to Lin and Zhao 2004)

$$R_P^+(L) = \{a \leftarrow B \mid ((a \leftarrow B) \in P) \wedge (a \in L) \wedge (B^+ \cap L \neq \emptyset)\}, \tag{3}$$

$$R_P^-(L) = \{a \leftarrow B \mid ((a \leftarrow B) \in P) \wedge (a \in L) \wedge (B^+ \cap L = \emptyset)\}. \tag{4}$$

Note that this partition only takes the positive occurrences of atoms in the loop into account. Intuitively, the set $R_P^+(L)$ contains the rules that are "in" the loop L, i.e., the rules that are jointly responsible for the creation of the loop in the positive dependency graph, whereas the rules in $R_P^-(L)$ are the rules that are outside of this loop. We will refer to them as "loop rules," respectively, "nonloop rules."

Example 13
Consider program P from Example 1. It is clear that for the loop $L = \{a, c\}$, the set of loop rules is $R_P^+(L) = \{r_1, r_3\}$ and the set of nonloop rules is $R_P^-(L) = \emptyset$.

Example 14
Consider program P from Example 1 with interpretations I_1 and I_2 from Example 1 once again. It is clear that in I_1 no loop rules were used to derive the values of a and c, whereas in I_2 only loop rules are used.

Hence, there is a problem when the value of literals in a loop is only derived from rules in the loop. To solve this problem, we should require that at least one nonloop rule motivates the value of these loop literals. As illustrated in the next example, one nonloop rule is sufficient as the value provided by this rule can propagate through the loop by applying loop rules.

Example 15
Consider program P_{change} from Example 12 again. Clearly, this program has a loop $L = \{a, b\}$ with $R_P^+(L) = \{r_2, r_3\}$ and $R_P^-(L) = \{r_1\}$. Consider then interpretations $I_1 = \{a^{0.3}, b^{0.3}\}$ and $I_2 = \{a^1, b^1\}$. We can easily see that I_1 is an answer set of P, whereas I_2 is not although they are both models of $comp(P)$. The problem is that in I_2 the values of a and b are higher than what can be derived from the nonloop rule r_1, whereas in I_1 their values are exactly what can be justified from applying rule r_1. The latter is allowed, as values are properly supported from outside the loop, while the former is not, as in this case the loop is "self-motivating."

To remove the nonanswer set models of the completion, we add loop formulas to the completion, defined as follows.

Definition 7 (*Loop Formula*)
Let P be a FASP program and $L = \{l_1, \ldots, l_m\}$ a loop of P. Suppose that $R_P^-(L) = \{r_1, \ldots, r_n\}$. Then, the loop formula induced by loop L, denoted by $\mathbb{LF}(L, P)$, is the following fuzzy logic formula:

$$\mathcal{I}(\max(l_1, \ldots, l_m), \max(B(r_1), \ldots, B(r_n)), \tag{5}$$

where $\mathcal{I}$ is an arbitrary residual implicator. If $R_P^-(L) = \emptyset$, the loop formula becomes

$$\mathcal{I}(\max(l_1, \ldots, l_m), 0).$$

https://doi.org/10.1017/S1471068411000093 Published online by Cambridge University Press

The loop formula proposed for Boolean answer set programs in Lin and Zhao (2004) is of the form

$$\neg(\bigwedge B(r_1) \vee \ldots \vee \bigwedge B(r_n)) \Rightarrow (\neg l_1 \wedge \ldots \wedge \neg l_m). \tag{6}$$

It can easily be seen that equation (5) is a straightforward generalization of equation (6) as the latter is equivalent to

$$(l_1 \vee \ldots \vee l_m) \Rightarrow (\bigwedge B(r_1) \vee \ldots \vee \bigwedge B(r_n)).$$

Note that this equivalence is preserved in Łukasiewicz logic, but not in Gödel or product logic. Furthermore, since $I \models \mathcal{I}(\max(l_1, \ldots, l_m), 0)$ only when $\max(I(l_1), \ldots, I(l_m)) \leqslant 0$, it is easy to see that in the case where $R_P^-(L) = \emptyset$, the truth value of all atoms in the loop L is 0.

Example 16
Consider program P and interpretations I_1 and I_2 from Example 1. The loop formula for its loop $L = \{a, c\}$ is the fuzzy formula $\mathcal{I}_m(\max(a, c), 0)$, since $R_P^-(L) = \emptyset$. It is easy to see that I_2 does not satisfy this formula, while interpretation I_1 does.

Example 17
Consider program P_{change} from Example 12. The loop formula for its loop $L = \{a, b\}$ is the propositional formula $\mathcal{I}_m(\max(a, b), 0.3)$, since $R_P^-(L) = \{r_1\}$. Again, we see that interpretation I_1 from Example 15 satisfies this loop formula, whereas interpretation I_2 from the same example does not.

We now show that by adding loop formulas to the completion of a program, we get a fuzzy propositional theory that is both sound and complete with respect to the answer set semantics. First, we show that this procedure is complete.

Proposition 4 (*Completeness*)
Let P be a FASP program, let $\mathcal{L}$ be the set of all loops of P, and define $\mathbb{LF}(P) = \{\mathbb{LF}(L, P) \mid L \in \mathcal{L}\}$. For any answer set I of P, it holds that $I \models \mathbb{LF}(P) \cup comp(P)$.

Proof
Suppose I is an answer set of P and $I \not\models \mathbb{LF}(P) \cup comp(P)$. Since any answer set is a model of $comp(P)$ according to Proposition 2, this means that $I \not\models \mathbb{LF}(P)$. Hence, the loop formula of some loop L in P is not fulfilled; this means the following:

$$\sup_{u \in L} I(u) > \sup_{r \in R_P^-(L)} I(B(r)).$$

Consider then the set $U = \{u \in L \mid I(u) > \sup_{r \in R_P^-(L)} I(B(r))\}$. We show that U is unfounded with respect to I, i.e., we show that for each $u \in U$ and rule $r \in P_u$, at least one of the conditions of Definition 1 applies.

Since $P_u = R_{P_u}^+(L) \cup R_{P_u}^-(L)$, each rule $r \in P_u$ must either be in $R_{P_u}^+(L)$ or in $R_{P_u}^-(L)$. We consider the following cases:

(1) If $r \in R_{P_u}^-(L)$ then by construction of U it holds that $I(B(r)) < I(u)$.
(2) If $r \in R_{P_u}^+(L)$ and $I(B(r)) \leqslant \sup_{r' \in R_{P_u}^-(L)} I(B(r'))$, by construction of U, we have that $I(B(r)) < I(u)$.

https://doi.org/10.1017/S1471068411000093 Published online by Cambridge University Press

(3) Suppose $r \in R^+_{P_u}(L)$ and $I(B(r)) > \sup_{r' \in R^-_{P_u}(L)} I(B(r'))$. Since $\mathcal{T}(x, y) \leqslant \min(x, y)$ for each t-norm $\mathcal{T}$, we know that $I(B(r)) \leqslant I(l)$ for each $l \in Lit^+(B(r))$. Hence, for each $l \in Lit^+(B(r))$, we have $I(l) > \sup_{r' \in R^-_{P_u}(L)} I(B(r'))$. This means that, since $r \in R^+_P(L)$ and thus $Lit^+(B(r)) \cap L \neq \emptyset$, we know from the definition of U that $Lit^+(B(r)) \cap U \neq \emptyset$.

Now, remark that $U \cap supp(I) \neq \emptyset$ as $U \subseteq supp(I)$ due to $I(u) > 0$ for each $u \in U$. From the above, we can thus conclude that U is unfounded with respect to I, and since $U \cap supp(I) \neq \emptyset$, that I is not unfounded-free: a contradiction. □

Second, we show that adding the loop formulas to the completion of a program is a sound procedure.

Lemma 5
Let $G = \langle V, E \rangle$ be a directed graph and $X \subseteq V$, with V finite, such that each node of X has at least one outgoing edge to another node in X. Then, there is a set $L \subseteq X$ such that L is a maximal loop in X and for each $l \in L$, we have that there is no $x \in X \setminus L$ for which $(l, x) \in E$.

Proof
From Lemma 3, we already know that there must be a loop in X. Hence, there must also be a maximal loop in X. First, remark that maximal loops must of course be disjoint as otherwise their union would form a bigger loop. Consider then the set X, which is a collection of disjoint maximal loops L and remaining nodes S (single nodes that are not in any loop). There is an induced graph G' of G with nodes $S \cup L$ (i.e., each maximal loop is a single node in the induced graph) and edges E induced as usual (i.e., $(L_1, L_2) \in E$ if for some node l_1 in L_1, there is a node l_2 in L_2 such that $(l_1, l_2) \in E$ and likewise for the nodes in S). Clearly, G' is acyclic as otherwise the nodes in G' on the cycle would create a bigger loop in X. Hence, G' has leafs without outgoing edges. However, a leaf cannot be in S since that would imply a node in X without an outgoing edge. Thus, we can conclude that all leafs in G' are maximal loops in X. □

Proposition 5 (*Soundness*)
Let P be a FASP program and let $\mathbb{LIF}(P)$ be the set of all loop formulas of P. Then, for any interpretation I of P, it holds that if $I \models \mathbb{LIF}(P) \cup comp(P)$, then I must be an answer set of P.

Proof
Suppose $I \models \mathbb{LIF}(P) \cup comp(P)$ and I is not an answer set of P. Since any model of $comp(P)$ must be a model of P, this must mean that I is not unfounded-free, i.e., there exists a set $U \subseteq \mathcal{B}_P$ such that U is unfounded with respect to I. From Lemma 4, we know that for each $u \in U \cap supp(I)$ there must be some $r \in P_u$ such that $Lit^+(B(r)) \cap U \cap supp(I) \neq \emptyset$. Hence, by definition of G_P, this means that for each $u \in U \cap supp(I)$ there is some $u' \in U \cap supp(I)$ such that $(u, u') \in G_P$. Using Lemma 5, this means that there is a set $L \subseteq U \cap supp(I)$ such that L is a loop in P and for each $l \in L$, there is no $u \in (U \cap supp(I)) \setminus L$ such that $(l, u) \in G_P$. In other

https://doi.org/10.1017/S1471068411000093 Published online by Cambridge University Press

words, for each $l \in L$ and rule $r \in P_l$, we have that

$$(U \cap supp(I) \cap Lit^+(B(r)) \neq \emptyset) \Rightarrow (L \cap Lit^+(B(r)) \neq \emptyset). \tag{7}$$

Now, consider $l \in L$. Since $L \subseteq U \cap supp(I)$, we know that $I(l) > 0$. Hence, if $I(B(r)) = I(l)$ for some rule $r \in P_l$, we know that $I(B(r)) > 0$. As U is unfounded with respect to I, it follows from Definition 1 that $L \cap Lit^+(B(r)) \neq \emptyset$.

Using contraposition, this means that for each $l \in L$ and $r \in P_l$ we have that

$$(L \cap Lit^+(B(r)) = \emptyset) \Rightarrow (I(B(r)) \neq I(l)). \tag{8}$$

By the definition of $comp(P)$, however, we know that for each model of $comp(P)$ and for each $a \in \mathcal{B}_P$ and $r \in P_a$ we have $I(a) \geqslant I(B(r))$. Hence, for each $l \in L$ and $r \in P_l$ from equation (8), we have that

$$(L \cap Lit^+(B(r)) = \emptyset) \Rightarrow (I(B(r)) < I(l)). \tag{9}$$

Now, for each $l \in L$ and $r \in R_P^-(L) \cap P_l$ by definition of $R_P^-(L)$, it holds that $L \cap Lit^+(B(r)) = \emptyset$, meaning $I(B(r)) < I(l)$. Thus, $\sup\{I(B(r)) \mid r \in R_P^-(L)\} < \sup\{I(l) \mid l \in L\}$, meaning $I \not\models \mathbb{LLF}(L, P)$, a contradiction. □

A straightforward procedure for finding answer sets would now be to extend the completion of a program with all possible loop formulas and let a fuzzy SAT solver generate models of the resulting fuzzy propositional theory. The models of this theory are the answer sets of the program, as ensured by Propositions 4 and 5. As there may be an exponential number of loops, however, this translation is not polynomial in general. A similar situation arises for classical ASP. The solution proposed in Lin and Zhao (2004) overcomes this limitation by iteratively adding loop formulas. In particular, a SAT solver is first used to find a model of the completion of a classical ASP program. Then, it is checked in polynomial time whether this model is an answer set. If this is not the case, a loop formula, which is not satisfied by the model that was found, is added to the completion. The whole process is then repeated until an answer set is found. We will show that a similar procedure can be used to find answer sets of a FASP program.

Starting from the fixpoint characterization of answer sets of FASP programs, we show that for any given model of the completion that is not an answer set, we can construct a loop that is violated.

Proposition 6
Let P be a FASP program. If an interpretation I of P is a model of $comp(P)$ and $I \neq \Pi^*_{P^I}$, then some $L \subseteq supp(I \ominus \Pi^*_{P^I})$ must exist such that $I \not\models \mathbb{LLF}(P, L)$.

Proof
Suppose I is an interpretation of P and $I \models comp(P)$, then from the definition of $comp(P)$ and Lemma 1, we can easily see that I is a fixpoint of Π_{P^I}. Since $I \neq \Pi^*_{P^I}$, some $I' \subset I$ must exist such that $I' = \Pi^*_{P^I}$.

Consider then the set $U = \{u \in \mathcal{B}_P \mid I(u) > I'(u)\}$. It holds that $U = supp(I \ominus I')$ since $I' \subset I$ and thus $U = supp(I \ominus \Pi^*_{P^I})$ by definition of I'. From the proof of Proposition 1, we then also know that for this set U the following property holds:

$$\forall u \in U : \forall r \in P_u : (Lit^+(B(r)) \cap U = \emptyset) \Rightarrow (I(B(r)) < I(u)). \tag{10}$$

https://doi.org/10.1017/S1471068411000093 Published online by Cambridge University Press

We can then show that there is a loop in U, whose loop formula is violated. Since $I = \Pi_{P^I}(I)$, we know from Lemma 1 that $I = \Pi_P(I)$. From the definition of Π_P, this means

$$\forall l \in \mathcal{B}_P : I(l) = \sup\{I(B(r)) \mid r \in P_l\}.$$

Since the supremum is attained because P is finite, we obtain

$$\forall l \in \mathcal{B}_P : \exists r \in P_l : I(l) = I(B(r)).$$

As $U \subseteq \mathcal{B}_P$. this means

$$\forall u \in U : \exists r \in P_u : I(l) = I(B(r)).$$

Using equation (10), it then holds that

$$\forall u \in U : \exists r \in P_u : Lit^+(B(r)) \cap U \neq \emptyset.$$

From the definition of G_P, we thus get

$$\forall u \in U : \exists u' \in U : (u, u') \in G_P.$$

Using Lemma 5, it follows that there is a set $L \subseteq U$ that is a loop in P such that for each $l \in L$ there is no $l' \in U \setminus L$ such that $(l, l') \in E$. In other words, for each $l \in L$, there is no $l' \in U \setminus L$ such that there is a rule $r \in P_l$ for which $l' \in Lit^+(B(r))$. Hence, for each $l \in L$ and rule $r \in P_l$ such that $U \cap Lit^+(B(r)) \neq \emptyset$, it follows that $L \cap Lit^+(B(r)) \neq \emptyset$. From (10) and using contraposition, this means there is some $L \subseteq U$ that is a loop in P and for each $l \in L$ and $r \in P_l$ if $L \cap Lit^+(B(r)) = \emptyset$ it must hold that $I(B(r)) < I(l)$. Now, for each $l \in L$ and $r \in R_P^-(L) \cap P_l$ by definition, it holds that $L \cap supp(I) = \emptyset$, meaning $I(B(r)) < I(l)$. Thus, $\sup\{I(B(r)) \mid r \in R_P^-(L)\} < \sup\{I(l) \mid l \in L\}$, meaning $I \not\models \mathbb{LF}(L, P)$. □

Now, we can extend the ASSAT-procedure from Lin and Zhao (2004) to fuzzy answer set programs P. The main idea of this method is to use fuzzy SAT solving techniques to find models of the fuzzy propositional theory that consists of the completion of P, together with the loop formulas of particular maximal loops of P. If a model is found that is not an answer set, then we determine a loop that is violated by the model and add its loop formula to the fuzzy propositional theory, after which the fuzzy SAT solver is invoked again. The algorithm thus becomes as follows:

(1) Initialize $Loops = \emptyset$.
(2) Generate a model M of $comp(I) \cup \mathbb{LF}(P, Loops)$, where $\mathbb{LF}(P, Loops)$ is the set of loop formulas of all loops in *Loops*.
(3) If $M = \Pi^*_{P^M}$, return M as it is an answer set. Else, find the loops occurring in $supp(I \ominus \Pi^*_{P^M})$, add their loop formulas to *Loops* and return to step 2.

The reason that we can expect this process to be efficient is articulated by Proposition 6. Indeed, when searching for violated loops, we can restrict our attention to subsets of $supp(I \ominus \Pi^*_{P^I})$. Although the worst-case complexity of this algorithm is still exponential, in most practical applications, we can expect $supp(I \ominus \Pi^*_{P^I})$ to be small, as well as the number of iterations of the process that is needed before an

https://doi.org/10.1017/S1471068411000093 Published online by Cambridge University Press

answer set is found. In Lin and Zhao (2004), experimental evidence for this claim is provided in the case of classical ASP. Finally, note that the fuzzy SAT solving technique depends on the t-norms used in the program. If only the Łukasiewicz t-norm is used, we can use (bounded) MIP (bMIP) (Hähnle 1994). Since Fuzzy Description Logic Solvers are based on the same techniques as fuzzy SAT solvers, we also know that for the product t-norm we need to resort to bounded mixed integer quadratically constrained programming (bMICQP) (Bobillo and Straccia 2007).

6 Example: The ATM location selection problem

In this section, we illustrate our algorithm on a FASP program modeling a real-life problem. Suppose we are tasked with placing k ATM machines $ATM = \{a_1, \ldots, a_k\}$ on roads connecting n towns $Towns = \{t_1, \ldots, t_n\}$ such that the distance between each town and some ATM machine is minimized, i.e., we aim to find a configuration in which each town has an ATM that is as close as practically possible. To obtain this, we optimize the sum of closeness degrees for each town and ATM. Note that this problem closely resembles the well-known k-center selection problem (see, e.g., Ausiello *et al.* 1999). The difference is that in the k-center problem the ATMs need to be placed in towns, where we allow them to be placed on the roads connecting towns. We can model this problem as an undirected weighted graph $G = \langle V, E \rangle$, where $V = Towns$ is the set of vertices and the edge set E connects two towns if they are directly connected by a road. Given a distance function $d : Towns \times Towns \to \mathbb{R}$ that models the distance between two towns[2], the weight of the edge $(a, b) \in E$ is given by the normalized distance $d(a, b)/d_{sum}$, where $d_{sum} = \sum\{d(t_1, t_2) \mid t_1, t_2 \in Towns\}$.

Since our FASP programs can only have t-norms in rule bodies, we also need to find a way to sum up the distances between towns and ATM machines. By using the *nearness degree*, or closeness degree, which for a normalized distance d is defined as $1 - d$, we can perform summations of distances in our program. To see this, consider the following derivation:

$$\begin{aligned} \mathcal{T}_l(1 - dist_1, 1 - dist_2) &= \max(1 - dist_1 + 1 - dist_2 - 1, 0) \\ &= \max(1 - (dist_1 + dist_2), 0) \\ &= 1 - \min(dist_1 + dist_2, 1). \end{aligned}$$

Hence, by applying the Łukasiewicz t-norm on the nearness degrees, we are summing the distances.

[2] For cities that are not connected the function d models the distance of the shortest path between them.

https://doi.org/10.1017/S1471068411000093 Published online by Cambridge University Press

The program P_{ATM} solving the ATM selection problem is given as follows:

$$\begin{aligned}
gloc: &\quad loc(A,T1,T2) \leftarrow \mathcal{T}_l(conn(T1,T2),\beta)\\
gnear: &\quad locNear(A,T1) \leftarrow \mathcal{N}_l(locNear'(A,T1))\\
gnear': &\quad locNear'(A,T1) \leftarrow \mathcal{T}_l(loc(A,T1,T2),\mathcal{N}_l(near(T1,T2)),\\
&\qquad\qquad locNear(A,T2)),T1 \neq T2\\
nearr: &\quad near(T1,T2) \leftarrow \mathcal{T}_l(conn(T1,T3),near(T1,T3),near(T3,T2))\\
locr: &\quad loc(A,T1,T2) \leftarrow loc(A,T2,T1)\\
atmr: &\quad ATMNear(A,T) \leftarrow \mathcal{T}_l(loc(A,T1,T2),locNear(A,T1),near(T,T1))\\
tDist: &\quad totNear \leftarrow \mathcal{T}_l(\{ATMNear(a,t) \mid a \in ATM, t \in Towns\})
\end{aligned}$$

where

$$\beta = \mathcal{T}_l(\{\mathcal{N}_m(loc(A,T1',T2')) \mid \{T_1',T_2'\} \neq \{T_1,T_2\}\}).$$

Note that, due to grounding, a rule such as *locr* actually corresponds to a set of variable-free rules $\{locr_{a,t_1,t_2} \mid a \in ATM, t_1,t_2 \in Towns\}$. We will keep referring to the specific grounded instance of a rule by the subscript.

Program P_{ATM} consists of a *generate* and *define* part, which for a specific configuration is augmented with an *input* part consisting of facts. The *generate* part consists of the three rules *gloc*, *gnear*, and *gnear'*, which generate a specific configuration of ATMs. The *gloc* rule chooses an edge on which the ATM machine A is placed by guessing a location for an ATM that does not yet has an assigned location, as ensured by the β part of this rule. The *gnear* and *gnear'* rules generate a location on this edge, where A is placed. Rules *gnear* and *gnear'* originate from the constraint $d(a,t_1) = d(t_1,t_2) - d(a,t_2)$, where $d(x,y)$ is the distance between x and y, if ATM a is placed on the edge between t_1 and t_2. Defining $n(x,y)$ as the nearness degree between x and y and noting that $n(a,t_1) = 1-d(a,t_1) = 1-(d(t_1,t_2)-d(a,t_2))$, we can rewrite this constraint in terms of t-norms and nearness degrees as follows:

$$\begin{aligned}
n(a,t_1) &= 1-(d(t_1,t_2)-d(a,t_2))\\
&= 1-(d(t_1,t_2)+(1-d(a,t_2))-1)\\
&= 1-\mathcal{T}_l(d(t_1,t_2),1-d(a,t_2))\\
&= 1-\mathcal{T}_l(1-n(t_1,t_2),n(a,t_2))\\
&= \mathcal{N}_l(\mathcal{T}_l(1-n(t_1,t_2),n(a,t_2))).
\end{aligned}$$

Hence, the bodies of rules *gnear* and *gnear'* ensure that this constraint is satisfied. The reason we need two rules and cannot directly write a rule with body $\mathcal{N}_s(\mathcal{T}_l(loc(A,T1,T2),\mathcal{N}_l(near(T1,T2)),locNear(A,T2))$ is that the syntax does not allow negation in front of arbitrary expressions.

Rule *nearr* recursively defines the degree of closeness between two towns based on the known distances for connected towns. Additionally, since the bodies of rules with the same head are combined using the maximum, the nearness degree obtained by *nearr* is always one minus the distance of the shortest path. The *locr* rule makes sure that if an ATM is located on the edge between town $T1$ and $T2$, it is also recognized as being on the edge between $T2$ and $T1$, as we are working with an undirected graph. The *atmr* rule defines the location between a particular ATM machine and a town. Note that due to rule *locr*, this rule also covers the case when

https://doi.org/10.1017/S1471068411000093 Published online by Cambridge University Press

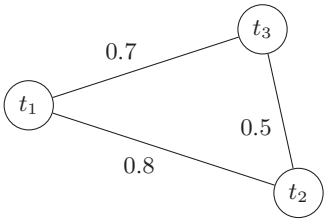


Fig. 3. Town configuration for P_{ATM}. The weights on the edges denote the nearness degrees between towns t_1, t_2, and t_3.

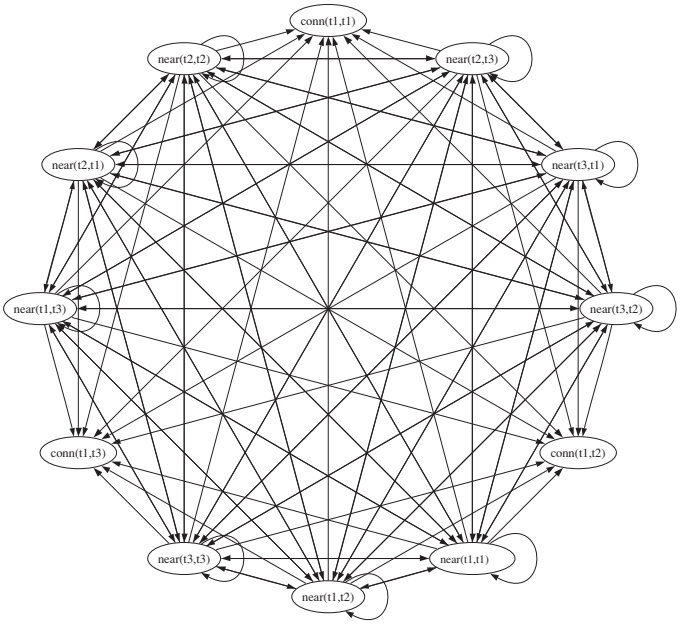


Fig. 4. Dependency graph of P_{ATM}.

near$(T, T2)$ is higher than *near*$(T, T1)$. The *tDist* rule aggregates the total distances such that different answer sets of this program can be compared and ordered. In this way, we could, for example, search for the answer set that has a maximal total degree of nearness, i.e., in which the distance from the towns to the ATMs is lowest.

Consider the specific configuration $G_P = \langle V, E \rangle$ of towns $Towns = \{t_1, t_2, t_3\}$ depicted in Figure 3 and suppose $ATM = \{a_1, a_2\}$. In Figure 4, we depicted a subset of the dependency graph of the grounded version of $P'_{ATM} = P_{ATM} \cup F$, where F is the input part of the problem, given by the following rules:

$$F = \{conn(t, t') \leftarrow 1 \mid t, t' \in Towns, (t, t') \in E\}$$
$$\cup \{near(t, t') \leftarrow k \mid t, t' \in Towns, (t, t') \in E, k = 1 - (d(t, t')/d_{sum})\}.$$

https://doi.org/10.1017/S1471068411000093 Published online by Cambridge University Press

For the configuration depicted in Figure 3, the input part F is

$$\begin{aligned}F =&\{conn(t_1,t_1)\leftarrow 1, conn(t_1,t_2)\leftarrow 1, conn(t_1,t_3)\leftarrow 1\}\\ &\cup\{conn(t_2,t_1)\leftarrow 1, conn(t_2,t_2)\leftarrow 1, conn(t_2,t_3)\leftarrow 1\}\\ &\cup\{conn(t_3,t_1)\leftarrow 1, conn(t_3,t_2)\leftarrow 1, conn(t_3,t_3)\leftarrow 1\}\\ &\cup\{near(t_1,t_1)\leftarrow 1, near(t_1,t_2)\leftarrow 0.8, near(t_1,t_3)\leftarrow 0.7\}\\ &\cup\{near(t_2,t_1)\leftarrow 0.8, near(t_2,t_2)\leftarrow 1, near(t_2,t_3)\leftarrow 0.5\}\\ &\cup\{near(t_3,t_1)\leftarrow 0.7, near(t_3,t_2)\leftarrow 0.5, near(t_3,t_3)\leftarrow 1\}.\end{aligned}$$

It is clear that P'_{ATM} contains a number of loops. The completion of P'_{ATM} is the following fuzzy propositional theory:

$$\begin{aligned}&conn(t_1,t_1)\approx 1,\quad conn(t_1,t_2)\approx 1,\quad conn(t_1,t_3)\approx 1\\ &conn(t_2,t_1)\approx 1,\quad conn(t_2,t_2)\approx 1,\quad conn(t_2,t_3)\approx 1\\ &conn(t_3,t_1)\approx 1,\quad conn(t_3,t_2)\approx 1,\quad conn(t_3,t_3)\approx 1\\ &near(t_1,t_1)\approx 1,\quad near(t_1,t_2)\approx 0.8,\quad near(t_1,t_3)\approx 0.7\\ &near(t_2,t_1)\approx 0.8,\quad near(t_2,t_2)\approx 1,\quad near(t_2,t_3)\approx 0.5\\ &near(t_3,t_1)\approx 0.7,\quad near(t_3,t_2)\approx 0.5,\quad near(t_3,t_3)\approx 1\\ &loc(a_1,t_1,t_1)\approx_l \max(\mathcal{T}_l(conn(t_1,t_1),\beta_{1,1,1}),loc(a_1,t_1,t_1))\\ &loc(a_1,t_1,t_2)\approx_l \max(\mathcal{T}_l(conn(t_1,t_2),\beta_{1,1,2}),loc(a_1,t_2,t_1))\\ &loc(a_1,t_1,t_3)\approx_l \max(\mathcal{T}_l(conn(t_1,t_3,\beta_{1,1,3}),loc(a_1,t_3,t_1))\\ &\qquad\cdots\\ &loc(a_2,t_3,t_1)\approx_l \max(\mathcal{T}_l(conn(t_3,t_1),\beta_{2,3,1}),loc(a_2,t_1,t_3))\\ &loc(a_2,t_3,t_2)\approx_l \max(\mathcal{T}_l(conn(t_3,t_2),\beta_{2,3,2}),loc(a_2,t_2,t_3))\\ &loc(a_2,t_3,t_3)\approx_l \max(\mathcal{T}_l(conn(t_3,t_3),\beta_{2,3,3}),loc(a_2,t_3,t_3))\\ &locNear(a_1,t_1)\approx_l \mathcal{N}_l(locNear'(a_1,t_1))\\ &\qquad\cdots\\ &locNear(a_2,t_3)\approx_l \mathcal{N}_l(locNear'(a_2,t_3))\\ &locNear'(a_1,t_1)\approx_l \max(\mathcal{T}_l(loc(a_1,t_1,t_2),locNear(a_1,t_2),\mathcal{N}_l(near(t_1,t_2))),\\ &\qquad\qquad\mathcal{T}_l(loc(a_1,t_1,t_3),locNear(a_1,t_3),\mathcal{N}_l(near(t_1,t_3))))\\ &\qquad\cdots\\ &locNear'(a_2,t_3)\approx_l \max(\mathcal{T}_l(loc(a_2,t_3,t_1),locNear(a_2,t_1),\mathcal{N}_l(near(t_3,t_1))),\\ &\qquad\qquad\mathcal{T}_l(loc(a_2,t_3,t_2),locNear(a_2,t_2),\mathcal{N}_l(near(t_3,t_2))))\\ &near(t_1,t_1)\approx_l \max(\mathcal{T}_l(conn(t_1,t_1),near(t_1,t_1),near(t_1,t_1)),\\ &\qquad\qquad\mathcal{T}_l(conn(t_1,t_2),near(t_1,t_2),near(t_2,t_1)),\\ &\qquad\qquad\mathcal{T}_l(conn(t_1,t_3),near(t_1,t_3),near(t_3,t_1)),1)\\ &near(t_1,t_2)\approx_l \max(\mathcal{T}_l(conn(t_1,t_1),near(t_1,t_1),near(t_1,t_2)),\\ &\qquad\qquad\mathcal{T}_l(conn(t_1,t_2),near(t_1,t_2),near(t_2,t_2)),\\ &\qquad\qquad\mathcal{T}_l(conn(t_1,t_3),near(t_1,t_3),near(t_3,t_2)),0.8)\end{aligned}$$

https://doi.org/10.1017/S1471068411000093 Published online by Cambridge University Press

$$
\begin{aligned}
&\ldots\\
near(t_3,t_3) \approx_l \max(&\mathcal{T}_l(conn(t_3,t_3), near(t_3,t_3), near(t_3,t_3)),\\
&\mathcal{T}_l(conn(t_3,t_2), near(t_3,t_2), near(t_2,t_3)),\\
&\mathcal{T}_l(conn(t_3,t_1), near(t_3,t_1), near(t_1,t_3)), 1)\\
ATMNear(a_1,t_1) \approx \max(&\mathcal{T}_l(loc(a_1,t_1,t_1), locNear(a_1,t_1), near(t_1,t_1)),\\
&\mathcal{T}_l(loc(a_1,t_1,t_2), locNear(a_1,t_1), near(t_1,t_1)),\\
&\ldots\\
&\mathcal{T}_l(loc(a_1,t_3,t_2), locNear(a_1,t_3), near(t_1,t_3))\\
&\mathcal{T}_l(loc(a_1,t_3,t_3), locNear(a_1,t_3), near(t_1,t_3)))\\
&\ldots\\
ATMNear(a_2,t_3) \approx \max(&\mathcal{T}_l(loc(a_2,t_1,t_1), locNear(a_2,t_1), near(t_3,t_1)),\\
&\mathcal{T}_l(loc(a_2,t_1,t_2), locNear(a_2,t_1), near(t_3,t_1)),\\
&\ldots\\
&\mathcal{T}_l(loc(a_2,t_3,t_2), locNear(a_2,t_3), near(t_3,t_3))\\
&\mathcal{T}_l(loc(a_2,t_3,t_3), locNear(a_2,t_3), near(t_3,t_3)))\\
totNear \approx \mathcal{T}_l\{&ATMNear(a,t) \mid a \in ATM, t \in Towns\}
\end{aligned}
$$

where

$$\beta_{i,j,k} = \mathcal{T}_l(\{\mathcal{N}_m(loc(a_i,t'_j,t'_k)) \mid \{t'_j,t'_k\} \neq \{t_j,t_k\}\}).$$

Note that, e.g., the 1 in the right-hand side of the fuzzy proposition with $near(t_1,t_1)$ on the right-hand side stems from the inputs F we added to P_{ATM}. From the completion $comp(P'_{ATM})$, we can see that an interpretation M satisfying $M(near(t1,t2)) = 1$ can be a model of $comp(P'_{ATM})$, which is clearly unwanted as this would overestimate the nearness degrees between towns (i.e., underestimate the distances). For example, consider

$$
\begin{aligned}
M = \{&loc(a_1,t_1,t_2)^1, loc(a_1,t_2,t_1)^1, loc(a_2,t_1,t_3)^1, loc(a_2,t_3,t_1)^1,\\
&locNear(a_1,t_1)^1, locNear(a_1,t_2)^1, locNear(a_2,t_1)^{0.75}, locNear(a_2,t_3)^{0.75},\\
&locNear'(a_2,t_1)^{0.25}, locNear'(a_2,t_3)^{0.25}, near(t_1,t_1)^1, near(t_1,t_2)^1,\\
&near(t_2,t_1)^1, near(t_1,t_3)^{0.7}, near(t_3,t_1)^{0.7}, near(t_2,t_3)^{0.5}, near(t_3,t_2)^{0.5},\\
&near(t_2,t_2)^1, near(t_3,t_3)^1, ATMNear(a_1,t_1)^1, ATMNear(a_1,t_2)^1,\\
&ATMNear(a_1,t_3)^{0.7}, ATMNear(a_2,t_1)^{0.75}, ATMNear(a_2,t_2)^{0.75},\\
&ATMNear(a_2,t_3)^{0.75}\}.
\end{aligned}
$$

Note that atoms a for which $M(a) = 0$ are not included in the set notation, which is, e.g., the case for $totNear$. One can easily verify that M is a model of $comp(P'_{ATM})$. To check whether M is an answer set, we compute $\Pi^*_{(P'_{ATM})^M}$ by repeatedly applying $\Pi_{(P'_{ATM})^M}$, starting from the empty set, until we obtain a fixpoint, and check whether

https://doi.org/10.1017/S1471068411000093 Published online by Cambridge University Press

$M = \Pi^*_{(P'_{ATM})^M}$. Performing this procedure, we obtain

$$\begin{aligned}\Pi^*_{(P'_{ATM})^M} = \{&loc(a_1,t_1,t_2)^1, loc(a_1,t_2,t_1)^1, loc(a_2,t_1,t_3)^1, loc(a_2,t_3,t_1)^1,\\ &locNear(a_1,t_1)^1, locNear(a_1,t_2)^1, locNear(a_2,t_1)^{0.75},\\ &locNear(a_2,t_3)^{0.75}, locNear'(a_2,t_1)^{0.25}, locNear'(a_2,t_3)^{0.25},\\ &near(t_1,t_1)^1, near(t_1,t_2)^{0.8}, near(t_2,t_1)^{0.8}, near(t_1,t_3)^{0.5},\\ &near(t_3,t_1)^{0.5}, near(t_2,t_3)^{0.7}, near(t_3,t_2)^{0.7}, near(t_2,t_2)^1, near(t_3,t_3)^1,\\ &ATMNear(a_1,t_1)^1, ATMNear(a_1,t_2)^1, ATMNear(a_1,t_3)^{0.7},\\ &ATMNear(a_2,t_1)^{0.75}, ATMNear(a_2,t_2)^{0.75}, ATMNear(a_2,t_3)^{0.75}\}.\end{aligned}$$

We can see that $\Pi^*_{(P'_{ATM})^M}(near(t_1,t_2)) = 0.8 \neq M(near(t_1,t_2))$; hence, M is not an answer set of P'_{ATM}. From Proposition 6, we then know that there must be a loop in $supp(M \ominus \Pi^*_{(P'_{ATM})^M}) = \{near(t_1,t_2), near(t_2,t_1)\}$ whose loop formula is violated. Looking at the dependency graph, we can see that $L = supp(M \ominus \Pi^*_{(P'_{ATM})^M}) = \{near(t_1,t_2), near(t_2,t_1)\}$ contains three loops: $L_1 = L$, $L_2 = \{near(t_1,t_2)\}$, and $L_3 = \{near(t_2,t_1)\}$. Their loop formulas are

$$\begin{aligned}&\mathbb{LF}(L_1, P'_{ATM}) = \mathcal{I}(\max(near(t_1,t_2), near(t_2,t_1)), \max(\mathcal{T}_l(conn(t_1,t_3),\\ &\qquad near(t_1,t_3), near(t_3,t_2)), 0.8, \mathcal{T}_l(conn(t_2,t_3), near(t_2,t_3), near(t_3,t_1)))\\ &\mathbb{LF}(L_2, P'_{ATM}) =\\ &\qquad \mathcal{I}(\max(near(t_1,t_2)), \max(\mathcal{T}_l(conn(t_1,t_3), near(t_1,t_3), near(t_3,t_2)), 0.8)\\ &\mathbb{LF}(L_3, P'_{ATM}) =\\ &\qquad \mathcal{I}(\max(near(t_2,t_1)), \max(\mathcal{T}_l(conn(t_2,t_3), near(t_2,t_3), near(t_3,t_1)), 0.8).\end{aligned}$$

Clearly, these loop formulas are violated by M; hence, following the algorithm introduced in Section 5, we create a new fuzzy propositional theory $comp(P'_{ATM}) \cup \{\mathbb{LF}(L_1, P'_{ATM}), \mathbb{LF}(L_2, P'_{ATM}), \mathbb{LF}(L_3, P'_{ATM})\}$, and try to find a model of this new theory. Consider then the following model of this new theory:

$$\begin{aligned}M = \{&loc(a_1,t_1,t_2)^1, loc(a_1,t_2,t_1)^1, loc(a_2,t_1,t_3)^1, loc(a_2,t_3,t_1)^1,\\ &locNear(a_1,t_1)^{0.15}, locNear(a_1,t_2)^{0.05}, locNear'(a_1,t_1)^{0.85},\\ &locNear'(a_1,t_2)^{0.95} locNear(a_2,t_1)^{0.75}, locNear(a_2,t_3)^{0.75},\\ &locNear'(a_2,t_1)^{0.25}, locNear'(a_2,t_3)^{0.25}, near(t_1,t_1)^1, near(t_1,t_2)^{0.8},\\ &near(t_2,t_1)^{0.8}, near(t_1,t_3)^{0.7}, near(t_3,t_1)^{0.7}, near(t_2,t_3)^{0.5}, near(t_3,t_2)^{0.5},\\ &near(t_2,t_2)^1, near(t_3,t_3)^1, ATMNear(a_1,t_1)^{0.85}, ATMNear(a_1,t_2)^{0.95},\\ &ATMNear(a_1,t_3)^{0.55}, ATMNear(a_2,t_1)^{0.75}, ATMNear(a_2,t_2)^{0.55},\\ &ATMNear(a_2,t_3)^{0.75}\}.\end{aligned}$$

One can readily verify that this model is an answer set of P'_{ATM}; hence, the algorithm stops and returns M.

https://doi.org/10.1017/S1471068411000093 Published online by Cambridge University Press

We could have solved this problem using MIP[3]. However, the exact encoding of this problem would be less clear and straightforward to write. The reason for this is that in the MIP translation the loop formulas would need to be explicitly represented in the program, while in FASP this is handled implicitly. Hence, only the implementer of a FASP system needs to handle these loop formulas, not the developer who writes the FASP programs. This is exactly the power of FASP: providing an elegant, concise, and clear modeling language for representing continuous problems, which, thanks to the results in this paper, can be automatically translated to lower level languages for solving continuous problems, such as MIP.

7 Discussion

The reader might wonder why we limit our approach to FASP programs with t-norms in their body, because at first sight it seems the presented approach is easily extendable to arbitrary functions. It turns out that this is not the case, however. Consider FASP with the Łukasiewicz t-norm in rule bodies. As mentioned before, the completion of such a program and its loop formulas are formulas in Łukasiewicz logic and are implementable using MIP. Now, let us consider FASP where both the Łukasiewicz t-norm and the Łukasiewicz t-conorm may occur in rule bodies. At first, one would suspect that the loop formulas of such a program would again be formulas in Łukasiewicz logic. This turns out to be wrong, however. To see this, consider the following rules:

$$b \leftarrow \mathcal{N}_l(a)$$
$$b \leftarrow \mathcal{S}_l(b, b).$$

One can readily verify that in the answer sets of a program containing these rules, literal b will be equal to $\mathcal{N}_m(a)$ (provided that b does not occur in the head of any other rule). However, the negation $\mathcal{N}_m$ cannot be implemented in MIP, as the solution space of a MIP problem is always a topologically closed set (viz., the union of a finite number of polyhedra), whereas the solution space of a constraint $b \approx \mathcal{N}_m(a)$ cannot be represented as a closed set due to the strict negation in the definition of $\mathcal{N}_m$. This means that as soon as the Łukasiewicz t-conorm is allowed, in general, there will not exist a Łukasiewicz logic theory such that the models of that theory coincide with the answer sets of a given program. Hence, it is clear that the case where other operators than t-norms are used requires a different strategy.

Finding generalized loop formulas that cover, e.g., both the Łukasiewicz t-norm and t-conorm is not a trivial problem. To illustrate some of the issues, let us examine two intuitive candidates. First, remark that the loop formulas introduced in Section 5 eliminate certain answer sets (i.e., they are too strict). Consider the

[3] Though in general, the Gödel negation $\mathcal{N}_m$ cannot be implemented in MIP; in the ATM example, we can implement the *gloc* rules using integer variables.

https://doi.org/10.1017/S1471068411000093 Published online by Cambridge University Press

following program P:

$$a \leftarrow \mathcal{S}_l(a, b),$$
$$b \leftarrow k,$$
$$b \leftarrow a,$$

where $k \in [0, 1]$. This program has a loop, viz., $\{a, b\}$ with corresponding loop formula $\max(a, b) \leqslant k$. Now, note that for $k > 0$ the value of a in any answer set is equal to 1. Hence, the loop formula incorrectly eliminates all answer sets in this case. One might think this can be solved by including a condition in the loop formula: $(\max(a, b) \leqslant l) \vee (b > 0)$. This formula, however, fails to eliminate models that are not answer sets (i.e., it is not strict enough) on the following program:

$$a \leftarrow \mathcal{T}_m(\mathcal{S}_l(a, b), 0.8),$$
$$b \leftarrow \mathcal{S}_l(a, b),$$
$$b \leftarrow k.$$

If $k > 0$, the unique answer set of this program is $\{a^{0.8}, b^1\}$. However, $\{a^1, b^1\}$ is also a model of the completion of this program and satisfies the above loop formula.

Although again more refined loop formulas can be thought of that handle the latter program correctly, we are pessimistic about the possibility of finding loop formulas that cover all cases. It appears that such a general solution should be able to capture some underlying idea of recursion: one loop may justify the truth value of some atom a, up to a certain level, which may then trigger other rules that justify the truth value of a, up to some higher level, etc.

Note that this problem does not occur in classical ASP (or when using the maximum t-conorm), since, e.g., $a \leftarrow b \vee c$ is equivalent to $a \leftarrow b$ and $a \leftarrow c$, which is indeed why disjunctions in the body of rules are not considered in classical ASP.

8 Related work

The approach to FASP for which we provided the translation to fuzzy SAT is called an *unweighted implication-based approach (IB)*. There also exist *weighted IB approaches* (e.g., Lukasiewicz and Straccia 2007a; Madrid and Ojeda-Aciego 2008, 2009, 2011), which use rules of the form

$$r : a \overset{\alpha}{\leftarrow} \mathcal{T}(b_1, \ldots, b_n), \tag{11}$$

where r is a rule label, a is an atom, b_i, for $1 \leqslant i \leqslant n$, are extended literals, and $\alpha \in [0, 1]$. An interpretation I models this rule iff

$$\mathcal{I}_r(\mathcal{T}(I(b_1), \ldots, I(b_n)), I(a)) \geqslant \alpha.$$

Since $\mathcal{I}_r$ is the residual implicator of $\mathcal{T}$, this is equivalent to

$$I(a) \geqslant \mathcal{T}(I(b_1), \ldots, I(b_n), \alpha).$$

Hence, a weighted rule of the form equation (11) above can be simulated by the rule

$$r' : a \leftarrow \mathcal{T}(b_1, \ldots, b_n, \alpha).$$

https://doi.org/10.1017/S1471068411000093 Published online by Cambridge University Press

Thus, the translation presented in this paper can equally be applied to weighted IB approaches.

In addition to the IB approaches, one also finds *annotation-based (AB) approaches* (see, e.g., Straccia 2006). In the AB setting, a rule is of the form

$$A : f(\beta_1, \ldots, \beta_n) \leftarrow B_1 : \beta_1, \ldots, B_n : \beta_n.$$

Such a rule asserts that the value of atom A is at least $f(\beta_1, \ldots, \beta_n)$ if the value of each atom B_i, $1 \leqslant n$ is at least β_i. In this setting, f is a computable function and β_i is either a constant or a variable ranging over an appropriate truth domain. Due to the difference in semantics between the IB and AB approaches, our method is not directly applicable to AB frameworks. One can find an in-depth overview of logic programming with fuzzy logic in Straccia (2008).

In Van Nieuwenborgh *et al.* (2007a), an implementation method for FASP programs with a finite truth value set is presented, which consists of translating a FASP program to a specific DLVHEX program. For solving continuous problems, however, we need infinite truth values, for which a solving method is much harder to construct. Our method is able to handle continuous problems, and additionally is more flexible than Van Niewenborgh *et al.* (2007a) since any method for solving continuous problems can be used as the backend, including fuzzy SAT solvers and the vast body of existing MIP solvers.

Apart from FASP, in recent years possibilistic and probabilistic ASP have been developed. Both of these approaches can be reduced to classical SAT. In the case of probabilistic ASP, there is a direct translation method (Saad 2009b), while a possibilistic ASP program can be translated to an equivalent ASP program, on which the ASSAT procedure can then be applied.

9 Conclusion

In this paper, we have focused on the translation of FASP programs to particular satisfiability problems. We have introduced the completion of a program and have shown that in the case of programs without loops, the models of the completion are exactly the answer sets. Furthermore, to solve the general problem, we have generalized the notion of loop formulas. This translation is important because it allows to solve FASP programs using fuzzy SAT solvers. Under appropriate restrictions, for example, the satisfiability problems that are obtained can be solved using off-the-shelf MIP methods. From an application point of view, this allows us to encode continuous optimization problems in a declarative style, which is similar to traditional ASP. This style of encoding problems is often more intuitive, as well as more concise, while the results we have presented ensure that the power of mathematical programming techniques can still be employed to find the solutions.

Acknowledgement

The authors would like to thank the anonymous reviewers for their useful suggestions and remarks.

https://doi.org/10.1017/S1471068411000093 Published online by Cambridge University Press

References

Alsinet, T., Godo, L. and Sandri, S. 2002. Two formalisms of extended possibilistic logic programming with context-dependent fuzzy unification: A comparative description. *Electronic Notes in Theoretical Computer Science 66,* 5, 1–21.

Ausiello, G., Crescenzi, P., Gambosi, G., Kann, V., Marchetti-Spaccamela, A. and Protasi, M. 1999. *Complexity and Approximation.* Springer-Verlag, Berlin/Heidelberg.

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

Baral, C., Gelfond, M. and Rushton, N. 2007. Probabilistic reasoning with answer sets. In *Proceedings of the Ninth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR '07)*, V. Lifschitz and I. Niemelä, Eds. Lecture Notes in Computer Science, vol. 2923. Springer, Berlin/Heidelberg, 21–33.

Bauters, K., Schockaert, S., De Cock, M. and Vermeir, D. 2010. Possibilistic answer set programming revisited. In *Proceedings of the 26th Conference on Uncertainty in Artificial Intelligence (UAI '10)*, P. Grünwald and P. Spirtes, Eds. AUAI Press.

Bobillo, F. and Straccia, U. 2007. A fuzzy description logic with product t-norm. In *Proceedings of the 16th IEEE International Conference on Fuzzy Systems (FUZZ-IEEE '07)*. IEEE Computer Society, 652–657.

Cao, T. H. 2000. Annotated fuzzy logic programs. *Fuzzy Sets and Systems 113,* 2, 277–298.

Damásio, C. V., Medina, J. and Ojeda-Aciego, M. 2004. Sorted multiadjoint logic programs: Termination results and applications. In *Proceedings of the Ninth European Conference on Logics in Artificial Intelligence (JELIA '04)*, J. J. Alferes and J. Leite, Eds. Lecture Notes in Computer Science, vol. 3229. Springer, Berlin/Heidelberg, 252–265.

Damásio, C. V., Medina, J. and Ojeda-Aciego, M. 2007. Termination of logic programs with imperfect information: Applications and query procedure. *Journal of Applied Logic 5,* 3, 435–458.

Damásio, C. V. and Pereira, L. M. 2000. Hybrid probabilistic logic programs as residuated logic programs. In *Proceedings of the Seventh European Workshop on Logics in Artificial Intelligence (JELIA '00)*, M. Ojeda-Aciego, I. de Guzmán, G. Brewka and L. Moniz Pereira, Eds. Lecture Notes in Computer Science, vol. 1919. Springer, Berlin/Heidelberg, 57–72.

Damásio, C. V. and Pereira, L. M. 2001a. Antitonic logic programs. In *Proceedings of the Sixth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR '01)*, T. Eiter, W. Faber and M. Truszczynski, Eds. Lecture Notes in Computer Science, vol. 2173. Springer, Berlin/Heidelberg, 379–393.

Damásio, C. V. and Pereira, L. M. 2001b. Monotonic and residuated logic programs. In *Proceedings of the Sixth European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU '01)*, S. Benferhat and P. Besnard, Eds. Lecture Notes in Computer Science, vol. 2143. Springer, Berlin/Heidelberg, 748–759.

Damásio, C. V. and Pereira, L. M. 2004. Sorted monotonic logic programs and their embeddings. In *Proceedings of the Information Processing and Management of Uncertainty (IPMU '04)*, 807–814.

Davis, M. and Putnam, H. 1960. A computing procedure for quantification theory. *Journal of the ACM 7,* 3, 201–215.

Fages, F. 1994. Consistency of Clark's completion and existence of stable models. *Methods of Logic in Computer Science 1*, 51–60.

Fitting, M. 1991. Bilattices and the semantics of logic programming. *Journal of Logic Programming 11,* 2, 91–116.

Fuhr, N. 2000. Probabilistic datalog: Implementing logical information retrieval for advanced applications. *Journal of the American Society for Information Science 51,* 2, 95–110.

https://doi.org/10.1017/S1471068411000093 Published online by Cambridge University Press

Gebser, M., Kaufmann, B. and Schaub, T. 2009. The conflict-driven answer set solver clasp: Progress report. In *Proceedings of the 10th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR '09)*, E. Erdem, F. Lin and T. Schaub, Eds. Lecture Notes in Computer Science, vol. 5753. Springer, Berlin/Heidelberg, 509–514.

Gelfond, M. and Lifschitz, V. 1988. The stable model semantics for logic programming. In *Proceedings of the Fifth International Conference and Symposium on Logic Programming (ICLP/SLP '88)*. MIT Press, 1081–1086.

Giunchiglia, E., Lierler, Y. and Maratea, M. 2004. SAT-based answer set programming. In *Proceedings of the 19th national conference on Artifical intelligence (AAAI '04)*. AAAI Press/The MIT Press, 61–66.

Hähnle, R. 1994. Many-valued logic and mixed integer programming. *Annals of Mathematics and Artificial Intelligence 12,* 3–4, 231–263.

Hájek, P. 2001. *Metamathematics of Fuzzy Logic (Trends in Logic)*. Springer.

Ishizuka, M. and Kanai, N. 1985. Prolog-ELF incorporating fuzzy logic. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence (IJCAI '85)*, 701–703.

Janssen, J., Heymans, S., Vermeir, D. and De Cock, M. 2008. Compiling fuzzy answer set programs to fuzzy propositional theories. In *Proceedings of the 24th International Conference on Logic Programming (ICLP '08)*, M. Garcia de la Banda and E. Pontelli, Eds. Lecture Notes in Computer Science, vol. 5366. Springer, Berlin/Heidelberg, 362–376.

Kifer, M. and Li, A. 1988. On the semantics of rule-based expert systems with uncertainty. In *Proceedings of the Second International Conference on Database Theory (ICDT '88)*, M. Gyssens, J. Paredaens and D. Van Gucht, Eds. Lecture Notes in Computer Science, vol. 326. Springer, Berlin/Heidelberg, 102–117.

Kifer, M. and Subrahmanian, V. S. 1992. Theory of generalized annotated logic programming and its applications. *Journal of Logic Programming 12,* 3–4, 335–367.

Lakshmanan, L. V. S. 1994. An epistemic foundation for logic programming with uncertainty. In *Proceedings of the 14th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS '94)*, P. Thiagarajan, Ed. Lecture Notes in Computer Science, vol. 880. Springer, Berlin/Heidelberg, 89–100.

Lakshmanan, L. V. S. 1997. *Towards a Generalized Theory of Deductive Databases with Uncertainty*. PhD Thesis, Concordia University.

Lakshmanan, L. V. S. and Sadri, F. 1994. Modeling uncertainty in deductive databases. In *Proceedings of the Fifth International Conference on Database and Expert Systems Applications (DEXA '94)*, D. Karagiannis, Ed. Lecture Notes in Computer Science, vol. 856. Springer, Berlin/Heidelberg, 724–733.

Lakshmanan, L. V. S. and Sadri, F. 1997. Uncertain deductive databases: A hybrid approach. *Information Systems 22,* 9, 483–508.

Lakshmanan, L. V. S. and Shiri, N. 2001. A parametric approach to deductive databases with uncertainty. *IEEE Transactions on Knowledge and Data Engineering 13,* 4, 554–570.

Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S. and Scarcello, F. 2006. The DLV system for knowledge representation and reasoning. *ACM Transactions on Computational Logic 7,* 3, 499–562.

Lin, F. and Zhao, Y. 2004. ASSAT: Computing answer sets of a logic program by SAT solvers. *Artificial Intelligence 157,* 1–2, 115–137.

Liu, L. and Truszczyński, M. 2005. Pbmodels—software to compute stable models by pseudoboolean solvers. In *Proceedings of the Eighth international conference on Logic Programming and Nonmonotonic Reasoning (LPNMR '05)*, C. Baral, G. Greco, N. Leone and G. Terracina, Eds. Lecture Notes in Computer Science, vol. 3662. Springer, Berlin/Heidelberg, 410–415.

https://doi.org/10.1017/S1471068411000093 Published online by Cambridge University Press

Loyer, Y. and Straccia, U. 2002. The well-founded semantics in normal logic programs with uncertainty. In *Proceedings of the Sixth International Symposium on Functional and Logic Programming (FLOPS '02)*, Z. Hu and M. Rodríguez-Artalejo, Eds. Lecture Notes in Computer Science, vol. 2441. Springer, 152–166.

Loyer, Y. and Straccia, U. 2003. The approximate well-founded semantics for logic programs with uncertainty. In *Proceedings of the 28th International Symposium on Mathematical Foundations of Computer Science (MFCS '03)*, B. Rovan and P. Vojtáš, Eds. Lecture Notes in Computer Science, vol. 2747. Springer, Berlin/Heidelberg, 541–550.

Loyer, Y. and Straccia, U. 2006. Epistemic foundation of stable model semantics. *Journal of Theory and Practice of Logic Programming 6*, 355–393.

Lukasiewicz, T. 1998. Probabilistic logic programming. In *Proceedings of the 13th European Conference on Artificial Intelligence (ECAI '98)*. J. Wiley & Sons, 388–392.

Lukasiewicz, T. 1999. Many-valued disjunctive logic programs with probabilistic semantics. In *Proceedings of the Fifth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR '99)*, M. Gelfond, N. Leone and G. Pfeifer, Eds. Lecture Notes in Computer Science, vol. 1730. Springer, Berlin/Heidelberg, 277–289.

Lukasiewicz, T. 2006. Fuzzy description logic programs under the answer set semantics for the semantic web. In *Proceedings of the Second International Conference on Rules and Rule Markup Languages for the Semantic Web (RuleML '06)*, 89–96.

Lukasiewicz, T. and Straccia, U. 2007a. Tightly integrated fuzzy description logic programs under the answer set semantics for the semantic web. In *Proceedings of the First International Conference on Web Reasoning and Rule Systems (RR '07)*, M. Marchiori, J. Pan and C. Marie, Eds. Lecture Notes in Computer Science, vol. 4524. Springer, Berlin/Heidelberg, 289–298.

Lukasiewicz, T. and Straccia, U. 2007b. Top-k retrieval in description logic programs under vagueness for the semantic web. In *Proceedings of the First International Conference on Scalable Uncertainty Management (SUM '07)*, H. Prade and V. Subrahmanian, Eds. Lecture Notes in Computer Science, vol. 4772. Springer, Berlin/Heidelberg, 16–30.

Madrid, N. and Ojeda-Aciego, M. 2008. Towards a fuzzy answer set semantics for residuated logic programs. In *Proceedings of the 2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT '08)*, 260–264.

Madrid, N. and Ojeda-Aciego, M. 2009. On coherence and consistence in fuzzy answer set semantics for residuated logic programs. In *Proceedings of the Eighth International Workshop on Fuzzy Logic and Applications (WILF '09)*, V. Di Gesù, S. Pal and A. Petrosino, Eds. Lecture Notes in Computer Science, vol. 5571. Springer, Berlin/Heidelberg, 60–67.

Madrid, N. and Ojeda-Aciego, M. 2011. On the existence and unicity of stable models in normal residuated logic programs. *International Journal on Computer Mathematics*, to appear.

Nerode, A., Remmel, J. B. and Subrahmanian, V. S. 1997. Annotated nonmonotonic rule systems. *Theoretical Computer Science 171,* 1–2, 111–146.

Ng, R. and Subrahmanian, V. S. 1993. A semantical framework for supporting subjective and conditional probabilities in deductive databases. *Journal of Automated Reasoning 10,* 2, 191–235.

Ng, R. and Subrahmanian, V. S. 1994. Stable semantics for probabilistic deductive databases. *Information and Computation 110,* 1, 42–83.

Nicolas, P., Garcia, L. and Stéphan, I. 2005. Possibilistic stable models. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI'05)*. Morgan Kaufmann Publishers Inc. San Francisco, CA, USA.

Nicolas, P., Garcia, L., Stéphan, I. and Lefèvre, C. 2006. Possibilistic uncertainty handling for answer set programming. *Annals of Mathematics and Artificial Intelligence 47,* 1–2, 139–181.

https://doi.org/10.1017/S1471068411000093 Published online by Cambridge University Press

Novák, V., Perfilieva, I. and Močkoř, J. 1999. *Mathematical Principles of Fuzzy Logic*. Kluwer Academic Publishers.

Saad, E. 2009a. Extended fuzzy logic programs with fuzzy answer set semantics. In *Proceedings of the Third International Conference on Scalable Uncertainty Management (SUM '09)*, L. Godo and A. Pugliese, Eds. Lecture Notes in Computer Science, vol. 5785. Springer, Berlin/Heidelberg, 223–239.

Saad, E. 2009b. Probabilistic reasoning by SAT solvers. In *Proceedings of the 10th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU '09)*, C. Sossai and G. Chemello, Eds. Lecture Notes in Computer Science, vol. 5590. Springer, Berlin/Heidelberg, 663–675.

Shapiro, E. Y. 1983. Logic programs with uncertainties: A tool for implementing rule-based systems. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence (IJCAI '83)*, A. Bundy, Ed. William Kaufmann, 529–532.

Simons, P. 2000. *Extending and Implementing the Stable Model Semantics*. PhD Thesis, Helsinki University of Technology.

Straccia, U. 2005. Query answering in normal logic programs under uncertainty. In *Proceedings of the Eighth European Conferences on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU '05)*, L. Godo, Ed. Lecture Notes in Computer Science, vol. 3571. Springer, Berlin/Heidelberg, 470–470.

Straccia, U. 2006. Annotated answer set programming. In *Proceedings of the 11th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU '06)*.

Straccia, U. 2008. Managing uncertainty and vagueness in description logics, logic programs and description logic programs. In *Proceedings of the 4th International Summer School 2008 on Reasoning Web*, C. Baroglio, P. A. Bonatti, J. M. uszynski, M. Marchiori, A. Polleres and S. Schaffert, Eds. Lecture Notes in Computer Science, vol. 5224. Springer, Berlin/Heidelberg, 54–103.

Straccia, U., Ojeda-Aciego, M. and Damásio, C. V. 2009. On fixed-points of multivalued functions on complete lattices and their application to generalized logic programs. *SIAM Journal on Computing 38,* 5, 1881–1911.

Subrahmanian, V. S. 1994. Amalgamating knowledge bases. *ACM Transactions on Database Systems 19,* 2, 291–331.

Tarski, A. 1955. A lattice theoretical fixpoint theorem and its application. *Pacific Journal of Mathematics 5*, 285–309.

van Emden, M. H. 1986. Quantitative deduction and its fixpoint theory. *Journal of Logic Programming 30,* 1, 37–53.

Van Nieuwenborgh, D., De Cock, M. and Vermeir, D. 2007a. Computing fuzzy answer sets using DLVHEX. In *Proceedings of the 23rd International Conference on Logic Programming (ICLP '07)*, V. Dahl and I. Niemelä, Eds. Lecture Notes in Computer Science, vol. 4670. Springer, Berlin/Heidelberg, 449–450.

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

Vojtás, P. 2001. Fuzzy logic programming. *Fuzzy Sets and Systems 124,* 3, 361–370.

Wagner, G. 1998. Negation in fuzzy and possibilistic logic programs. *Uncertainty Theory in Artificial Intelligence Series 3*, 113–128.

https://doi.org/10.1017/S1471068411000093 Published online by Cambridge University Press