TLP **15** (4–5): 588–603, 2015. © Cambridge University Press 2015
doi:10.1017/S1471068415000241

Fuzzy answer set computation via satisfiability modulo theories

MARIO ALVIANO
University of Calabria, Italy

RAFAEL PEÑALOZA
Free University of Bozen-Bolzano, Italy

submitted 29 April 2015; revised 3 July 2015; accepted 15 July 2015

Abstract

Fuzzy answer set programming (FASP) combines two declarative frameworks, answer set programming and fuzzy logic, in order to model reasoning by default over imprecise information. Several connectives are available to combine different expressions; in particular the Gödel and Łukasiewicz fuzzy connectives are usually considered, due to their properties. Although the Gödel conjunction can be easily eliminated from rule heads, we show through complexity arguments that such a simplification is infeasible in general for all other connectives. The paper analyzes a translation of FASP programs into satisfiability modulo theories (SMT), which in general produces quantified formulas because of the minimality of the semantics. Structural properties of many FASP programs allow to eliminate the quantification, or to sensibly reduce the number of quantified variables. Indeed, integrality constraints can replace recursive rules commonly used to force Boolean interpretations, and completion subformulas can guarantee minimality for acyclic programs with atomic heads. Moreover, head cycle free rules can be replaced by shifted subprograms, whose structure depends on the eliminated head connective, so that ordered completion may replace the minimality check if also Łukasiewicz disjunction in rule bodies is acyclic. The paper also presents and evaluates a prototype system implementing these translations.

KEYWORDS: answer set programming, fuzzy logic, satisfiability modulo theories.

1 Introduction

Answer set programming (ASP) (Gelfond and Lifschitz 1991; Niemelä 1999; Marek and Truszczyński 1999) is a declarative language for knowledge representation, particularly suitable to model common non-monotonic tasks such as reasoning by default, abductive reasoning, and belief revision (Baral 2003; Marek and Remmel 2004; Lin and You 2002; Delgrande *et al.* 2008). If on the one hand ASP makes logic closer to the real world allowing for reasoning on incomplete knowledge, on the other hand it is unable to model imprecise information that may arise from the intrinsic limits of sensors, or the vagueness of natural language. Fuzzy answer set programming (FASP) (Nieuwenborgh *et al.* 2007) overcomes this limitation by

https://doi.org/10.1017/S1471068415000241 Published online by Cambridge University Press

interpreting propositions with a truth degree in the real interval $[0, 1]$. Intuitively, the higher the degree assigned to a proposition, the *more true* it is, with 0 and 1 denoting *totally false* and *totally true*, respectively. The notion of fuzzy answer set, or fuzzy stable model, was recently extended to arbitrary propositional formulas (Lee and Wang 2014). Lee and Wang also propose an example on modeling dynamic *trust* in social networks, which inspired the following simplified scenario that clarifies how truth degrees increase the knowledge representation capability of ASP.

Example 1
A user of a social network may trust or distrust another user, and these are vague concepts that can be naturally modeled by truth degrees. These degrees may change over time. For example, if at some point A has a conflict with B, it is likely that her distrust on B will increase and her trust on B will decrease. These are non-monotonic concepts that can be naturally handled in FASP.

In practice, however, ASP offers many efficient solvers such as DLV (Alviano *et al.* 2010), CMODELS (Lierler and Maratea 2004), CLASP (Gebser *et al.* 2012), and WASP (Alviano *et al.* 2013), which is not the case for FASP. A preliminary FASP solver for programs with atomic heads and Łukasiewicz conjunction, called FASP, was presented at ICLP'13 by (Alviano and Peñaloza 2013). It implements approximation operators and a translation into bilevel programming (Blondeel *et al.* 2014). A more general solver, called FFASP (Mushthofa *et al.* 2014), is based on a translation into ASP for computing stable models whose truth degrees are in the set $\mathbb{Q}_k := \{i/k \mid i \in [0..k]\}$, for a fixed k. In general, exponentially many k must be tested for checking the existene of a stable model, which is infeasible in practice. Therefore, FFASP tests by default a limited set of values. Neither FASP nor FFASP accept nesting of negation, which would allow to encode *choice rules*, a convenient way for guessing truth degrees without using auxiliary atoms (Lee and Wang 2014). Indeed, choice rules allow to check satisfiability of fuzzy propositional formulas without adding new atomic propositions. Our aim is to provide a more flexible FASP solver supporting useful patterns like choice rules.

Satisfiability modulo theories (SMT) (Barrett *et al.* 2009) extends propositional logic with external background theories—e.g. real arithmetic (Ratschan 2006; Akbarpour and Paulson 2010)—for which specialized methods provide efficient decision procedures. SMT is thus a good candidate as a target framework for computing fuzzy answer sets efficiently. This is non-trivial because the minimality condition that fuzzy stable models must satisfy makes the problem hard for the second level of the polynomial hierarchy; indeed, the translation provided in Section 4 produces quantified theories in general. However, structural properties of the program that decrease the complexity to NP can be taken into account in order to obtain more tailored translations. For example, disabling head connectives and recursive definitions yields a compact translation into fuzzy propositional logic known as *completion* (Janssen *et al.* 2012), which in turn can be expressed in SMT (see Section 4.1). Since completion is unsound for programs with recursive definitions, the notion of *ordered completion* has arisen in the ASP literature (Ben-Eliyahu and Dechter 1994; Janhunen 2004; Niemelä 2008; Asuncion *et al.* 2012). In a nutshell,

https://doi.org/10.1017/S1471068415000241 Published online by Cambridge University Press

stable models of ASP programs with atomic heads can be recasted in terms of program reducts and fixpoint of the immediate consequence operator, where the computation of the fixpoint defines a ranking of the derived atoms. Fuzzy stable models of programs with atomic heads can also be defined in terms of reducts and fixpoint of the immediate consequence operator (Janssen *et al.* 2012), although the notion of ranking can be extended to FASP only when recursive Łukasiewicz disjunction is disabled. Using these notions, ordered completion is defined for FASP programs in Section 4.2.

In ASP, completion and ordered completion are also applicable to disjunctive programs having at most one recursive atom in each rule head. Such programs, referred to as *head cycle free* (HCF) (Ben-Eliyahu and Dechter 1994), are usually translated into equivalent programs with atomic heads by a so-called *shift* (Eiter *et al.* 2007). The same translation also works for HCF FASP programs using Łukasiewicz disjunction in rule heads. On the other hand, Łukasiewicz conjunction and Gödel disjunction require more advanced constructions (Section 3.2) which introduce recursive Łukasiewicz disjunction in rule bodies to restrict auxiliary atoms to be Boolean. Such rules are handled by integrality constraints in the theory produced by the completion, while they inhibit the application of the ordered completion. As in ASP, the shift is unsound in general for FASP programs with head cycles, and complexity arguments given in Section 3.1 prove that it is unlikely that head connectives other than Gödel conjunction can be eliminated in general.

The general translation into SMT, completion, and ordered completion are implemented in a new FASP solver called FASP2SMT (`http://alviano.net/software/fasp2smt/`; see Section 5). FASP2SMT uses GRINGO (Gebser *et al.* 2011) to obtain a ground representation of the input program, and z3 (de Moura and Bjørner 2008) to solve SMT instances encoding ground programs. Efficiency of FASP2SMT is compared with the previously implemented solver FFASP (Mushthofa *et al.* 2014), showing strengths and weaknesses of the proposed approach.

2 Background

We briefly recall the syntax and semantics of FASP (Nieuwenborgh *et al.* 2007; Lee and Wang 2014) and SMT (Barrett *et al.* 2009). Only the notions needed for the paper are introduced; for example, we only consider real arithmetic for SMT.

2.1 Fuzzy answer set programming

Let $\mathscr{B}$ be a fixed set of propositional atoms. A *fuzzy atom* (*atom* for short) is either a propositional atom from $\mathscr{B}$, or a numeric constant in $[0,1]$. *Fuzzy expressions* are defined inductively as follows: every atom is a fuzzy expression; if α is a fuzzy expression then $\sim\alpha$ is a fuzzy expression, where $\sim$ denotes *negation as failure*; if α and β are fuzzy expressions, and $\odot \in \{\otimes, \oplus, \underline{\vee}, \overline{\wedge}\}$ is a connective, $\alpha \odot \beta$ is a fuzzy expression. Connectives $\otimes, \oplus$ are known as the Łukasiewicz connectives, and $\underline{\vee}, \overline{\wedge}$ are the Gödel connectives. A *head expression* is a fuzzy expression of the form $p_1 \odot \cdots \odot p_n$, where $n \geqslant 1$, $p_1, \ldots, p_n$ are atoms, and $\odot \in \{\otimes, \oplus, \underline{\vee}, \overline{\wedge}\}$. A *rule* is of

https://doi.org/10.1017/S1471068415000241 Published online by Cambridge University Press

the form $\alpha \leftarrow \beta$, where α is a head expression, and β is a fuzzy expression. A *FASP program* Π is a finite set of rules. Let $At(\Pi)$ denote the set of atoms used by Π.

A *fuzzy interpretation* I for a FASP program Π is a function $I : \mathscr{B} \to [0,1]$ mapping each propositional atom of $\mathscr{B}$ into a truth degree in $[0,1]$. I is extended to fuzzy expressions as follows: $I(c) = c$ for $c \in [0,1]$; $I(\sim\alpha) = 1 - I(\alpha)$; $I(\alpha \otimes \beta) = \max\{I(\alpha) + I(\beta) - 1, 0\}$; $I(\alpha \oplus \beta) = \min\{I(\alpha) + I(\beta), 1\}$; $I(\alpha \veebar \beta) = \max\{I(\alpha), I(\beta)\}$; and $I(\alpha \barwedge \beta) = \min\{I(\alpha), I(\beta)\}$. I satisfies a rule $\alpha \leftarrow \beta$ ($I \models \alpha \leftarrow \beta$) if $I(\alpha) \geqslant I(\beta)$; I is a model of a FASP program Π, denoted $I \models \Pi$, if $I \models r$ for each $r \in \Pi$. I is a *stable model* of the FASP program Π if $I \models \Pi$ and there is no interpretation J such that $J \subset I$ and $J \models \Pi^I$, where the *reduct* Π^I is obtained from Π by replacing each occurrence of a fuzzy expression $\sim\alpha$ by the constant $1 - I(\alpha)$. Let $SM(\Pi)$ denote the set of stable models of Π. A program Π is *coherent* if $SM(\Pi) \neq \emptyset$; otherwise, Π is *incoherent*. Two programs Π, Π' are equivalent w.r.t. a crisp set $S \subseteq \mathscr{B}$, denoted $\Pi \equiv_S \Pi'$, if $|SM(\Pi)| = |SM(\Pi')|$ and $\{I \cap S \mid I \in SM(\Pi)\} = \{I \cap S \mid I \in SM(\Pi')\}$, where $I \cap S$ is the interpretation assigning $I(p)$ to all $p \in S$, and 0 to all $p \notin S$.

Example 2

Consider the scenario described in Example 1. Let U be a set of users, and $[0..T]$ the timepoints of interest, for some $T \geqslant 1$. Let $trust(x, y, t)$ be a propositional atom expressing that $x \in U$ trusts $y \in U$ at time $t \in [0..T]$. Similarly, $distrust(x, y, t)$ represents that x distrusts y at time t, and $conflict(x, y, t)$ encodes that x has a conflict with y at time t. The social network example can be encoded by the FASP program Π_1 containing the following rules, for all $x \in U$, $y \in U$, and $t \in [0..T-1]$:

$$distrust(x, y, t+1) \leftarrow distrust(x, y, t) \oplus conflict(x, y, t)$$
$$trust(x, y, t+1) \leftarrow trust(x, y, t) \otimes \sim(distrust(x, y, t+1) \otimes \sim distrust(x, y, t))$$

The second rule above states that the trust degree of x on y decreases when her distrust degree on y increases. A stable model I of $\Pi_1 \cup \{trust(Alice, Bob, 0) \leftarrow 0.8,\ conflict(Alice, Bob, 1) \leftarrow 0.2\}$ is such that $I(distrust(Alice, Bob, 2)) = 0.2$, and $I(trust(Alice, Bob, 2)) = 0.6$.

ASP programs are FASP programs such that all head connectives are $\veebar$, all body connectives are $\barwedge$, and all numeric constants are 0 or 1. Moreover, an ASP program Π implicitly contains *crispifying* rules of the form $p \leftarrow p \oplus p$, for all $p \in At(\Pi)$. In ASP programs, $\veebar$ and $\barwedge$ are usually denoted $\vee$ and $\wedge$, respectively.

2.2 Satisfiability modulo theories

Let $\Sigma = \Sigma^V \cup \Sigma^C \cup \Sigma^F \cup \Sigma^P$ be a *signature* where Σ^V is a set of *variables*, Σ^C is a set of *constant* symbols, Σ^F is the set of binary *function* symbols $\{+, -\}$, and Σ^P is the set of binary *predicate* symbols $\{<, \leqslant, \geqslant, >, =, \neq\}$. *Terms* and *formulas* over Σ are defined inductively, where we use infix notation for all binary symbols. Constants and variables are terms. If t_1, t_2 are terms and $\odot \in \Sigma^F$ then $t_1 \odot t_2$ is a term. If t_1, t_2 are terms and $\odot \in \Sigma^P$ then $t_1 \odot t_2$ is a formula. If φ is a formula and t_1, t_2 are terms then $ite(\varphi, t_1, t_2)$ is a term (*ite* stands for *if-then-else*). If φ_1, φ_2 are formulas and $\odot \in \{\vee, \wedge, \rightarrow, \leftrightarrow\}$ then $\varphi_1 \odot \varphi_2$ is a formula. If x is a variable and φ is a formula

https://doi.org/10.1017/S1471068415000241 Published online by Cambridge University Press

then $\forall x.\varphi$ is a formula. We consider only closed formulas, i.e., formulas in which all free variables are universally quantified. For a term t and integers a, b with $a < b$, we use $t \in [a..b]$ in formulas to represent the subformula $\bigvee_{i=a}^{b} t = i$. Similarly, for terms t, t_1, t_2, $t \in [t_1, t_2]$ represents $t_1 \leqslant t \wedge t \leqslant t_2$. A Σ-theory Γ is a set of Σ-formulas.

A Σ-structure $\mathscr{A}$ is a pair $(\mathbb{R}, \cdot^{\mathscr{A}})$, where $\cdot^{\mathscr{A}}$ is a mapping such that $p^{\mathscr{A}} \in \mathbb{R}$ for each constant symbol p, $(c)^{\mathscr{A}} = c$ for each number c, $\odot^{\mathscr{A}}$ is the binary function $\odot$ over reals if $\odot \in \Sigma^F$, and the binary relation $\odot$ over reals if $\odot \in \Sigma^P$. Composed terms and formulas are interpreted as follows: for $\odot \in \Sigma^F$, $(t_1 \odot t_2)^{\mathscr{A}} = t_1^{\mathscr{A}} \odot t_2^{\mathscr{A}}$; $ite(\varphi, t_1, t_2)^{\mathscr{A}}$ equals $t_1^{\mathscr{A}}$ if $\varphi^{\mathscr{A}}$ is true, and $t_2^{\mathscr{A}}$ otherwise; for $\odot \in \Sigma^P$, $(t_1 \odot t_2)^{\mathscr{A}}$ is true if and only if $t_1^{\mathscr{A}} \odot t_2^{\mathscr{A}}$; for $\odot \in \{\vee, \wedge, \rightarrow, \leftrightarrow\}$, $(\varphi_1 \odot \varphi_2)^{\mathscr{A}}$ equals $\varphi_1^{\mathscr{A}} \odot \varphi_2^{\mathscr{A}}$ (in propositional logic); $(\forall x.\varphi)^{\mathscr{A}}$ is true if and only if $\varphi[x/n]$ is true for all $n \in \mathbb{R}$, where $\varphi[x/n]$ is the formula obtained by substituting x with n in φ. $\mathscr{A}$ is a Σ-model of a theory Γ, denoted $\mathscr{A} \models \Gamma$, if $\varphi^{\mathscr{A}}$ is true for all $\varphi \in \Gamma$.

Example 3
Let Σ^C be $\{p, q, s, z\}$, x be a variable, and $\Gamma_1 = \{z \in [0, 1], \forall x.(x \geqslant z)\}$ be a Σ-theory. Any Σ-model of Γ_1 maps z to 0. If $ite(p + q \leqslant 1, p + q, 1) \geqslant s \leftrightarrow (p \geqslant ite(s - q \geqslant 0, s - q, 0) \wedge q \geqslant ite(s - p \geqslant 0, s - p, 0))$ is added to Γ_1, then any Σ-model of Γ_1 maps z to 0, and p, q, s to real numbers in the interval $[0, 1]$.

3 Structure simplification

The structure of FASP programs can be simplified through rewritings that leave at most one connective in each rule body (Mushthofa *et al.* 2014). Essentially, a rule of the form $\alpha \leftarrow \beta \odot \gamma$, with $\odot \in \{\otimes, \oplus, \veebar, \barwedge\}$, is replaced by the rules $\alpha \leftarrow p \odot q$, $p \leftarrow \beta$, and $q \leftarrow \gamma$, with p and q fresh atoms. A further simplification, implicit in the translation into crisp ASP by (Mushthofa *et al.* 2014), eliminates $\barwedge$ in rule heads and $\veebar$ in rule bodies: a rule of the form $p_1 \barwedge \cdots \barwedge p_n \leftarrow \beta$, $n \geqslant 2$, is equivalently replaced by n rules $p_i \leftarrow \beta$, for $i \in [1..n]$; and a rule of the form $\alpha \leftarrow \beta \veebar \gamma$ is replaced by $\alpha \leftarrow \beta$, $\alpha \leftarrow \gamma$. Moreover, a rule of the form $\alpha \leftarrow {\sim}\beta$ can be equivalently replaced by the rules $\alpha \leftarrow {\sim}p$ and $p \leftarrow \beta$, where p is a fresh atom. Let $simp(\Pi)$ be the program obtained from Π by applying these substitutions.

Proposition 1
For every FASP program Π, it holds that $\Pi \equiv_{At(\Pi)} simp(\Pi)$, i.e., $|SM(\Pi)| = |SM(simp(\Pi))|$ and $\{I \cap At(\Pi) \mid I \in SM(\Pi)\} = \{I \cap At(\Pi) \mid I \in SM(simp(\Pi))\}$.

Mushthofa *et al.* also simplify rule heads: $\alpha \odot \beta \leftarrow \gamma$ is replaced by $p \odot q \leftarrow \gamma$, $p \leftarrow \alpha$, $\alpha \leftarrow p$, $q \leftarrow \beta$, and $\beta \leftarrow q$, where p and q are fresh atoms. We do not apply these rewritings as they may inhibit other simplifications introduced in Section 3.2.

3.1 Hardness results

A relevant question is whether more rule connectives can be eliminated in order to further simplify the structure of FASP programs. We show that this is not possible, unless the polynomial hierarchy collapses, by adapting the usual reduction

https://doi.org/10.1017/S1471068415000241 Published online by Cambridge University Press

of 2-QBF$_\exists$ satisfiability to ASP coherence testing (Eiter and Gottlob 1995): for $n > m \geqslant 1$, $k \geqslant 1$ and formula $\phi := \exists x_1, \ldots, x_m \forall x_{m+1}, \ldots, x_n \bigvee_{i=1}^{k} L_{i,1} \wedge L_{i,2} \wedge L_{i,3}$, test the coherence of Π_ϕ below

$$x_i^T \vee x_i^F \leftarrow 1 \qquad \forall i \in [1..n] \tag{1}$$

$$x_i^T \leftarrow sat \quad x_i^F \leftarrow sat \quad 0 \leftarrow {\sim}sat \qquad \forall i \in [m+1..n] \tag{2}$$

$$sat \leftarrow \sigma(L_{i,1}) \wedge \sigma(L_{i,2}) \wedge \sigma(L_{i,3}) \qquad \forall i \in [1..k] \tag{3}$$

where $\sigma(x_i) := x_i^T$, and $\sigma(\neg x_i) := x_i^F$, for all $i \in [1..n]$. Σ_2^P-hardness for FASP programs with $\veebar$ in rule heads is proved by defining a FASP program $\Pi_\phi^{\veebar}$ comprising (1)–(3) (recall that $\vee$ is $\veebar$, and $\wedge$ is $\barwedge$). This also holds if we replace $\wedge$ with $\otimes$ in (3). Another possibility is to replace $\vee$ with $\oplus$ in (1), and add $p \leftarrow p \oplus p$ for all atoms in $At(\Pi_\phi)$, showing Σ_2^P-hardness for FASP programs with $\oplus$ in rule heads, a result already proved by Blondeel *et al.* (2014) with a different construction.

The same result also applies to $\otimes$, but we need a more involved argument. Let $\Pi_\phi^{\otimes}$ be the program obtained from Π_ϕ by replacing $\wedge$ with $\otimes$, substituting the rule (1) with the following three rules for each $i \in [1..n]$:

$$x_i^T \otimes x_i^F \leftarrow 0.5 \qquad x_i^T \otimes x_i^T \otimes x_i^T \leftarrow x_i^T \otimes x_i^T \qquad x_i^F \otimes x_i^F \otimes x_i^F \leftarrow x_i^F \otimes x_i^F$$

For all interpretations I, the first rule enforces $I(x_i^T) + I(x_i^F) \geqslant 1.5$. The second rule enforces $3 \cdot I(x_i^T) - 2 \geqslant 2 \cdot I(x_i^T) - 1$ whenever $2 \cdot I(x_i^T) - 1 > 0$, i.e., $I(x_i^T) \geqslant 1$ whenever $I(x_i^T) > 0.5$. Similarly, the third rule enforces $I(x_i^F) \geqslant 1$ whenever $I(x_i^F) > 0.5$. Hence, one of x_i^T, x_i^F is assigned 1, and the other 0.5. Since conjunctions are modeled by $\otimes$, and each conjunction contains three literals whose interpretation is either 0.5 or 1, it follows that the interpretation of the conjunction is 1 if all literals are 1, and at most 0.5 otherwise. Hence, ϕ is satisfiable if and only if $\Pi_\phi^{\otimes}$ is coherent.

Theorem 1
Checking coherence of FASP programs is Σ_2^P-hard already in the following cases: (i) all connectives are $\otimes$; (ii) head connectives are $\veebar$, and body connectives are $\barwedge$ (or $\otimes$); and (iii) head connectives are $\oplus$, and body connectives are $\barwedge$ (or $\otimes$) and $\oplus$.

3.2 Shifting heads

Theorem 1 shows that $\oplus$, $\otimes$, and $\veebar$ cannot be eliminated from rule heads in general by a polytime translation, unless the polynomial hierarchy collapses. This situation is similar to the case of disjunctions in ASP programs, which cannot be eliminated either. However, *head cycle free* (HCF) programs admit a translation known as *shift* that eliminates $\vee$ preserving stable models (Eiter *et al.* 2007). We extend this idea to FASP connectives. The definition of HCF programs relies on the notion of *dependency graph*. Let $pos(\alpha)$ denote the set of propositional atoms occurring in α but not under the scope of any $\sim$ symbol. The dependency graph $\mathscr{G}_\Pi$ of a FASP program Π has vertices $At(\Pi)$, and an arc (p, q) if there is a rule $\alpha \leftarrow \beta \in \Pi$ such that $p \in pos(\alpha)$, and $q \in pos(\beta)$. A *(strongly connected) component* of Π is a maximal set containing pairwise reachable vertices of $\mathscr{G}_\Pi$. A program Π is *acyclic* if $\mathscr{G}_\Pi$ is acyclic; Π is HCF if there is no rule $\alpha \leftarrow \beta$ where α contains two atoms from

https://doi.org/10.1017/S1471068415000241 Published online by Cambridge University Press

the same component of Π; Π has non-recursive $\odot \in \{\otimes, \oplus, \veebar, \barwedge\}$ in rule bodies if whenever $\odot$ occurs in the body of a rule r of $simp(\Pi)$ but not under the scope of a $\sim$ symbol then for all $p \in H(r)$ and for all $q \in pos(B(r))$ atoms p and q belong to different components of $simp(\Pi)$.

Example 4

The program $\{p \leftarrow q \oplus {\sim}{\sim}p\}$ is acyclic. Note that ${\sim}{\sim}p$ does not provide an arc to the dependency graph. Adding the rule $q \otimes s \leftarrow p$ makes the program cyclic but still HCF because q and s belong to two different components. If also $q \leftarrow s$ is added, then the program is no more HCF. Finally, note that Π_1 in Example 2 is acyclic.

It should now be clear why we decided not to reduce the number of head connectives in the translation *simp* defined at the beginning of this section. By removing a connective in the head of a rule of an HCF program, we might produce a program that is not HCF. Consider for example the HCF program $\{p \otimes q \otimes s \leftarrow 1\}$. To reduce one of the occurrences of $\otimes$, we can introduce a fresh atom *aux* that stands for $q \otimes s$. However, q and s would belong to the same component of the resulting program $\{p \otimes aux \leftarrow 1, q \otimes s \leftarrow aux, aux \leftarrow q \otimes s\}$.

We now define the *shift* of a rule for all types of head connectives. The essential idea is to move all head atoms but one to the body (hence the name shift). To preserve stable models, this has to be repeated for all head atoms, and some additional conditions might be required. For a rule of the form $p_1 \oplus \cdots \oplus p_n \leftarrow \beta$, the shift essentially mimics the original notion for ASP programs, and produces

$$p_i \leftarrow \beta \otimes {\sim}p_1 \otimes \cdots \otimes {\sim}p_{i-1} \otimes {\sim}p_{i+1} \otimes \cdots \otimes {\sim}p_n \tag{4}$$

for all $i \in [1..n]$. Intuitively, the original rule requires any model I to satisfy the condition $I(p_1) + \cdots + I(p_n) \geqslant I(\beta)$. This is the case if and only if

$$I(p_i) \geqslant I(\beta) + \sum_{j \in [1..n], j \neq i} (1 - I(p_j)) - (n-1) = I(\beta) - \sum_{j \in [1..n], j \neq i} I(p_j);$$

i.e., if and only if (4) is satisfied, for all $i \in [1..n]$. The shift of rules with other connectives in the head is more elaborate. For $p_1 \otimes \cdots \otimes p_n \leftarrow \beta$, it produces

$$p_i \leftarrow q \otimes (\beta \oplus {\sim}p_1 \oplus \cdots \oplus {\sim}p_{i-1} \oplus {\sim}p_{i+1} \oplus \cdots \oplus {\sim}p_n) \quad q \leftarrow \beta \quad q \leftarrow q \oplus q \tag{5}$$

for all $i \in [1..n]$, where q is a fresh atom. The last two rules enforce $I(q) = 1$ whenever $I(\beta) > 0$, and $I(q) = 0$ otherwise. For all $i \in [1..n]$, $I(q) = 0$ implies that the body of the first rule is interpreted as 0, and $I(q) = 1$ implies $I(q \otimes \gamma) = I(\gamma)$, where γ is $\beta \oplus {\sim}p_1 \oplus \cdots \oplus {\sim}p_{i-1} \oplus {\sim}p_{i+1} \oplus \cdots \oplus {\sim}p_n$. Since the original rule is associated with the satisfaction of $\sum_{i \in [1..n]} I(p_i) - (n-1) \geqslant I(\beta)$, which is the case if and only if $I(p_i) \geqslant I(\beta) + \sum_{j \in [1..n], j \neq i} (1 - I(p_j))$, for all $i \in [1..n]$, this translation preserves stable models for HCF programs.

The shift of $p_1 \veebar \cdots \veebar p_n \leftarrow \beta$ requires an even more advanced construction. Notice first that since the program is HCF, we can order head atoms such that for every $1 \leqslant i < j \leqslant n$, p_i does not reach p_j in $\mathscr{G}_\Pi$. Assume w.l.o.g. that one such ordering is

https://doi.org/10.1017/S1471068415000241 Published online by Cambridge University Press

given. Then, the shift of this rule is the program containing the rules

$$p_i \leftarrow \beta \mathbin{\overline{\wedge}} {\sim}q_1 \mathbin{\overline{\wedge}} \cdots \mathbin{\overline{\wedge}} {\sim}q_{i-1} \mathbin{\overline{\wedge}} q_i \tag{6}$$

$$q_i \leftarrow (p_i \veebar \cdots \veebar p_n) \otimes {\sim}(p_{i+1} \veebar \cdots \veebar p_n) \quad q_i \leftarrow q_i \oplus q_i \quad q_n \leftarrow 1 \tag{7}$$

for all $i \in [1..n]$, where each q_i is a fresh atom. Intuitively, (7) enforces $I(q_i) = 1$ whenever $I(p_i) > \max\{I(p_{i+1}), \ldots, I(p_n)\}$, and $I(q_i) = 0$ otherwise, with the exception of $I(q_n)$ which is always 1. The rule (6) enforces that $I(p_i) \geqslant I(\beta)$ whenever $I(p_i) \geqslant \max\{I(p_1), \ldots, I(p_{i-1})\}$, and either $I(p_i) > \max\{I(p_{i+1}), \ldots, I(p_n)\}$ or $i = n$. In the following, let $shift(\Pi)$ denote the program obtained by shifting all rules of Π.

Theorem 2
Let Π be FASP program. If Π is HCF then $\Pi \equiv_{At(\Pi)} shift(\Pi)$.

4 Translation into SMT

We now define a translation *smt* mapping Π into a Σ-theory, where $\Sigma^C = At(\Pi)$, and $\Sigma^V = \{x_p \mid p \in At(\Pi)\}$. The theory has two parts, *out* and *inn*, for producing a model and checking its minimality, respectively. In more detail, $f \in \{out, inn\}$ is the following: for $c \in [0,1]$, $f(c) = c$; for $p \in At(\Pi)$, $f(p)$ is p if $f = out$, and x_p otherwise; $f({\sim}\alpha) = 1 - out(\alpha)$; $f(\alpha \oplus \beta) = ite(t \leqslant 1, t, 1)$, where t is $f(\alpha) + f(\beta)$; $f(\alpha \otimes \beta) = ite(t \geqslant 0, t, 0)$, where t stands for $f(\alpha) + f(\beta) - 1$; $f(\alpha \veebar \beta) = ite(f(\alpha) \geqslant f(\beta), f(\alpha), f(\beta))$; $f(\alpha \mathbin{\overline{\wedge}} \beta) = ite(f(\alpha) \leqslant f(\beta), f(\alpha), f(\beta))$; $f(\alpha \leftarrow \beta) = f(\alpha) \geqslant f(\beta)$. Note that propositional atoms are mapped to constants by *out*, and to variables by *inn*. Moreover, negated expressions are always mapped by *out*. Define $smt(\Pi) := \{p \in [0,1] \mid p \in At(\Pi)\} \cup \{out(r) \mid r \in \Pi\} \cup \{\varphi_{inn}\}$, where

$$\varphi_{inn} := \forall\{x_p \mid p \in At(\Pi)\}. \bigwedge_{p \in At(\Pi)} x_p \in [0,p] \wedge \bigwedge_{r \in \Pi} inn(r) \rightarrow \bigwedge_{p \in At(\Pi)} x_p = p. \tag{8}$$

Example 5
Consider the program $\Pi_2 = \{p \leftarrow q \veebar {\sim}s,\ q \oplus s \leftarrow {\sim}{\sim}p\}$. The theory $smt(\Pi_2)$ is $\{p \in [0,1], q \in [0,1], s \in [0,1]\} \cup \{p \geqslant ite(q \geqslant 1-s, q, 1-s),\ ite(q+s \leqslant 1, q+s, 1) \geqslant 1-(1-p)\} \cup \{\forall x_p.\forall x_q.\forall x_s.x_p \in [0,p] \wedge x_q \in [0,q] \wedge x_s \in [0,s] \wedge x_p \geqslant ite(x_q \geqslant 1-s, x_q, 1-s) \wedge ite(x_q + x_s \leqslant 1, x_q + x_s, 1) \geqslant 1-(1-p) \rightarrow x_p = p \wedge x_q = q \wedge x_s = s\}$. Let $\mathscr{A}$ be a Σ-structure such that $p^{\mathscr{A}} = q^{\mathscr{A}} = 1$ and $s^{\mathscr{A}} = 0$. It can be checked that $\mathscr{A} \models smt(\Pi_2)$. Also note that $I(p) = I(q) = 1$ and $I(s) = 0$ implies $I \in SM(\Pi_2)$.

For an interpretation I of Π, let $\mathscr{A}_I$ be the one-to-one Σ-structure for $smt(\Pi)$ such that $p^{\mathscr{A}_I} = I(p)$, for all $p \in At(\Pi)$.

Theorem 3
Let Π be a FASP program. $I \in SM(\Pi)$ if and only if $\mathscr{A}_I \models smt(\Pi)$.

4.1 Completion

A drawback of *smt* is that it produces quantified theories, which are usually handled by incomplete heuristics in SMT solvers (Ge and de Moura 2009). Structural

https://doi.org/10.1017/S1471068415000241 Published online by Cambridge University Press

properties of FASP programs may be exploited to obtain a more tailored translation that extends *completion* (Clark 1977) to the fuzzy case. Completion is a translation into propositional theories used to compute stable models of acyclic ASP programs with atomic heads. Intuitively, the models of the completion of a program Π coincide with the *supported models* of Π, i.e., those models I with $I(p) = \max\{I(\beta) \mid p \leftarrow \beta \in \Pi\}$, for each $p \in At(\Pi)$. This notion was extended to FASP programs by Janssen *et al.* (2012), with fuzzy propositional theories as target framework. We adapt it to produce Σ-theories, for the Σ defined before.

Let Π be a program with atomic heads, and $p \in At(\Pi)$. We denote by $heads(p, \Pi)$ the set of rules in Π whose head is p, and by $constraints(\Pi)$ the set of rules in Π whose head is a numeric constant. The completion of Π is the Σ-theory:

$$comp(\Pi) := \{p \in [0,1] \wedge p = supp(p, heads(p,\Pi)) \mid p \in At(\Pi)\} \cup \{out(r) \mid r \in constraints(\Pi)\}, \tag{9}$$

where $supp(p, \emptyset) := 0$, and for $n \geqslant 1$, $supp(p, \{p \leftarrow \beta_i \mid i \in [1..n]\}) := ite(out(\beta_1) \geqslant t, out(\beta_1), t)$, where t is $supp(p, \{p \leftarrow \beta_i \mid i \in [2..n]\})$. Basically, $supp(p, heads(p, \Pi))$ yields a term interpreted as $\max\{out(\beta)^{\mathscr{A}_I} \mid p \leftarrow \beta \in \Pi\}$ by all Σ-structures $\mathscr{A}$.

Example 6
Since Π_2 in Example 5 is acyclic, $\Pi_2 \equiv_{At(\Pi_2)} shift(\Pi_2)$. The theory $comp(shift(\Pi_2))$ is $\{p \in [0,1] \wedge p = ite(q \geqslant 1 - s, q, 1 - s),\ q \in [0,1] \wedge q = ite(t_1 \geqslant 0, t_1, 0),\ s \in [0,1] \wedge s = ite(p - q \geqslant 0, p - q, 0)\}$, where t_1 is $(1 - (1 - p)) + (1 - s) - 1$, and t_2 is $(1 - (1 - p)) + (1 - q) - 1$.

Since $smt(\Pi)$ and $comp(\Pi)$ have the same constant symbols, $\mathscr{A}_I$ defines a one-to-one mapping between interpretations of Π and Σ-structures of $comp(\Pi)$. An interesting question is whether correctness can be extended to HCF programs, for example by first shifting heads. Notice that (5) and (7) introduce rules of the form $q \leftarrow q \oplus q$ through the shift of $\otimes$ or $\veebar$, breaking acyclicity. However, $q \leftarrow q \oplus q$ is a common pattern to force a Boolean interpretation of q, which can be encoded by integrality constraints in the theory. The same observation applies to rules of the form $q \otimes q \leftarrow q$. Define $bool(\Pi) := \{p \leftarrow p \oplus p \in \Pi\} \cup \{p \otimes p \leftarrow p \in \Pi\}$, and let $bool^{-}(\Pi)$ be the program obtained from $\Pi \setminus bool(\Pi)$ by performing the following operations for each $p \in At(bool(\Pi))$: first, occurrences of p in rule bodies are replaced by b_p, where b_p is a fresh atom; then, a choice rule $b_p \leftarrow {\sim\sim} b_p$ is added. The refined completion is the following:

$$rcomp(\Pi) := comp(bool^{-}(\Pi)) \cup \{b_p = ite(p > 0, 1, 0) \mid p \in At(bool(\Pi))\}, \tag{10}$$

and the associated Σ-structure $\mathscr{A}^r_I$ is such that $p^{\mathscr{A}^r_I} = I(p)$ for $p \in At(\Pi)$, and $b_p^{\mathscr{A}^r_I}$ equals 1 if $I(p) > 0$, and 0 otherwise, for $p \in At(bool(\Pi))$.

Theorem 4
Let Π be a program such that $\Pi \setminus bool(\Pi)$ is acyclic. Then, $I \in SM(\Pi)$ if and only if $\mathscr{A}^r_I \models rcomp(shift(simp(\Pi)))$.

Note that in the above theorem *simp* and *shift* are only required because *comp* and *rcomp* are defined for normal programs.

https://doi.org/10.1017/S1471068415000241 Published online by Cambridge University Press

4.2 Ordered completion

Stable models of recursive programs do not coincide with supported models, making completion unsound. To regain soundness, *ordered completion* (Ben-Eliyahu and Dechter 1994; Janhunen 2004; Niemelä 2008; Asuncion *et al.* 2012) uses a notion of *acyclic support*. Let Π be an ASP program with atomic heads. I is a stable model of Π if and only if there exists a *ranking* r such that, for each $p \in I$, $I(p) = \max\{I(\beta) \mid p \leftarrow \beta \in \Pi, r(p) = 1 + \max(\{0\} \cup \{r(q) \mid q \in pos(\beta)\})\}$ (Janhunen 2004). This holds because the reduct Π^I is also $\sim$-free, and thus its unique minimal model is the least fixpoint of the immediate consequence operator $\mathscr{T}_{\Pi^I}$, mapping interpretations J to $\mathscr{T}_{\Pi^I}(J)$ where $\mathscr{T}_{\Pi^I}(J)(p) := \max\{J(\beta) \mid p \leftarrow \beta \in \Pi^I\}$. Since $J(\alpha \wedge \beta) \leqslant J(\alpha)$ and $J(\alpha \wedge \beta) \leqslant J(\beta)$, for all interpretations J, the limit is reached in $|At(\Pi)|$ steps. For FASP programs, however, the least fixpoint of $\mathscr{T}_{\Pi^I}$ is not reached within a linear number of applications (Janssen *et al.* 2012). For example, 2^n applications are required for the program $\{p \leftarrow p \oplus c\}$, for $c = 1/2^n$ and $n \geqslant 0$ (Blondeel *et al.* 2014). On the other hand, for $\odot \in \{\overline{\wedge}, \otimes\}$ and all interpretations J, we have $J(\alpha \odot \beta) \leqslant J(\alpha)$ and $J(\alpha \odot \beta) \leqslant J(\beta)$. The claim can thus be extended to the fuzzy case if recursion over $\oplus$ and $\veebar$ is disabled.

Lemma 1
Let Π be such that Π has atomic heads and non-recursive $\oplus, \veebar$ in rule bodies. Let I be an interpretation for Π. The least fixpoint of $\mathscr{T}_{\Pi^I}$ is reached in $|At(\Pi)|$ steps.

Ordered completion can be defined for this class of FASP programs. Let J be the least fixpoint of $\mathscr{T}_{\Pi^I}$. The *rank* of $p \in At(\Pi)$ in J is the step at which $J(p)$ is derived. Let r_p be a constant symbol expressing the rank of p. Define $rank(\emptyset) := 1$, and $rank(\{q_i \mid i \in [1..n]\}) := ite(r_{q_1} \geqslant t, r_{q_1}, t)$ for $n \geqslant 1$, where $t = rank(\{q_i \mid i \in [2..n]\})$. Also define $osupp(p, \emptyset) := 0$, and for $n \geqslant 1$,

$$osupp(p, \{p \leftarrow \beta_i \mid i \in [1..n]\}) := \bigvee_{i \in [1..n]} (p = out(\beta_i) \wedge r_p = 1 + rank(pos(\beta_i))).$$

The ordered completion of Π, denoted $ocomp(\Pi)$, is the following theory:

$$comp(\Pi) \cup \{r_p \in [1..|At(\Pi)|] \wedge p > 0 \rightarrow osupp(p, heads(p, \Pi)) \mid p \in At(\Pi)\}. \quad (11)$$

Example 7
The Σ-theory $ocomp(\{p \leftarrow 0.1,\ p \leftarrow q,\ q \leftarrow p\})$ is the following:

$$\begin{aligned} &\{p \in [0,1] \wedge p = ite(0.1 \geqslant q, 0.1, q)\} \cup \{q \in [0,1] \wedge q = p\} \\ &\cup \{r_p \in [1..2] \wedge p > 0 \rightarrow (p = 0.1 \wedge r_p = 1 + 0) \vee (p = q \wedge r_p = 1 + r_q)\} \\ &\cup \{r_q \in [1..2] \wedge q > 0 \rightarrow q = p \wedge r_q = 1 + r_p)\}. \end{aligned}$$

The theory is satisfied by $\mathscr{A}$ if $p^{\mathscr{A}} = q^{\mathscr{A}} = 0.1$, $r_p^{\mathscr{A}} = 1$, and $r_q^{\mathscr{A}} = 2$.

The correctness of *ocomp*, provided that Π satisfies the conditions of Lemma 1, is proved by the following mappings: for $I \in SM(\Pi)$, let $\mathscr{A}_I^o$ be the Σ-model for $ocomp(\Pi)$ such that $p^{\mathscr{A}_I^o} = I(p)$ and $r_p^{\mathscr{A}_I^o}$ is the rank of p in I, for all $p \in At(\Pi)$; for $\mathscr{A}$ such that $\mathscr{A} \models ocomp(\Pi)$, let $I_{\mathscr{A}}$ be the interpretation for Π such that $I_{\mathscr{A}}(p) = p^{\mathscr{A}}$, for all $p \in At(\Pi)$.

https://doi.org/10.1017/S1471068415000241 Published online by Cambridge University Press

Theorem 5
Let Π be an HCF program with non-recursive $\oplus$ in rule bodies, and whose head connectives are $\overline{\wedge}, \oplus$. If $I \in SM(\Pi)$ then $\mathscr{A}_I^o \models ocomp(shift(simp(\Pi)))$. Dually, if $\mathscr{A} \models ocomp(shift(simp(\Pi)))$ then $I_{\mathscr{A}} \in SM(\Pi)$.

The above theorem does not apply in case of recursive $\oplus$ in rule bodies. For example, $\{p \leftarrow p \oplus 0.1\}$ has a unique stable model assigning 1 to p, while its ordered completion is the following Σ-theory with no Σ-model: $\{p \in [0,1] \wedge p = ite(p + 0.1 \leqslant 1, p + 0.1, 1)\} \cup \{r_p \in [1..1] \wedge p > 0 \rightarrow p = ite(p + 0.1 \leqslant 1, p + 0.1, 1) \wedge r_p = 1 + r_p\}$.

5 Implementation and experiment

We implemented the translations from Section 3 in the new FASP solver FASP2SMT. FASP2SMT is written in PYTHON, and uses GRINGO (Gebser *et al.* 2011) to obtain a ground representation of the input program, and Z3 (de Moura and Bjørner 2008) to solve SMT instances encoding ground programs. The output of GRINGO encodes a propositional program, say Π, that is conformant with the syntax in Section 2. The components of Π are computed, and the structure of the program is analyzed. If $\Pi \setminus bool(\Pi)$ is acyclic, $rcomp(shift(simp(\Pi)))$ is built. If Π is HCF with non-recursive $\oplus$ in rule bodies, and only $\overline{\wedge}$ and $\oplus$ in rule heads, then $ocomp(shift(simp(\Pi)))$ is built. In all other cases, $smt(simp(\Pi))$ is built. The built theory is fed into Z3, and either a stable model or the string INCOHERENT is reported.

The performance of FASP2SMT was assessed on instances of a benchmark used to evaluate the FASP solver FFASP (Mushthofa *et al.* 2014). The benchmark comprises two (synthetic) problems, the fuzzy versions of *Graph Coloring* and *Hamiltonian Path*, originally considered by Alviano and Peñaloza (2013). In Graph Coloring edges of an input graph are associated with truth degrees, and each vertex x is non-deterministically colored with a shadow of gray, i.e., truth degree 1 is distributed among the atoms $black_x$ and $white_x$. The truth degree of each edge xy, say d, enforces $d \otimes black_x \otimes black_y = 0$ and $d \otimes white_x \otimes white_y = 0$, i.e., adjacent vertices must be colored with sufficiently different shadows of gray. Similarly, in Hamiltonian Path vertices and edges of an input graph are associated with truth degrees, and Boolean connectives are replaced by Łukasiewicz connectives in the usual ASP encoding. The truth degree of each edge xy, say d, is non-deterministically distributed among the atoms in_{xy} and out_{xy}. Reaching a vertex y from the initial vertex x via an edge xy guarantees that y is reached with truth degree in_{xy}. Reaching a third vertex z via an edge yz, instead, guarantees that z is reached with truth degree $in_{xy} \otimes in_{yz}$. In other words, the more uncertain is the selection of an edge xy, the more uncertain is the membership of y in the selected path, which in turn implies an even more uncertain membership of any z reached by an edge yz. In the original encodings, Łukasiewicz disjunction was used to guess (fuzzy) membership of elements in one of two sets. For example, Hamiltonian Path used a rule of the form $in(X,Y) \oplus out(X,Y) \leftarrow edge(X,Y)$, which was shifted and replaced by $in(X,Y) \leftarrow edge(X,Y) \otimes {\sim}out(X,Y)$ and $out(X,Y) \leftarrow edge(X,Y) \otimes {\sim}in(X,Y)$ by Alviano and Peñaloza. In fact, in 2013 the focus was on FASP programs with

https://doi.org/10.1017/S1471068415000241 Published online by Cambridge University Press

Table 1. *Average execution time (seconds) and memory consumption (MB).*

			FASP2SMT			FFASP			FFASP (shifted enc.)		
	den	**inst**	**sol**	**time**	**mem**	**sol**	**time**	**mem**	**sol**	**time**	**mem**
graph–col	20	6	6	94.0	174	6	5.3	302	6	1.5	69
	40	6	6	102.4	178	6	19.8	1112	6	5.3	181
	60	6	6	107.6	180	6	46.7	2472	6	11.8	342
	80	6	6	111.1	181	6	90.1	4420	6	21.0	550
	100	6	6	111.7	181	6	151.9	7025	6	33.6	812
ham–path	20	10	10	1.7	25	10	17.3	410	10	3.5	101
	40	10	10	1.8	25	10	20.3	462	10	2.3	105
	60	10	10	2.1	25	10	13.2	481	10	2.0	107
	80	10	10	2.4	25	10	32.9	868	10	3.9	188
	100	10	10	2.1	25	10	69.0	1385	10	6.5	323
	120	10	10	2.0	25	10	125.5	2042	10	10.5	475
	140	10	10	1.9	25	10	176.8	2821	10	14.7	669
	160	10	10	2.2	25	9	139.6	3769	10	20.8	960
	180	10	10	2.4	26	8	203.1	4914	10	28.9	1270
	200	10	10	2.2	25	7	288.4	6191	10	36.6	1628

atomic heads and only $\otimes$ in rule bodies, and the shift of $\oplus$ for these programs was implicit in the work of Blondeel *et al.* (2014). Since our focus is now on a more general setting, the original encodings were restored, even if it is clear that FASP2SMT shifts such programs by itself. In fact, Graph Coloring is recognized as acyclic, and Hamiltonian Path as HCF with no $\oplus$ in rule bodies. It turns out that FASP2SMT uses completion for Graph Coloring, and ordered completion for Hamiltonian Path. The experiment was run on an Intel Xeon CPU 2.4 GHz with 16 GB of RAM. CPU and memory usage were limited to 600 seconds and 15 GB, respectively. FASP2SMT and FFASP were tested with their default settings, and the performance was measured by PYRUNLIM (`http://alviano.net/software/pyrunlim/`), the tool used in the last ASP Competitions (Alviano *et al.* 2013; Calimeri *et al.* 2014).

The results are reported in Table 1. Instances are grouped according to the granularity of numeric constants, where instances with **den** $= d$ are characterized by numeric constants of the form n/d. There are 6 instances of Graph Coloring and 10 of Hamiltonian Path in each group. All instances of Graph Coloring are coherent, while there is an average of 4 incoherent instances in each group of Hamiltonian Path. All instances are solved by FASP2SMT (column **sol**), and the granularity of numeric constants does not really impact on execution time and memory consumption. The performance is particularly good for Hamiltonian Path, while FFASP is faster than FASP2SMT in Graph Coloring for numeric constants of limited granularity. The performance of FFASP deteriorates when the granularity of numeric constants increases, and 6 timeouts are reported for the largest instances of Hamiltonian Path. Another strength of FASP2SMT is the limited memory consumption compared to FFASP. If we decrease the memory limit to 3 GB, FFASP runs out of

https://doi.org/10.1017/S1471068415000241 Published online by Cambridge University Press

memory on 12 instances of Graph Coloring and 34 instances of Hamiltonian Path, while FASP2SMT still succeeds in all instances. For the sake of completeness, manually shifted encodings were also tested. The performance of FASP2SMT did not change, while FFASP improves considerably, especially regarding memory consumption. We also tested 180 instances (not reported in Table 1) of two simple problems called *Stratified* and *Odd Cycle* (Alviano and Peñaloza 2013; Mushthofa *et al.* 2014), which both FASP2SMT and FFASP solve in less than 1 second.

The main picture resulting from the experimental analysis is that FASP2SMT is slower than FFASP in Graph Coloring, but it is faster in Hamiltonian Path. The reason for these different behaviors can be explained by the fact that all tested instances of Graph Coloring are coherent, while incoherent instances are also present among those tested for Hamiltonian Path. To confirm such an intuition, we tested the simple program $\{p \oplus q \leftarrow 1, 0 \leftarrow p \oplus q\}$. Its incoherence is proved instantaneously by FASP2SMT, while FFASP requires 71.8 seconds and 446 MB of memory (8.3 seconds and 96 MB of memory if the program is manually shifted).

6 Conclusions

SMT proved to be a reasonable target language to compute fuzzy answer sets efficiently. In fact, when structural properties of the evaluated programs are taken into account, efficiently evaluable theories are produced by FASP2SMT. This is the case for acyclic programs, for which completion can be used, as well as for HCF programs with only $\oplus$ in rule heads and no recursive $\oplus$ in rule bodies, for which ordered completion is proposed. Moreover, common patterns to *crispify* atoms, which would introduce recursive $\oplus$ in rule bodies, are possibly replaced by integrality constraints. The performance of FASP2SMT was compared with FFASP, which performs multiple calls to an ASP solver. An advantage of FASP2SMT is that, contrary to FFASP, its performance is not affected by the approximation used to represent truth degrees in the input program. On the other hand, FFASP is currently faster than FASP2SMT for instances having a stable model with truth degrees in $\mathbb{Q}_k$, for some small k, which however cannot be determined a priori. Such a k does not exist for incoherent instances, and indeed in this case FASP2SMT significantly overcomes FFASP. It is also important to note that in general the amount of memory required by FASP2SMT is negligible compared to FFASP. Future work will evaluate the possibility to extend the approximation operators by Alviano and Peñaloza (2013) to the broader language considered in this paper, with the aim of identifing classes of programs for which the fixpoints are reached within a linear number of applications.

Acknowledgement

Mario Alviano was partially supported by MIUR within project “SI-LAB BA2-KNOW – Business Analitycs to Know”, by Regione Calabria, POR Calabria FESR 2007-2013, within projects “ITravel PLUS” and “KnowRex”, by the National

https://doi.org/10.1017/S1471068415000241 Published online by Cambridge University Press

Group for Scientific Computation (GNCS-INDAM), and by Finanziamento Giovani Ricercatori UNICAL. Rafael Peñaloza was partially supported by the DFG within the Cluster of Excellence 'cfAED;' this work was developed while still being affiliated with TU Dresden and the Center for Advancing Electronics Dresden, Germany.

References

Akbarpour, B. and Paulson, L. C. 2010. Metitarski: An automatic theorem prover for real-valued special functions. *J. Autom. Reasoning 44,* 3, 175–205.

Alviano, M., Calimeri, F., Charwat, G., Dao-Tran, M., Dodaro, C., Ianni, G., Krennwallner, T., Kronegger, M., Oetsch, J., Pfandler, A., Pührer, J., Redl, C., Ricca, F., Schneider, P., Schwengerer, M., Spendier, L. K., Wallner, J. P. and Xiao, G. 2013. The fourth answer set programming competition: Preliminary report. In *LPNMR*, P. Cabalar and T. C. Son, Eds. LNCS. 42–53.

Alviano, M., Dodaro, C., Faber, W., Leone, N. and Ricca, F. 2013. WASP: A native ASP solver based on constraint learning. In *Logic Programming and Nonmonotonic Reasoning, 12th International Conference, LPNMR 2013, Corunna, Spain, September 15-19, 2013. Proceedings*, P. Cabalar and T. C. Son, Eds. Lecture Notes in Computer Science, vol. 8148. Springer, 54–66.

Alviano, M., Faber, W., Leone, N., Perri, S., Pfeifer, G. and Terracina, G. 2010. The disjunctive datalog system DLV. In *Datalog Reloaded - First International Workshop, Datalog 2010, Oxford, UK, March 16-19, 2010. Revised Selected Papers*, O. de Moor, G. Gottlob, T. Furche, and A. J. Sellers, Eds. Lecture Notes in Computer Science, vol. 6702. Springer, 282–301.

Alviano, M. and Peñaloza, R. 2013. Fuzzy answer sets approximations. *TPLP 13,* 4-5, 753–767.

Asuncion, V., Lin, F., Zhang, Y. and Zhou, Y. 2012. Ordered completion for first-order logic programs on finite structures. *Artif. Intell. 177-179*, 1–24.

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

Barrett, C. W., Sebastiani, R., Seshia, S. A. and Tinelli, C. 2009. Satisfiability modulo theories. In *Handbook of Satisfiability*, A. Biere, M. Heule, H. van Maaren, and T. Walsh, Eds. Frontiers in Artificial Intelligence and Applications, vol. 185. IOS Press, 825–885.

Ben-Eliyahu, R. and Dechter, R. 1994. Propositional semantics for disjunctive logic programs. *Ann. Math. Artif. Intell. 12,* 1-2, 53–87.

Blondeel, M., Schockaert, S., Vermeir, D. and Cock, M. D. 2014. Complexity of fuzzy answer set programming under łukasiewicz semantics. *Int. J. Approx. Reasoning 55,* 9, 1971–2003.

Calimeri, F., Gebser, M., Maratea, M. and Ricca, F. 2014. The design of the fifth answer set programming competition. *CoRR abs/1405.3710.*

Clark, K. L. 1977. Negation as failure. In *Logic and Data Bases*. 293–322.

de Moura, L. M. and Bjørner, N. 2008. Z3: an efficient SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, C. R. Ramakrishnan and J. Rehof, Eds. Lecture Notes in Computer Science, vol. 4963. Springer, 337–340.

Delgrande, J. P., Schaub, T., Tompits, H. and Woltran, S. 2008. Belief revision of logic programs under answer set semantics. In *Principles of Knowledge Representation*

https://doi.org/10.1017/S1471068415000241 Published online by Cambridge University Press

and Reasoning: Proceedings of the Eleventh International Conference, KR 2008, Sydney, Australia, September 16-19, 2008, G. Brewka and J. Lang, Eds. 411–421.

Eiter, T., Fink, M. and Woltran, S. 2007. Semantical characterizations and complexity of equivalences in answer set programming. *ACM Trans. Comput. Log. 8,* 3.

Eiter, T. and Gottlob, G. 1995. On the computational cost of disjunctive logic programming: Propositional case. *Ann. Math. Artif. Intell. 15,* 3-4, 289–323.

Ge, Y. and de Moura, L. M. 2009. Complete instantiation for quantified formulas in satisfiabiliby modulo theories. In *Computer Aided Verification, 21st International Conference, CAV 2009, Grenoble, France, June 26 - July 2, 2009. Proceedings*, A. Bouajjani and O. Maler, Eds. Lecture Notes in Computer Science, vol. 5643. Springer, 306–320.

Gebser, M., Kaminski, R., König, A. and Schaub, T. 2011. Advances in *gringo* series 3. In *Logic Programming and Nonmonotonic Reasoning - 11th International Conference, LPNMR 2011, Vancouver, Canada, May 16-19, 2011. Proceedings*, J. P. Delgrande and W. Faber, Eds. Lecture Notes in Computer Science, vol. 6645. Springer, 345–351.

Gebser, M., Kaufmann, B. and Schaub, T. 2012. Conflict-driven answer set solving: From theory to practice. *Artif. Intell. 187*, 52–89.

Gelfond, M. and Lifschitz, V. 1991. Classical negation in logic programs and disjunctive databases. *New Generation Comput. 9,* 3/4, 365–386.

Janhunen, T. 2004. Representing normal programs with clauses. In *Proceedings of the 16th Eureopean Conference on Artificial Intelligence, ECAI'2004, Valencia, Spain, August 22-27, 2004*, R. L. de Mántaras and L. Saitta, Eds. IOS Press, 358–362.

Janssen, J., Schockaert, S., Vermeir, D. and Cock, M. D. 2012. *Answer Set Programming for Continuous Domains - A Fuzzy Logic Approach.* Atlantis Computational Intelligence Systems, vol. 5. Atlantis Press.

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

Lee, J. and Wang, Y. 2014. Stable models of fuzzy propositional formulas. In *Logics in Artificial Intelligence - 14th European Conference, JELIA 2014, Funchal, Madeira, Portugal, September 24-26, 2014. Proceedings*, E. Fermé and J. Leite, Eds. Lecture Notes in Computer Science, vol. 8761. Springer, 326–339.

Lierler, Y. and Maratea, M. 2004. Cmodels-2: Sat-based answer set solver enhanced to non-tight programs. In *Logic Programming and Nonmonotonic Reasoning, 7th International Conference, LPNMR 2004, Fort Lauderdale, FL, USA, January 6-8, 2004, Proceedings*, V. Lifschitz and I. Niemelä, Eds. Lecture Notes in Computer Science, vol. 2923. Springer, 346–350.

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

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

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

Mushthofa, M., Schockaert, S. and Cock, M. D. 2014. A finite-valued solver for disjunctive fuzzy answer set programs. In *ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic*, T. Schaub, G. Friedrich, and B. O'Sullivan, Eds. Frontiers in Artificial Intelligence and Applications, vol. 263. IOS Press, 645–650.

https://doi.org/10.1017/S1471068415000241 Published online by Cambridge University Press

Niemelä, I. 1999. Logic programs with stable model semantics as a constraint programming paradigm. *Ann. Math. Artif. Intell. 25,* 3-4, 241–273.

Niemelä, I. 2008. Stable models and difference logic. *Ann. Math. Artif. Intell. 53,* 1-4, 313–329.

Nieuwenborgh, D. V., Cock, M. D. and Vermeir, D. 2007. An introduction to fuzzy answer set programming. *Ann. Math. Artif. Intell. 50,* 3-4, 363–388.

Ratschan, S. 2006. Efficient solving of quantified inequality constraints over the real numbers. *ACM Trans. Comput. Log. 7,* 4, 723–748.

https://doi.org/10.1017/S1471068415000241 Published online by Cambridge University Press