Fuzzy Answer Set Computation via Satisfiability Modulo Theories*

Mario Alviano\textsuperscript{1} and Rafael Peñaloza\textsuperscript{2}

\textsuperscript{1} University of Calabria, Italy
\textsuperscript{2} Free University of Bozen-Bolzano, Italy

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) \cite{dantsin01,Dung95,Kakas95} 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 \cite{Caminada10,Maratea13,Marek13,Marek15}. 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

\* This work will be also presented at the 31st International Conference on Logic Programming (ICLP 2015). Mario Alviano was partially supported by MIUR within project “SI-LAB BA2-KNOW – Business Analytics to Know”, by Regione Calabria, POR Calabria FESR 2007-2013, within project “ITravel PLUS” and project “KnowRex”, by the National 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.
may arise from the intrinsic limits of sensors, or the vagueness of natural language. Fuzzy answer set programming (FASP) [32] overcomes this limitation by 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 [23]. In [23] there is also 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 [4], CMODELS [24], CLASP [18], and WASP [3], which is not the case for FASP. A preliminary FASP solver for programs with atomic heads and Łukasiewicz conjunction, called FASP, was presented in [5]. It implements approximation operators and a translation into bilvel programming [10]. A more general solver, called FFASP [29], is based on a translation into ASP for computing stable models whose truth degrees are in the set $\mathbb{Q}_k := \{i/k | i \in [0..k]\}$, for a fixed $k$. In general, exponentially many $k$ must be tested for checking the existence of a stable model, which is infeasible in practice. Hence, 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 [23]. 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) [8] extends propositional logic with external background theories—e.g. real arithmetic [33, 1]—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 [22], 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 [9, 20, 31, 6]. In a nutshell, 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 [22], 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 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) [9], are usually translated into equivalent programs with
atomic heads by a so-called shift [14]. 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 comple-
tion. 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 con-
nectives other than Gödel conjunction can be eliminated in general.

The general translation into SMT, completion, and ordered completion are imple-
mented in a new solver called FASP2SMT (http://alviano.net/software/
fasp2smt/; see Section 5). FASP2SMT uses GRINGO [17] to obtain a ground repre-
sentation of the input program, and z3 [28] to solve SMT instances encoding ground
programs. Efficiency of FASP2SMT is compared with the previously implemented solver
FFASP [29], showing strengths and weaknesses of the proposed approach.

2 Background

We briefly recall the syntax and semantics of FASP [32, 23] and SMT [8]. Only the
notions needed for the paper are introduced.

2.1 Fuzzy Answer Set Programming

Let \( B \) be a fixed set of propositional atoms. A fuzzy atom (atom for short) is either a
propositional atom from \( B \), or a numeric constant in \([0, 1]\). Fuzzy expressions are defined
inductively as follows: every atom is a fuzzy expression; if \( \alpha \) is a fuzzy expression then
\( \sim \alpha \) is a fuzzy expression, where \( \sim \) denotes negation as failure; if \( \alpha \) and \( \beta \) are
fuzzy expressions, and \( \odot \in \{\otimes, \oplus, \odot, \vee, \wedge\} \) is a connective, \( \alpha \odot \beta \) is a fuzzy expression.
Connectives \( \otimes, \oplus \) are known as the Łukasiewicz connectives, and \( \vee, \wedge \) are the Gödel
connectives. A head expression is a fuzzy expression of the form \( p_1 \odot \cdots \odot p_n \), where
\( n \geq 1 \), \( p_1, \ldots, p_n \) are atoms, and \( \odot \in \{\otimes, \oplus, \vee, \wedge\} \). A rule is of the form \( \alpha \leftarrow \beta \),
where \( \alpha \) is a head expression, and \( \beta \) is a fuzzy expression. A FASP program \( \Pi \) is a
finite set of rules. Let \( \text{At}(\Pi) \) denote the set of atoms used by \( \Pi \).

A fuzzy interpretation \( I \) for a FASP program \( \Pi \) is a function \( I : B \to [0, 1] \) mapping
each propositional atom of \( B \) into a truth degree in \([0, 1]\). \( I \) is extended to fuzzy expres-
sions as follows: \( I(c) = c \) for \( c \in [0, 1] \); \( I(\sim \alpha) = 1 - I(\alpha) \); \( I(\alpha \otimes \beta) = \max\{I(\alpha) + 
I(\beta) - 1, 0\} \); \( I(\alpha \oplus \beta) = \min\{I(\alpha) + I(\beta), 1\} \); \( I(\alpha \vee \beta) = \max\{I(\alpha), I(\beta)\} \); and
\( I(\alpha \wedge \beta) = \min\{I(\alpha), I(\beta)\} \). \( I \) satisfies a rule \( \alpha \leftarrow \beta \) if \( I(\alpha) \geq I(\beta) \);
\( I \) is a model of a FASP program \( \Pi \), denoted \( I \models \Pi \), if \( I \models r \) for each \( r \in \Pi \). \( I \) is a
stable model of the FASP program \( \Pi \) if \( I \models \Pi \) and there is no interpretation \( J \) such
that \( J \subseteq I \) and \( J \models \Pi^I \), where the reduct \( \Pi^I \) is obtained from \( \Pi \) 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 \( \Pi \). A program \( \Pi \) is coherent if \( SM(\Pi) \neq \emptyset \); otherwise, \( \Pi \) is incoherent. Two programs \( \Pi, \Pi' \) are equivalent w.r.t. a crisp set \( S \subseteq B \), 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 \( T \geq 1 \). Let \( \text{trust}(x, y, t) \) be a propositional atom expressing that \( x \in U \) trusts \( y \in U \) at time \( t \in [0..T] \). Similarly, \( \text{distrust}(x, y, t) \) represents that \( x \) distrusts \( y \) at time \( t \), and \( \text{conflict}(x, y, t) \) encodes that \( x \) has a conflict with \( y \) at time \( t \). The social network example can be encoded by the FASP program \( H_1 \) containing the following rules, for all \( x \in U, y \in U, \) and \( t \in [0..T - 1] \):

\[
\text{distrust}(x, y, t + 1) \leftarrow \text{distrust}(x, y, t) \oplus \text{conflict}(x, y, t)
\]

\[
\text{trust}(x, y, t + 1) \leftarrow \text{trust}(x, y, t) \odot \neg (\text{distrust}(x, y, t + 1) \odot \neg \text{distrust}(x, y, t))
\]

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 \( H_1 \cup \{\text{trust}(Alice, Bob, 0) \leftarrow 0.8, \text{conflict}(Alice, Bob, 1) \leftarrow 0.2\} \) is such that \( I(\text{distrust}(Alice, Bob, 2)) = 0.2 \), and \( I(\text{trust}(Alice, Bob, 2)) = 0.6 \).

ASP programs are FASP programs such that all head connectives are \( \forall \), all body connectives are \( \land \), and all numeric constants are 0 or 1. Moreover, an ASP program \( \Pi \) implicitly contains crispifying rules of the form \( p \leftarrow p \odot p \), for all \( p \in \text{At}(\Pi) \). In ASP programs, \( \forall \) and \( \land \) are usually denoted \( \lor \) and \( \land \), respectively.

### 2.2 Satisfiability Modulo Theories

Let \( \Sigma = \Sigma^V \cup \Sigma^C \cup \Sigma^F \cup \Sigma^P \) be a signature where \( \Sigma^V \) is a set of variables, \( \Sigma^C \) is a set of constant symbols, \( \Sigma^F \) is the set of binary function symbols \( \{+, -, \} \), and \( \Sigma^P \) is the set of binary predicate symbols \( \{<, \leq, \geq, =, \neq\} \). Terms and formulas over \( \Sigma \) 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 \( \circ \in \Sigma^F \) then \( t_1 \circ t_2 \) is a term. If \( t_1, t_2 \) are terms and \( \circ \in \Sigma^P \) then \( t_1 \circ t_2 \) is a term. If \( \varphi \) is a formula and \( t_1, t_2 \) are terms then \( \text{ite}(\varphi, t_1, t_2) \) is a term (\( \text{ite} \) stands for if-then-else). If \( \varphi_1, \varphi_2 \) are formulas and \( \circ \in \{\lor, \land, \rightarrow, \leftrightarrow\} \) then \( \varphi_1 \circ \varphi_2 \) is a formula. If \( x \) is a variable and \( \varphi \) is a formula 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 \( t_1 \leq t \land t \leq t_2 \). A \( \Sigma \)-theory \( \Gamma \) is a set of \( \Sigma \)-formulas.

A \( \Sigma \)-structure \( A \) is a pair \((\mathbb{R}, \cdot^A)\), where \( \cdot^A \) is a mapping such that \( p^A \in \mathbb{R} \) for each constant symbol \( p \), \( (c)^A = c \) for each number \( c \), \( \circ^A \) is the binary function \( \circ \) over reals if \( \circ \in \Sigma^F \), and the binary relation \( \circ \) over reals if \( \circ \in \Sigma^P \). Composed terms and formulas are interpreted as follows: for \( \circ \in \Sigma^F \), \( (t_1 \circ t_2)^A = t_1^A \circ t_2^A \); \( \text{ite}(\varphi, t_1, t_2)^A \) equals \( t_1^A \) if \( \varphi^A \) is true, and \( t_2^A \) otherwise; for \( \circ \in \Sigma^P \), \( (t_1 \circ t_2)^A \) is true if and only if \( t_1^A \circ t_2^A \); for \( \circ \in \{\lor, \land, \rightarrow, \leftrightarrow\} \), \( (\varphi_1 \circ \varphi_2)^A \) equals \( \varphi_1^A \circ \varphi_2^A \) (in propositional logic);
\( (\forall x. \varphi)^A \) 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 \( \varphi \). \( A \) is a \( \Sigma \)-model of a theory \( \Gamma \), denoted \( A \models \Gamma \), if \( \varphi^A \) is true for all \( \varphi \in \Gamma \).

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

\[ \blacksquare \]

### 3 Structure Simplification

The structure of FASP programs can be simplified through rewritings that leave at most one connective in each rule body [29]. Essentially, a rule of the form \( \alpha \leftarrow \beta \odot \gamma \), with \( \odot \in \{\odot, \oplus, \odot, \land, \lor\} \), is replaced by the rules \( \alpha \leftarrow p \odot q, \beta \leftarrow \gamma \), and \( q \leftarrow \gamma \), with \( p \) and \( q \) fresh atoms. A further simplification, implicit in the translation into crisp ASP by [29], eliminates \( \land \) in rule heads and \( \lor \) in rule bodies: a rule of the form \( p_1 \land \cdots \land p_n \leftarrow \beta \), \( n \geq 2 \), is equivalently replaced by \( n \) rules \( p_i \leftarrow \beta \), for \( i \in [1..n] \); and a rule of the form \( \alpha \leftarrow \beta \land \gamma \) is replaced by \( \alpha \leftarrow \beta, \alpha \leftarrow \gamma \). Moreover, a rule of the form \( \alpha \leftarrow \lnot \beta \) can be equivalently replaced by the rules \( \alpha \leftarrow \lnot p \) and \( \beta \leftarrow q \), where \( p \) is a fresh atom. Let \( \text{simp}(\Pi) \) be the program obtained from \( \Pi \) by applying these substitutions.

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

In [29], rule heads are also simplified: \( \alpha \odot \beta \leftarrow \gamma \) is replaced by \( p \odot q \leftarrow \gamma, \beta \leftarrow \alpha, \alpha \leftarrow p, q \leftarrow \beta \), 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 of 2-QBF satisfiability to ASP coherence testing [15]: for \( n > m \geq 1 \), \( k \geq 1 \) and formula \( \phi := \exists x_1, \ldots, x_m \forall \forall x_{m+1}, \ldots, x_n \bigwedge_{i=1}^{k} L_{i,1} \land L_{i,2} \land L_{i,3} \), test the coherence of \( \Pi_\phi \)

\[ x_i^T \lor x_i^F \leftarrow 1 \quad \forall i \in [1..n] \]  
\[ x_i^T \leftarrow \text{sat} \quad x_i^F \leftarrow \text{sat} \quad 0 \leftarrow \lnot \text{sat} \quad \forall i \in [m + 1..n] \]  
\[ \text{sat} \leftarrow \sigma(L_{i,1}) \land \sigma(L_{i,2}) \land \sigma(L_{i,3}) \quad \forall i \in [1..k] \]

where \( \sigma(x_i) := x_i^T \), and \( \sigma(\lnot x_i) := x_i^F \), for all \( i \in [1..n] \). \( \Sigma_2 \)-hardness for FASP programs with \( \forall \) in rule heads is proved by defining a FASP program \( \Pi_\phi^\forall \) comprising (1)–(3) (recall that \( \forall \) is \( \forall \), and \( \land \) is \( \land \)). This also holds if we replace \( \land \) with \( \odot \) in (3). Another possibility is to replace \( \lor \) with \( \oplus \) in (1), and add \( p \leftarrow p \oplus p \) for all atoms.
in $At(Π_φ)$, showing $Σ^P_2$-hardness for FASP programs with $⊕$ in rule heads, a result already proved by [10] with a different construction.

The same result also applies to $⊙$, but we need a more involved argument. Let $Π_φ^⊙$ be the program obtained from $Π_φ$ by replacing $∧$ with $⊙$, substituting the rule (1) with the following three rules for each $i ∈ [1..n]$:

$$x_i^T ⊗ x_i^F \leftarrow 0.5 \quad x_i^T ⊗ x_i^T ⊗ x_i^T \leftarrow x_i^T ⊗ x_i^T \quad x_i^F ⊗ x_i^F ⊗ x_i^F \leftarrow x_i^F ⊗ x_i^F$$

For all interpretations $I$, the first rule enforces $I(x_i^T) + I(x_i^F) ≥ 1.5$. The second rule enforces $3 \cdot I(x_i^T) - 2 ≥ 2 \cdot I(x_i^T) - 1$ whenever $2 \cdot I(x_i^T) - 1 > 0$, i.e., $I(x_i^T) ≥ 1$ whenever $I(x_i^T) > 0.5$. Similarly, the third rule enforces $I(x_i^F) ≥ 1$ 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 $⊙$, 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 $Π_φ^⊙$ is coherent.

**Theorem 1.** Checking coherence of FASP programs is $Σ^P_2$-hard already in the following cases: (i) all connectives are $⊗$; (ii) head connectives are $⊙$, and body connectives are $∧$ (or $⊙$); (iii) head connectives are $⊕$, and body connectives are $∧$ (or $⊙$) and $⊕$.

### 3.2 Shifting heads

Theorem 1 shows that $⊕$, $⊙$, and $∨$ 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 $∨$ preserving stable models [14]. We extend this idea to FASP connectives. The definition of HCF programs relies on the notion of dependency graph. Let $pos(α)$ denote the set of propositional atoms occurring in $α$ but not under the scope of any $∼$ symbol. The dependency graph $G_II$ of a FASP program $Π$ has vertices $At(Π)$, and an arc $(p, q)$ if there is a rule $α ← β ∈ Π$ such that $p ∈ pos(α)$, and $q ∈ pos(β)$. A (strongly connected) component of $Π$ is a maximal set containing pairwise reachable vertices of $G_II$. A program $Π$ is acyclic if $G_II$ is acyclic; $Π$ is HCF if there is no rule $α ← β$ where $α$ contains two atoms from the same component of $Π$; $Π$ has non-recursive $⊙ ∈ \{⊕, ⊗, ∨, ∧, ∼\}$ in rule bodies if whenever $⊙$ occurs in the body of a rule $r$ of $simp(Π)$ but not under the scope of a $∼$ symbol then for all $p ∈ H(r)$ and for all $q ∈ pos(B(r))$ atoms $p$ and $q$ belong to different components of $simp(Π)$.

**Example 4.** The program $\{p ← q ⊗ ∼p\}$ is acyclic. Note that $∼p$ does not provide an arc to the dependency graph. Adding the rule $q ⊗ s ← p$ makes the program cyclic but still HCF because $q$ and $s$ belong to two different components. If also $q ← s$ 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 $\{p ⊗ q ⊗ s ← 1\}$. 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

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

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

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

e.i., 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

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

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 $\gamma$ is $\beta \oplus \sim p_1 \oplus \cdots \oplus \sim p_{i-1} \oplus \sim p_{i+1} \oplus \cdots \oplus p_n$. Since the original rule is associated with the satisfaction of $\sum_{i \in [1..n]} I(p_i) - (n - 1) \geq I(\beta)$, which is the case if and only if $I(p_i) \geq I(\beta) + \sum_{j \in [1..n], j \neq i} (1 - I(p_j))$, for all $i \in [1..n]$, this translation preserves stable models for HCF programs.

The shift of $p_1 \not\subseteq \cdots \not\subseteq p_n \leftarrow \beta$ requires an even more advanced construction. Notice first that since the program is HCF, we can order head atoms such that for every $1 \leq i < j \leq n$, $p_j$ does not reach $p_i$ in $G_H$. Assume w.l.o.g. that one such ordering is given. Then, the shift of this rule is the program containing the rules

$$p_i \leftarrow \beta \neg \sim q_1 \neg \cdots \neg \sim q_{i-1} \neg q_i$$

$$q_i \leftarrow (p_i \not\subseteq \cdots \not\subseteq p_n) \otimes \neg (p_{i+1} \not\subseteq \cdots \not\subseteq p_n) q_i \leftarrow q_i \oplus q_i q_n \leftarrow 1$$

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 $I(p_1) \geq I(\beta)$ whenever $I(p_1) \geq \max\{ I(p_1), \ldots, I(p_{i-1}) \}$, and either $I(p_i) > \max\{ I(p_{i+1}), \ldots, I(p_n) \}$ or $i = n$. In the following, let $\text{shift}(\Pi)$ denote the program obtained by shifting all rules of $\Pi$.

**Theorem 2.** Let $\Pi$ be FASP program. If $\Pi$ is HCF then $\Pi \equiv_{\text{A}(\Pi)} \text{simp}(\Pi)$. 
4 Translation into SMT

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

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

(8)

Example 5. Consider the program \( \Pi_2 = \{ p \leftarrow q \vee \sim s, q \oplus s \leftarrow \sim \sim p \} \). The theory \( \text{smt}(\Pi_2) \) is \( \{ p \in [0,1], q \in [0,1], s \in [0,1] \} \cup \{ p \geq \text{ite}(q \geq 1 - s, q, 1 - s), \}

\text{ite}(q + s \leq 1, q + s, 1) \geq 1 - (1 - p) \} \cup \{ \forall x_p, \forall x_q, \forall x_s, x_p \in [0, p] \land x_q \in [0, q] \land x_s \in [0, s] \land x_p \geq \text{ite}(x_q \geq 1 - s, x_q, 1 - s) \land \text{ite}(x_q + x_s \leq 1, x_q + x_s, 1) \geq 1 - (1 - \text{p}) \rightarrow \} \)

\( x_p = p \land x_q = q \land x_s = s \}. \) Let \( A \) be a \( \Sigma \)-structure such that \( p^A = q^A = 1 \) and \( s^A = 0 \). It can be checked that \( A \models \text{smt}(\Pi_2) \). Also note that \( I(p) = I(q) = 1 \) and \( I(s) = 0 \) implies \( I \models SM(\Pi_2) \).

For an interpretation \( I \) of \( \Pi \), let \( A_I \) be the one-to-one \( \Sigma \)-structure for \( \text{smt}(\Pi) \) such that \( p^{A_I} = I(p) \), for all \( p \in At(\Pi) \).

Theorem 3. Let \( \Pi \) be a FASP program. \( I \in SM(\Pi) \) if and only if \( A_I \models \text{smt}(\Pi) \).

4.1 Completion

A drawback of \( \text{smt} \) is that it produces quantified theories, which are usually handled by incomplete heuristics in SMT solvers [16]. Structural properties of FASP programs may be exploited to obtain a more tailored translation that extends completion [12] 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 \( \Pi \) coincide with the supported models of \( \Pi \), 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 [22], with fuzzy propositional theories as target framework. We adapt it to produce \( \Sigma \)-theories, for the \( \Sigma \) defined before.

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

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

(9)
where $\text{supp}(p, \emptyset) := 0$, and for $n \geq 1$, $\text{supp}(p, \{p \leftarrow \beta_i \mid i \in [1..n]\}) := \text{ite}(\text{out}(\beta_1) \geq t, \text{out}(\beta_1), t)$, where $t$ is $\text{supp}(p, \{p \leftarrow \beta_i \mid i \in [2..n]\})$. Basically, $\text{supp}(p, \text{heads}(p, \Pi))$ yields a term interpreted as $\max\{\text{out}(\beta)^{A_{t_i}} \mid p \leftarrow \beta \in \Pi\}$ by all $\Sigma$-structures $A$.

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

Since $\text{smt}(\Pi)$ and $\text{comp}(\Pi)$ have the same constant symbols, $A_I$ defines a one-to-one mapping between interpretations of $\Pi$ and $\Sigma$-structures of $\text{comp}(\Pi)$. 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 $q \leftarrow q \otimes q$ through the shift of $\otimes$ or $\lor$, breaking acyclicity. However, $q \leftarrow q \lor q$ 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 $\text{bool}(\Pi) := \{p \leftarrow p \otimes p \land \Pi\} \cup \{p \otimes p \leftarrow p \land \Pi\}$, and let $\text{bool}^{-1}(\Pi)$ be the program obtained from $\Pi \setminus \text{bool}(\Pi)$ by performing the following operations for each $p \in \text{At}(\text{bool}(\Pi))$: first, occurrences of $p$ in rule bodies are replaced by $b_p$, where $b_p$ is a fresh atom; then, a choice rule $b_p \leftarrow \sim b_p$ is added. The refined completion is

\[ r\text{comp}(\Pi) := \text{comp}(\text{bool}^{-1}(\Pi)) \cup \{b_p = \text{ite}(p > 0, 1, 0) \mid p \in \text{At}(\text{bool}(\Pi))\}, \]

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

**Theorem 4.** Let $\Pi$ be a program such that $\Pi \setminus \text{bool}(\Pi)$ is acyclic. Then, $I \in \text{SM}(\Pi)$ if and only if $A_I^r \models r\text{comp}(\text{shift}(\text{simp}(\Pi)))$.

### 4.2 Ordered Completion

Stable models of recursive programs do not coincide with supported models, making completion unsound. To regain soundness, ordered completion [9, 20, 31, 6] uses a notion of acyclic support. Let $\Pi$ be an ASP program with atomic heads. $I$ is a stable model of $\Pi$ if and only if there exists a ranking $r$ such that, for each $p \in I$, $I(p) = \max\{I(\beta) \mid p \leftarrow \beta \in \Pi, r(p) = 1 + \max\{0 \cup \{r(q) \mid q \in \text{pos}(\beta)\}\}\}$. This holds because the reduct $\Pi^I$ is also $\sim$-free, and thus its unique minimal model is the least fixpoint of the immediate consequence operator $\mathcal{T}_{\Pi^I}$, mapping interpretations $J$ to $\mathcal{T}_{\Pi^I}(J)$ where $\mathcal{T}_{\Pi^I}(J)(p) := \max\{J(\beta) \mid p \leftarrow \beta \in \Pi^I\}$. Since $J(\alpha \land \beta) \leq J(\alpha)$ and $J(\alpha \land \beta) \leq J(\beta)$, for all interpretations $J$, the limit is reached in $|\text{At}(\Pi)|$ steps. For FASP programs, however, the least fixpoint of $\mathcal{T}_{\Pi^I}$ is not reached within a linear number of applications [21]. For example, $2^n$ applications are required for the program $\{p \leftarrow p \oplus c\}$, for $c = 1/2^n$ and $n \geq 0$ [10]. On the other hand, for $\oplus \in \{\land, \oplus\}$ and all interpretations $J$, we have $J(\land) \leq J(\land)$ and $J(\land) \leq J(\land)$. The claim can thus be extended to the fuzzy case if recursion over $\oplus$ and $\lor$ is disabled.

**Lemma 1.** Let $\Pi$ be such that $\Pi$ has atomic heads and non-recursive $\oplus, \lor$ in rule bodies. Let $I$ be an interpretation. The least fixpoint of $\mathcal{T}_{\Pi^I}$ is reached in $|\text{At}(\Pi)|$ steps.
Ordered completion can be defined for this class of FASP programs. Let $J$ be the least fixpoint of $T_H$. The rank of $p \in At(II)$ in $J$ is the step at which $J(p)$ is derived. Let $r_p$ be a constant symbol expressing the rank of $p$. Define $\text{rank}(\emptyset) := 1$, and $\text{rank}(\{q_i \mid i \in [1..n]\}) := \text{ite}(r_{q_i} \geq t, r_{q_i}, t)$ for $n \geq 1$, where $t = \text{rank}(\{q_i \mid i \in [2..n]\})$. Also define $\text{osupp}(p, \emptyset) := 0$, and for $n \geq 1$,

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

The ordered completion of $II$, denoted $\text{ocomp}(II)$, is the following theory:

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

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

$$\begin{align*}
\{p \in [0, 1] \land p &= \text{ite}(0.1 \geq q, 0.1, q)\} \cup \left\{ q \in [0, 1] \land q = p \right\} \\
\cup \{r_p \in [1..2] \land p > 0 \rightarrow (p = 0.1 \land r_p = 1 + 0) \lor (p = q \land r_p = 1 + r_q)\} \\
\cup \{r_q \in [1..2] \land q > 0 \rightarrow q = p \land r_q = 1 + r_p\}. 
\end{align*}$$

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

The correctness of $\text{ocomp}$, provided that $II$ satisfies the conditions of Lemma 1, is proved by the following mappings: for $I \in SM(II)$, let $A_I^f$ be the $\Sigma$-model for $\text{ocomp}(II)$ such that $p^{A_I^f} = I(p)$ and $r_p^{A_I^f}$ is the rank of $p$ in $I$, for all $p \in At(II)$; for $A$ such that $A \models \text{ocomp}(II)$, let $I_A$ be the interpretation for $II$ such that $I_A(p) = p^A$, for all $p \in At(II)$.

**Theorem 5.** Let $II$ be an HCF program with non-recursive $\oplus$ in rule bodies, and whose head connectives are $\land, \lor$. If $I \in SM(II)$ then $A_I^f \models \text{ocomp}(\text{shift}(\text{simp}(II)))$. Dually, if $A \models \text{ocomp}(\text{shift}(\text{simp}(II)))$ then $I_A \in SM(II)$.

### 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 [17] to obtain a ground representation of the input program, and Z3 [28] to solve SMT instances encoding ground programs. The output of GRINGO encodes a propositional program, say $II$, that is conformant with the syntax in Section 2. The components of $II$ are computed, and the structure of the program is analyzed. If $II\ \backslash \ \text{bool}(II)$ is acyclic, $\text{rcomp}(\text{shift}(\text{simp}(II)))$ is built. If $II$ is HCF with non-recursive $\oplus$ in rule bodies, and only $\land$ and $\lor$ in rule heads, then $\text{ocomp}(\text{shift}(\text{simp}(II)))$ is built. In all other cases, $\text{smt}(\text{simp}(II))$ 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 [29]. The benchmark comprises two (synthetic) problems, the fuzzy versions of Graph Coloring and Hamiltonian Path, originally considered by [5]. 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 $\text{black}_x$ and $\text{white}_x$. The truth degree of each edge $xy$, say $d$, enforces $d \otimes \text{black}_x \otimes \text{black}_y = 0$ and $d \otimes \text{white}_x \otimes \text{white}_y = 0$, 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 $d$, enforces $d \otimes \text{black}_x \otimes \text{black}_y = 0$ and $d \otimes \text{white}_x \otimes \text{white}_y = 0$.

In 2013 the focus was on FASP programs with atomic heads and only $\otimes$ in rule bodies, and the shift of $\oplus$ for these programs was implicit in the work of [10]. 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 [2, 11].

The results are reported in Table 1. Instances are grouped according to the granularity of numeric constants, where instances with $\text{den} = d$ 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 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* [5, 29], 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

<table>
<thead>
<tr>
<th>den</th>
<th>inst</th>
<th>FASP2SMT</th>
<th>FFASP FFASP (shifted enc.)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>den</td>
<td>sol</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>20</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>40</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>60</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>80</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>100</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>40</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>60</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>80</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>100</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>120</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>140</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>160</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>180</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>200</td>
<td>10</td>
</tr>
</tbody>
</table>
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 [5] to the broader language considered in this paper, with the aim of identifying classes of programs for which the fixpoints are reached within a linear number of applications.

References


