Solving Fuzzy Answer Set Programs in Product Logic

Ivor Uhliarik

Department of Applied Informatics, Comenius University, Mlynská dolina, 842 48 Bratislava, Slovak Republic

Keywords: Fuzzy ASP, Fuzzy Logic, Answer Set Programming, Product Logic, Stable Models, DPLL.

Abstract: In recent years, foundations have been laid for a turn in logic programming paradigms in continuous domains. Fuzzy answer set programming (FASP) has emerged as a combination of a tool for non-monotonic reasoning and solving combinatorial problems (ASP) and a knowledge representation formalism that allows for modeling partial truth (fuzzy logic). There have been various attempts at designing a solver for FASP, but they either make use of transformations into optimization programs with scaling problems, operate only on finite-valued Łukasiewicz logic, or yield only approximate answer sets. Moreover, there has been no research focused on the product logic semantics in FASP. In this work we investigate the methods used in state-of-the-art classical ASP solvers with the aim of designing a FASP solver for product propositional logic. In particular, we base our approach on the conversion into fuzzy SAT (satisfiability problem) and the fuzzy generalization of the DPLL algorithm. Since both Łukasiewicz and (extended) Gödel logic can be embedded into product logic, the resulting system should be able to operate on all three logics uniformly.

1 INTRODUCTION

Answer set programming (ASP) is a well-known and popular logic programming paradigm based on the stable model semantics (Gelfond and Lifschitz, 1988). The solid theory behind ASP (Lifschitz, 1999) has allowed a range of effective solvers to become well-established, such as the Potassco suite (Gebser et al., 2012), smodels (Simons et al., 2002), or DLV (Leone et al., 2006).

The main use of ASP is in modeling and solving combinatorial search problems. However, to formalize a problem requiring several grades of truth, or solve a continuous optimization problem, it is desirable to use a system operating with real values.

One of the ideas to extend the capabilities of ASP to continuous domains is the combination of fuzzy logic and ASP (Van Nieuwenborgh et al., 2007b). Thus, fuzzy answer set programming (FASP) was born, where atoms may be assigned graded levels of truth. The area is rather new; the most recent solver was developed by (Mushthofa et al., 2015a) and a real-world application was shown in (Mushthofa et al., 2016) which focused on modeling biological networks.

Despite numerous proposals of FASP solvers, the design and implementation of available tools are still far from reaching the maturity of classical ASP solvers. One approach of solving FASP relies on the reduction of programs into fuzzy SAT (Janssen et al., 2011) using fuzzy extensions of Clark’s completion (Clark, 1978) and loop formulas (Lin and Zhao, 2004). However, the definition of loop formulas in FASP relies on the properties of Łukasiewicz logic and, to our knowledge, no implementation of this approach is available.

Another approach (Blondeel et al., 2012) analyzes the complexity of inference in FASP and proposes a reduction of FASP programs to bilevel linear programming problems. The approach is limited to Łukasiewicz logic and has been implemented in (Alviano and Pealoza, 2013).

There also exists a group of solver proposals that focus on searching finite many-valued domains, such as (Van Nieuwenborgh et al., 2007a) or (Mushthofa et al., 2014), refined in (Mushthofa et al., 2015b), where only the latter two support disjunctive FASP programs and have available implementations.

A method has been proposed that finds approximations of fuzzy answer sets (Alviano and Pealoza, 2013), which has also been implemented, but it operates only on normal programs and yields exact results only in the case of positive and stratified programs. Note that to date, this is the only implemented approach where any of Łukasiewicz, Gödel, and product t-norms may be used.
It is clear that the furthest developed proposals are those based on Łukasiewicz logic. Overall, to the best of our knowledge, there is no ad-hoc design or implementation of an exact FASP solver for programs with Gödel or product t-norms, i.e. the only existing solver is limited to stratified negation and relies on black-box optimization techniques.

A promising way to fill this gap is to adopt the mentioned approach of reducing the FASP program into a fuzzy SAT problem (Janssen et al., 2011). However, we focus on solving programs with product t-norms, as both Łukasiewicz and (extended) Gödel logics have a faithful interpretation in (extended) product logic (Baaz et al., 1998). Hence, the solver would be able to uniformly process all three semantics. The steps in the reduction of the FASP program need to be generalized to comply with the properties of product logic. Having available the result of the reduction, we do not rely on the potential existence of a fuzzy SAT solver—instead, the fuzzy theory is then translated into clausal form according to (Guller, 2013) and we make use of the Davis-Putnam-Logemann-Loveland (DPLL) procedure for propositional product logic (also suggested by Guller) with modifications.

Note that this is a work in progress: we describe our aim and the key concepts and identify the problems that need to be solved, but we omit the proofs and implementation details.

2 Fuzzy Answer Set Programming

In this section we describe the syntax and semantics of fuzzy answer set programs. We focus on product logic semantics, but otherwise we use the definitions from (Janssen et al., 2012) and the notation from (Guller, 2013).

2.1 Syntax

Let \( L \) be a lattice. In this work we will focus on the lattice \( L = ([0,1], \leq) \). Let \( PropAtom \) be the set of propositional atoms of product logic over \( L \).

Next, let \( \mathbb{T} = \{ \top \} \) be a set of truth constants where \( \{ 0, 1 \} \subseteq \mathbb{C} \subseteq [0, 1] \) and \( \mathbb{C} \) is a countable set; \( \top, \bot \in PropAtom \) are true and false in product logic, respectively. A (classical) literal is either a constant symbol \( \sigma \in \mathbb{C} \), an atom \( a \) or a classical negation literal \( \neg a \). An extended literal is either a classical literal \( a \) or a default negation literal not \( a \).

FASP rules are expressions of the form

\[
 r \equiv a \iff f(b_1, \ldots, b_n; c_1, \ldots, c_m)
\]

where \( a, b_i, c_j \in PropAtom \) for all \( 1 \leq i \leq n, 1 \leq j \leq m \), \( f \) is a function symbol representing a total mapping \( L^{n+m} \rightarrow L \) increasing in its \( n \) first and decreasing in its \( m \) last arguments. Thus, the atom \( a \) is either a constant or a classical literal, the atoms \( b_1, \ldots, b_n \) are classical literals, and the atoms \( c_1, \ldots, c_m \) are default negation literals. The head/body of the rule \( r, t \) (also \( H(r) / B(r) \)) is the left-hand/right-hand side of the rule. The Herbrand base of a rule \( r, B_r \), is the set of atoms occurring in \( r \). A rule of the aforementioned form is called

- a constraint if \( a \in \mathbb{C} \),
- a fact if all \( b_i, c_j \in \mathbb{T} \) for \( 1 \leq i \leq n, 1 \leq j \leq m \),
- positive if \( m = 0 \) or \( c_j \in \mathbb{T} \) for \( 1 \leq j \leq m \),
- simple if it is positive and not a constraint.

A FASP program is a finite set of FASP rules. Given a FASP program \( P \), the Herbrand base \( \mathcal{B}_P \) is \( \mathcal{B}_P = \bigcup \{ B_r | r \in P \} \). A program is called

- constraint-free if it does not contain constraints,
- positive if all rules occurring in it are positive,
- simple if all rules occurring in it are simple.

2.2 Semantics

We interpret product logic by the \( \Pi \)-algebra

\[
 \Pi = ([0,1], \leq, \lor, \land, \cdot, \Rightarrow, \sim, 0, 1)
\]

where \( \lor \) is the supremum and \( \land \) the infimum operator on \( [0,1] \); \( \cdot \) is the standard multiplication of reals;

\[
 a \Rightarrow b = \begin{cases} 
 1 & \text{if } a \leq b, \\
 \frac{b}{a} & \text{else}; \\
 \sim a = \begin{cases} 
 1 & \text{if } a = 0, \\
 0 & \text{else}.
\end{cases}
\end{cases}
\]

Hence, the mapping \( f \) in the rule

\[
 a \leftarrow f(b_1, \ldots, b_n; c_1, \ldots, c_m)
\]

constructs a body expression recursively:

- a constant \( \sigma \in \mathbb{C} \) and an extended literal are body expressions,
- if \( \alpha \) and \( \beta \) is a body expression, then \( \alpha \odot \beta \) is also a body expression for \( \odot \in \{ \land, \lor, \cdot, \Rightarrow, \sim \} \)

where \( \land, \lor, \Rightarrow, \sim \) are standard propositional connectives and \( \land \) is strong conjunction.

An interpretation of a FASP program \( P \) is a \( \mathcal{B}_P \rightarrow L \) mapping \( I = \{ I(a_1), \ldots, I(a_n) \} \) defined as

- \( I(a) = l_i \) if \( 1 \leq i \leq n \) and \( I(a) = 0 \) otherwise. We extend \( I \) to constants and expressions as follows:

\footnote{If the rule has no negative part or consists of constants.}
There are replaced by the constants i.e. the occurrences of default negation literals not are replaced by the constants $I(\text{not } a) \in L$. The reduce of $P$ is the set of rules $P_i$ defined as $P_i = \{r' | r \in P\}$. An interpretation $A$ of a program $P$ is an answer set of $P$ iff $A$ is the answer set of $P_i$.

A simple FASP program has exactly one fuzzy answer set. A positive FASP program may have no, one, or several fuzzy answer sets.

3 SOLVING FASP

In the introduction we have briefly covered a number of approaches to solving FASP programs. We have suggested an approach that would make use of the reduction of a FASP program into a fuzzy SAT instance (Janssen et al., 2011), but (1) we aim to cover product (instead of Łukasiewicz) propositional logic semantics, as there exist embeddings of Łukasiewicz and extended Gödel logics in product logic (Baaz et al., 1998); (2) we do not rely on the existence of a potential fuzzy SAT solver—instead, we modify and integrate the proposed DPLL procedure for product logic (Guller, 2013); and (3) we study how the embeddings could be integrated to lay foundations for a uniform solver.

The pipeline of the full system shall consist of:

1. reduction of the input FASP program into product propositional fuzzy logic theory,
2. translation of the theory into clausal form,
3. performing the DPLL procedure for product logic to find a valuation of the atoms which corresponds to an answer set of the program.

In this section we describe the steps above and identify the problems that need to be solved in order for such system to be integrated.

3.1 Reducing FASP to Fuzzy SAT

A theory has been developed (Janssen et al., 2011) with the idea to translate FASP programs into propositional fuzzy logic theories whose models correspond to the answer sets of the original programs. This approach assumes the availability of a fuzzy SAT solver.

The work generalizes the popular methods used in classical ASP solvers to the fuzzy case, specifically those introduced in (Lin and Zhao, 2004), known as the ASSAT method. In particular, the contributions of the approach are the following:

1. The definition of the completion of a FASP program, where the authors also show that the answer sets of FASP programs without loops are exactly the models of its completion.
2. The generalization of loop formulas from (Lin and Zhao, 2004) allowing for the computation of answer sets of arbitrary FASP programs. The authors also generalize the ASSAT procedure to overcome the problem with an exponential number of loops.

While loop formulas in FASP are formulated using the unfounded-set semantics, the generalization of the ASSAT procedure is based on the fixpoint semantics. Hence, the paper also shows that these two coincide in FASP.

The approach in (Janssen et al., 2011) introduces a flexible framework, as in theory, the only limitations imposed on the input FASP program $P$ are that (1) $P$ has to be normal (disjunction cannot appear in the head of any rule), and (2) the only connectives occurring in $P$ are t-norms. However, in the definition of the loop formula in the fuzzy case, an equivalence is used that is preserved in Łukasiewicz logic, but not in Gödel or product logic:
Definition 1. (Janssen et al., 2012)
(Loop Formula). Let P be a FASP program and
L = {l1, ..., lm}a loop of P. Suppose that R_P(L) = {r_1, ..., r_m}. Then the loop formula induced by loop
L, denoted by \( \mathcal{L}(L, P) \), is the following fuzzy logic formula:

\[
I(\max(l_1, ..., l_m), \max((r_1)_b, ..., (r_m)_b))
\]

where I is an arbitrary residual implicator. If
R_P(L) = \emptyset, then loop formula becomes

\[
I(\max(l_1, ..., l_m), 0)
\]

Proposition 1. (Janssen et al., 2012)
The loop formula proposed for boolean answer set
programs is of the form

\[
\neg(\vee(r_1)_b \cdots \vee(r_m)_b) \Rightarrow (\neg l_1 \vee \cdots \vee \neg l_m)
\]

which is equivalent to3

\[
(l_1 \cdots \vee l_m) \Rightarrow (\neg(r_1)_b \cdots \neg(r_m)_b)
\]

This equivalence is preserved in Łukasiewicz logic,
but not in Gödel or product logic.

To extend the definition to be fully flexible, further
research into generalizing the notion of loop formulas
is required.

3.2 Solving Fuzzy SAT

The reduction approach implies the necessity to
use a solver for the fuzzy SAT problem. For
each logic, however, only few such solvers ex-
ist. According to (Janssen et al., 2012), for
Gödel logic we can use boolean SAT solvers, for
Łukasiewicz logic we can use mixed integer pro-
gramming (MIP) (Hähnle, 1994), and for product logic the
bounded mixed integer quadratically constrained pro-
gramming (bMICQP) used for fuzzy description logics
(Bobillo and Straccia, 2007). According to (Al-
viano and Pealoza, 2013), these approaches introduce
many auxiliary variables that may negatively affect
the performance. Moreover, the optimization pro-
grams run as black boxes without any ad-hoc optimi-
zations suitable for the given propositional logic.

Another numerical approach to solving fuzzy SAT
is based on the state-of-the-art black-box optimization
algorithm on a continuous domain, Covariance
Matrix Adaptation Evolution Strategy (CMA-ES) and
its extensions resulting from landscape analysis (Brys
et al., 2013). The approach focuses on Łukasiewicz
logic, but the paper also benchmarks the case for
product logic semantics. The downside of this ap-
proach is the stochastic nature of the algorithm that is
prone to converging to local optima (Hansen, 2006).

3Formula (2) in definition 1 is the fuzzy generalization
of formula (4) in proposition 1.

4 EMBEDDINGS

A key concept that is to be explored is the embedding
of Łukasiewicz and (extended) Gödel logics in (ex-
tended) product logic driven by the motivation that
once a solver for extended product FASP is imple-
mented, we would be able to uniformly process all
three popular semantics. In this section we define the
extension of product propositional logic, extend the
axioms of product logic, and cite the theorem stating
that Łukasiewicz logic is a sublogic of extended prod-
uct logic.

4.1 Extended Product Logic

The extended product logic is interpreted by the stan-
dard II-algebra augmented by the operators \( \equiv \), \( \prec \), \( \Delta \)
for the connectives \( \equiv \), \( \prec \), \( \Delta \), respectively.

\[
\Pi = ([0, 1], \leq, \lor, \land, \cdot, \Rightarrow, \equiv, \prec, \Delta, 0, 1)
\]
where $\vee$ is the supremum and $\wedge$ the infimum operator on $[0,1]$;

$$x \Rightarrow y = \begin{cases} 
1 & \text{if } x \leq y, \\
\frac{y}{x} & \text{else};
\end{cases}$$

$$x \bowtie y = \begin{cases} 
1 & \text{if } x = y, \\
0 & \text{else};
\end{cases}$$

$$\Delta x = \begin{cases} 
1 & \text{if } x = 1, \\
0 & \text{else}.
\end{cases}$$

Similarly to section 2.2, recall that $\Pi$ is a complete linearly ordered lattice algebra; $\forall, \land$ is commutative, associative, idempotent, monotone; $0, 1$ is its neutral element; $\cdot$ is commutative, associative, monotone; $1$ is its neutral element; the residuum operator $\Rightarrow$ satisfies the residuation principle. Gödel negation $\sim$ satisfies the condition:

for all $x \in \Pi_\Lambda$, $\sim x = x \Rightarrow 0$;

$\Delta$ satisfies the condition:\footnote{We assume a decreasing operator precedence: $\sim, \Delta, \cdot, =, \bowtie, \cup, \lor, \Rightarrow$.}

for all $x \in \Pi_\Lambda$, $\Delta x = x \bowtie 1$.

### 4.2 Embeddings

In this section, we shall use the notation from (Hájek, 2001). We extend the axioms of product logic by the following axioms:

1. $\Delta \varphi \lor \sim \Delta \varphi$ (A1)
2. $\Delta(\varphi \lor \psi) \rightarrow (\Delta \varphi \lor \Delta \psi)$ (A2)
3. $\Delta \varphi \rightarrow \varphi$ (A3)
4. $\Delta \varphi \rightarrow \Delta \varphi$ (A4)
5. $\Delta(\psi \rightarrow \varphi) \rightarrow (\Delta \varphi \rightarrow \Delta \psi)$ (A5)

(Baaz et al., 1998) define how Łukasiewicz logic can be embedded in this extended product logic, i.e. how the Łukasiewicz t-norm can be isomorphically transformed to restricted product on $[a, 1]$ for arbitrary fixed $0 < a < 1$. For each formula $\varphi$ with propositional variables in $\{p_1, \ldots, p_n\}$ a translation $\varphi^*$ is defined using one new propositional variable $p_0$. Let:

$$\overline{\varphi} \equiv p_0$$

$$p_i^* \equiv p_0 \lor p_i$$

$$(\varphi \land \psi)^* \equiv p_0 \lor (\varphi^* \land \psi^*)$$

$$(\varphi \land \psi)^* \equiv \varphi^* \rightarrow \psi^*$$

$$(\sim \varphi)^* \equiv \varphi^* \rightarrow p_0$$

for $i \in \{1, \ldots, n\}$.

Then, the following theorem holds:

#### Theorem 1. (Baaz et al., 1998)

Let $\varphi^*$ denote the formula $\neg \neg p_0 \rightarrow \varphi^*$. For each formula $\varphi$ not containing $p_0$, $\varphi$ is 1-tautology of Łukasiewicz logic iff $\varphi^*$ is a 1-tautology of product logic.

As such, Łukasiewicz logic has a faithful interpretation in product logic. Similarly, it is also shown how to embed Gödel (extended by $\Delta$) logic into product logic (Baaz et al., 1998).

### 5 CONCLUSIONS

Our solution is based on product propositional logic extended by the operation $\Delta$. As we have described in section 4.2, we have that Łukasiewicz and Gödel logic are both sublogics of the extended product logic $\Pi_\Lambda$. Given this, we shall define $\Pi_\Lambda$-FASP programs and corresponding answer set semantics. With the aim of developing a FASP solver based on this logic, the final system should be able to uniformly handle Łukasiewicz, Gödel, and product logics.

As shown in (Guller, 2017), another non-trivial problem is that of the incorporation of intermediate truth constants belonging to a countable subset of the open interval $(0,1)$, the possibility to use them in the bodies and heads of rules of product FASP programs, and their handling in the computation of answer sets. The occurrence of these truth constants in fact or constraint rules is a trivial matter to a FASP solver (as such, it is merely a condition in the problem of optimization). However, the properties of product logic will have to be reviewed for possible violations and required modifications and proofs.

We shall base the product fuzzy answer set programming solver on the reduction of a program to a fuzzy theory as described in (Janssen et al., 2011) and formulate the notion of loop formulas for product logic. Next, we transform the theory into clausal form as proposed in (Davis et al., 1962) and find a model satisfying the fuzzy theory using the well-known DPLL procedure extended with intermediate constants. That is, we formalize the fuzzy generalization of DPLL for product logic and enhance it to allow for ad-hoc computation of stable models in product FASP.

### ACKNOWLEDGEMENTS

The research reported in this paper was supported by the grant UK/367/2017.
REFERENCES


