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.

Uhliarik I.
Solving Fuzzy Answer Set Programs in Product Logic.
DOI: 10.5220/0006518303670372
In *Proceedings of the 9th International Joint Conference on Computational Intelligence (IJCCI 2017)*, pages 367-372
ISBN: 978-989-758-274-5

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 $\mathcal{L}$ be a lattice. In this work we will focus on the lattice $\mathcal{L} = \langle [0,1], \leq \rangle$. Let *PropAtom* be the set of propositional atoms of product logic over $\mathcal{L}$.

Next, let $\overline{\mathbb{C}} = \{\overline{c} | c \in \mathbb{C}\}$ be a set of truth constants where $\{0,1\} \subseteq \mathbb{C} \subseteq [0,1]$ and $\mathbb{C}$ is a countable set; $\overline{0}, \overline{1} \in$ *PropAtom* are *true* and *false* in product logic, respectively. A *(classical) literal* is either a constant symbol $\overline{c} \in \overline{\mathbb{C}}$, 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

$$r \equiv a \leftarrow f(b_1, ..., b_n; c_1, ..., c_m)$$

where $a, b_i, c_j \in \mathit{PropAtom}$ for all $1 \leq i \leq n, 1 \leq j \leq m$, f is a function symbol representing a total mapping $\mathcal{L}^{n+m} \rightarrow \mathcal{L}$ 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 $b_1, ..., b_n$ are classical literals, and the atoms $c_1, ..., c_m$ are default negation literals. The *head*/*body* of the rule r, r_h/r_b (also $H(r)$ / $B(r)$) is the left-hand/right-hand side of the rule. The *Herbrand base* of a rule r, $\mathcal{B}_r$, is the set of atoms occurring in r. A rule of the aforementioned form is called

- a *constraint* if $a \in \overline{\mathbb{C}}$,
- a *fact* if all $b_i, c_j \in \overline{\mathbb{C}}$ for $1 \leq i \leq n$, $1 \leq j \leq m$,
- *positive* if $m = 0$ or $c_j \in \overline{\mathbb{C}}$ for $1 \leq j \leq m$[1],
- *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 $\mathcal{B}_P = \bigcup\{\mathcal{B}_r | r \in P\}$. 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 Π-algebra

$$\Pi = ([0,1], \leq, \vee, \wedge, \cdot, \Rightarrow, \sim, 0, 1)$$

where $\vee$ is the supremum and $\wedge$ the infimum operator on $[0,1]$; $\cdot$ is the standard multiplication of reals;

$$a \Rightarrow b = \begin{cases} 1 & \textit{if } a \leq b, \\ \frac{b}{a} & \textit{else}; \end{cases} \quad \sim a = \begin{cases} 1 & \textit{if } a = 0, \\ 0 & \textit{else}. \end{cases}$$

Hence, the mapping f in the rule

$$a \leftarrow f(b_1, ..., b_n; c_1, ..., c_m)$$

constructs a body expression recursively:

- a constant $\overline{c} \in \overline{\mathbb{C}}$ and an extended literal are body expressions,
- if α and β is a body expression, then $\alpha \diamond \beta$ is also a body expression for $\diamond \in \{\wedge, \&, \vee, \rightarrow, \leftrightarrow\}$

where $\wedge, \vee, \rightarrow, \leftrightarrow$ are standard propositional connectives and & is strong conjunction.

An *interpretation* of a FASP program P is a $\mathcal{B}_P \rightarrow \mathcal{L}$ mapping $I = \{a_1^{l_1}, ..., a_n^{l_n}\}$ defined as $I(a_i) = l_i$ if $1 \leq i \leq n$ and $I(a) = 0$ otherwise. We extend I to constants and expressions as follows:

[1] If the rule has no negative part or consists of constants.

- $I(\overline{c}) = c$ if $\overline{c} \in \overline{\mathbb{C}}$
- $I(\textbf{not}\ \alpha) = \sim I(\alpha)$
- $I(\alpha \,\&\, \beta) = I(\alpha) \cdot I(\beta)$
- $I(\alpha \diamond \beta) = I(\alpha) \diamond I(\beta)$ for $\diamond \in \{\wedge, \vee, \rightarrow\}$
- $I(\alpha \leftrightarrow \beta) = I(\alpha) \Rightarrow I(\beta) \cdot I(\beta) \Rightarrow I(\alpha)$

for expressions α and β. Recall that Π is a complete linearly ordered lattice algebra; $\vee, \wedge$ is commutative, associative, idempotent, monotone; 0, 1 is its neutral element; $\cdot$ is commutative, associative, monotone; 1 is its neutral element; the residuum operator $\Rightarrow$ of $\cdot$ satisfies the condition of residuation:

$$\text{for all } a, b, c \in \Pi,\ a \cdot b \leq c \Longleftrightarrow a \leq b \Rightarrow c; \qquad (1)$$

A rule $(r : a \leftarrow \alpha) \in P$ is *satisfied* by an interpretation I of P iff $I(a) \geq I(\alpha)$[2]. An interpretation I of program P is a *model* of P iff every rule $r \in P$ is satisfied by I. For interpretations I and J of P we define $I \subseteq J$ iff $\forall a \in \mathcal{B}_P : I(a) \leq J(a)$ and $I \subset J$ iff $(\forall a \in \mathcal{B}_P : I(a) \leq J(a)) \wedge (I \neq J)$. Given this ordering on interpretations, we say a model I of P is *minimal* iff no model J exists such that $J \subset I$.

Let P be a positive FASP program. An interpretation A of P is called the *answer set* of P iff A is the minimal model of P. For non-positive programs we use the fuzzy generalization of the GL reduct. For a non-positive program P, the *reduct* of a rule $(r : a \leftarrow f(b_1, ..., b_n; c_1, \ldots, c_m)) \in P$ w.r.t. an interpretation I of P is the positive rule r^I defined as

$$r^I = r : a \leftarrow f(b_1, ..., b_n; I(c_1), \ldots, I(c_m))$$

i.e. the occurrences of default negation literals **not** a are replaced by the constants $\overline{I(\textbf{not}\ a)} \in \mathcal{L}$. The reduct of P is the set of rules P^I defined as $P^I = \{r^I | r \in P\}$. An interpretation A of a program P is an *answer set* of P iff A is the answer set of P^A.

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.

[2]This is because we can regard rules as residual implicators.

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 $L = \{l_1, ..., l_m\}$ a loop of P. Suppose that $R_P^-(L) = \{r_1, ..., r_n\}$. Then the loop formula induced by loop L, denoted by $\mathbb{LF}(L, P)$, is the following fuzzy logic formula:

$$I(\max(l_1, ..., l_m), max((r_1)_b, ..., (r_n)_b)) \tag{2}$$

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

$$\neg(\bigwedge(r_1)_b \vee ... \vee \bigwedge(r_n)_b) \Rightarrow (\neg l_1 \vee ... \vee \neg l_m) \tag{3}$$

which is equivalent to[3]

$$(l_1 \vee ... \vee l_m) \Rightarrow (\bigwedge(r_1)_b \vee ... \vee \bigwedge(r_n)_b) \tag{4}$$

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 exist. According to (Janssen et al., 2012), for Gödel logic we can use boolean SAT solvers, for Łukasiewicz logic we can use mixed integer programming (MIP) (Hähnle, 1994), and for product logic the bounded mixed integer quadratically constrained programming (bMICQP) used for fuzzy description logics (Bobillo and Straccia, 2007). According to (Alviano and Pealoza, 2013), these approaches introduce many auxiliary variables that may negatively affect the performance. Moreover, the optimization programs run as black boxes without any ad-hoc optimizations 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 extension 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 approach is the stochastic nature of the algorithm that is prone to converging to local optima (Hansen, 2006).

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

3.2.1 DPLL-based SAT Solver

A different approach to solving the fuzzy SAT problem (Guller, 2013) proposes a variant of the DPLL procedure operating over order product clausal theories. The paper introduces the transformation of a product propositional theory to order clausal form, establishes the inference (branching) rules and proposes a method to compute the valuation of atoms. The procedure is proved to be refutation sound and complete for finite order clausal theories.

The problem with this approach is that it lacks the notion of intermediate constants, i.e. the terms of the input theory are limited to propositional atoms, the constants $\overline{0}$, $\overline{1}$, and expressions built from these terms using logical connectives—there is no support for intermediate truth constants[4] which are present in most practical applications of FASP. To incorporate this notion into the fuzzy DPLL procedure, we would need to modify the theory (branching rules, valuation, and associated proofs) and study the properties under which the modifications are admissible.

Another calculus used for automated deduction in (Gödel) fuzzy logics is hyperresolution (Guller, 2017). Although the work is concerned with the Gödel t-norm, the paper extends the translation of a fuzzy propositional theory to clausal form and the hyperresolution calculus by incorporating intermediate truth constants, therefore the notions serve as motivation for the enhancement of the DPLL procedure for the product case.

4 EMBEDDINGS

A key concept that is to be explored is the embedding of Łukasiewicz and (extended) Gödel logics in (extended) product logic driven by the motivation that once a solver for extended product FASP is implemented, 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 product logic.

4.1 Extended Product Logic

The extended product logic is interpreted by the standard Π-algebra augmented by the operators $\boldsymbol{\eqcirc}$, $\boldsymbol{\prec}$, $\boldsymbol{\Delta}$ for the connectives $\eqcirc$, $\prec$, Δ, respectively.

$$\Pi_\Delta = ([0,1], \leq, \vee, \wedge, \cdot, \Rightarrow, \sim, \eqcirc, \prec, \boldsymbol{\Delta}, 0, 1)$$

[4]Constants in the open interval $(0, 1)$.

where $\vee$ is the supremum and $\wedge$ the infimum operator on $[0,1]$;

$$x \Rightarrow y = \begin{cases} 1 & \textit{if } x \leq y, \\ \frac{y}{x} & \textit{else}; \end{cases} \qquad \sim x = \begin{cases} 1 & \textit{if } x = 0, \\ 0 & \textit{else}; \end{cases}$$

$$x \eqcirc y = \begin{cases} 1 & \textit{if } x = y, \\ 0 & \textit{else}; \end{cases} \qquad x \prec y = \begin{cases} 1 & \textit{if } x < y, \\ 0 & \textit{else}; \end{cases}$$

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

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

$$\text{for all } x \in \Pi_\Delta,\ \sim x = x \Rightarrow 0;$$

$\boldsymbol{\Delta}$ satisfies the condition:[5]

$$\text{for all } x \in \Pi_\Delta,\ \boldsymbol{\Delta} x = x \eqcirc 1.$$

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:

$$\Delta\varphi \vee \neg\Delta\varphi \qquad (\Delta 1)$$

$$\Delta(\varphi \vee \psi) \rightarrow (\Delta\varphi \vee \Delta\psi) \qquad (\Delta 2)$$

$$\Delta\varphi \rightarrow \varphi \qquad (\Delta 3)$$

$$\Delta\varphi \rightarrow \Delta\Delta\varphi \qquad (\Delta 4)$$

$$\Delta(\varphi \rightarrow \psi) \rightarrow (\Delta\varphi \rightarrow \Delta\psi) \qquad (\Delta 5)$$

(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 φ with propositional variables in $\{p_1, ..., p_n\}$ a translation φ^* is defined using one new propositional variable p_0. Let:

$$\overline{0}^* \equiv p_0$$

$$p_i^* \equiv p_0 \vee p_i$$

$$(\varphi \wedge \psi)^* \equiv p_0 \vee (\varphi^* \wedge \psi^*)$$

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

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

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

Then, the following theorem holds:

Theorem 1. *(Baaz et al., 1998)*
Let $\varphi^\dagger$ denote the formula $\neg\neg p_0 \rightarrow \varphi^*$. For each formula φ not containing p_0, φ is 1-tautology of Łukasiewicz logic iff $\varphi^\dagger$ 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 Δ) logic into product logic (Baaz et al., 1998).

5 CONCLUSIONS

Our solution is based on product propositional logic extended by the operation Δ. As we have described in section 4.2, we have that Łukasiewicz and Gödel logic are both sublogics of the extended product logic Π_Δ. Given this, we shall define Π_Δ-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.

[5] We assume a decreasing operator precedence: $\sim$, $\boldsymbol{\Delta}$, $\cdot$, $\eqcirc$, $\prec$, $\wedge$, $\vee$, $\Rightarrow$.

REFERENCES

Alviano, M. and Pealoza, R. (2013). Fuzzy answer sets approximations. *Theory and Practice of Logic Programming*, 13(4-5):753767.

Baaz, M., Hájek, P., Švejda, D., and Krajíček, J. (1998). Embedding logics into product logic. *Studia Logica*, 61(1):35–47.

Blondeel, M., Schockaert, S., De Cock, M., and Vermeir, D. (2012). *NP-completeness of fuzzy answer set programming under Lukasiewicz semantics*, pages 43–50.

Bobillo, F. and Straccia, U. (2007). A fuzzy description logic with product t-norm. In *2007 IEEE International Fuzzy Systems Conference*, pages 1–6.

Brys, T., Drugan, M. M., Bosman, P. A., De Cock, M., and Nowé, A. (2013). Solving satisfiability in fuzzy logics by mixing cma-es. In *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*, GECCO '13, pages 1125–1132, New York, NY, USA. ACM.

Clark, K. L. (1978). *Negation as Failure*, pages 293–322. Springer US, Boston, MA.

Davis, M., Logemann, G., and Loveland, D. (1962). A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397.

Gebser, M., Kaminski, R., Kaufmann, B., and Schaub, T. (2012). *Answer Set Solving in Practice*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan and Claypool Publishers.

Gelfond, M. and Lifschitz, V. (1988). The stable model semantics for logic programming. pages 1070–1080. MIT Press.

Guller, D. (2013). A dpll procedure for the propositional product logic. In *Proceedings of the 5th International Joint Conference on Computational Intelligence - Volume 1: FCTA, (IJCCI 2013)*, pages 213–224. INSTICC, SciTePress.

Guller, D. (2017). *Expanding Gödel Logic with Truth Constants and the Equality, Strict Order, Delta Operators*, pages 241–269. Springer International Publishing, Cham.

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

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

Hansen, N. (2006). *The CMA Evolution Strategy: A Comparing Review*, pages 75–102. Springer Berlin Heidelberg, Berlin, Heidelberg.

Janssen, J., Schockaert, S., Vermeir, D., and Cock, M. D. (2011). Reducing fuzzy answer set programming to model finding in fuzzy logics. *CoRR*, abs/1104.5133.

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

Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., and Scarcello, F. (2006). The dlv system for knowledge representation and reasoning. *ACM Trans. Comput. Logic*, 7(3):499–562.

Lifschitz, V. (1999). Action languages, answer sets and planning. In *The Logic Programming Paradigm: a 25-Year Perspective*, pages 357–373. Springer Verlag.

Lin, F. and Zhao, Y. (2004). Assat: computing answer sets of a logic program by sat solvers. *Artificial Intelligence*, 157(1):115 – 137.

Mushthofa, M., Schockaert, S., and Cock, M. D. (2014). A finite-valued solver for disjunctive fuzzy answer set programs. In *Proceedings of the Twenty-first European Conference on Artificial Intelligence*, ECAI'14, pages 645–650, Amsterdam, The Netherlands, The Netherlands. IOS Press.

Mushthofa, M., Schockaert, S., and De Cock, M. (2015a). Solving disjunctive fuzzy answer set programs. In *International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 453–466. Springer International Publishing.

Mushthofa, M., Schockaert, S., and De Cock, M. (2015b). *Solving Disjunctive Fuzzy Answer Set Programs*, pages 453–466. Springer International Publishing, Cham.

Mushthofa, M., Schockaert, S., and De Cock, M. (2016). Computing attractors of multi-valued gene regulatory networks using fuzzy answer set programming. In *Proceedings of the 2016 IEEE International Conference on Fuzzy Systems FUZZ-IEEE'2016*, pages 1955–1962. IEEE.

Simons, P., Niemel, I., and Soininen, T. (2002). Extending and implementing the stable model semantics. *Artificial Intelligence*, 138(1):181 – 234. Knowledge Representation and Logic Programming.

Van Nieuwenborgh, D., De Cock, M., and Vermeir, D. (2007a). *Computing Fuzzy Answer Sets Using dlvhex*, pages 449–450. Springer Berlin Heidelberg, Berlin, Heidelberg.

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