A core language for fuzzy answer set programming

Jeroen Janssen[a,1,*], Steven Schockaert[b,2,**], Dirk Vermeir[a,**], Martine De Cock[b,**]

[a] *Department of Computer Science, Vrije Universiteit Brussel*
Pleinlaan 2, 1050 Brussels, Belgium
[b] *Department of Applied Mathematics and Computer Science, Universiteit Gent*
Krijgslaan 281, 9000 Ghent, Belgium

Abstract

A number of different Fuzzy Answer Set Programming (FASP) formalisms have been proposed in the last years, which all differ in the language extensions they support. In this paper we investigate the expressivity of these frameworks. Specifically we show how a variety of constructs in these languages can be implemented using a considerably simpler core language. These simulations are important as a compact and simple language is easier to implement and to reason about, while an expressive language offers more options when modeling problems.

Keywords: Fuzzy Logic, Answer Set Programming, Declarative Problem Solving

1. Introduction

Answer Set Programming (ASP) (see e.g. [2]) is a declarative modeling language that is especially suitable for describing combinatorial problems. In ASP, rules of the form $r : a \leftarrow \beta$ are used to denote that a should hold when the conjunction of literals β holds. An ASP program is a collection of such rules. Typically, programs are written in the generate-define-test style, which means that certain rules generate potential solutions, and other rules eliminate solution candidates based on tests (constraints) that depend on certain concepts from the defining part. For example, the following ASP program P_{gc} models a 2-color graph coloring problem:

$$\begin{array}{rrl} gen_1: & white(X) & \leftarrow not\ black(X) \\ gen_2: & black(X) & \leftarrow not\ white(X) \\ sim_1: & sim(X,Y) & \leftarrow white(X), white(Y) \\ sim_2: & sim(X,Y) & \leftarrow black(X), black(Y) \\ constr: & & \leftarrow edge(X,Y), sim(X,Y) \end{array}$$

In this program, rules gen_1 and gen_2 form the generate part of the program. They create a potential coloring of the graph by stating that a node is either white or black. Rules sim_1 and sim_2 form the defining part and describe when two nodes are similar in color. The test part consists of the *constr* rule. This rule eliminates solution candidates where two adjacent nodes are similarly colored. We can use this program to solve the graph coloring problem for specific graphs. First, rules of the form $fact_{a,b} : edge(a, b) \leftarrow$ are added to the program, which encode that there is an edge between node a and b. The resulting rules are grounded, meaning that a rule such as gen_2 is replaced by the set of rules $\{gen_{2\,a} : black(a) \leftarrow not\ white(a) \mid a \in$

*Principal corresponding author
**Corresponding author
Email addresses: jeroen.janssen@vub.ac.be (Jeroen Janssen), steven.schockaert@ugent.be (Steven Schockaert), dvermeir@vub.ac.be (Dirk Vermeir), martine.decock@ugent.be (Martine De Cock)
[1] Funded by a joint Research Foundation–Flanders (FWO) project
[2] Postdoctoral fellow of the Research Foundation–Flanders (FWO)

Nodes}, where *Nodes* is the set of nodes in the graph. The grounded program is then solved using an answer set solver such as Smodels [58] or DLV [34], which generates answer sets of the program that correspond to admissible graph colorings.

An important research topic in ASP is determining whether certain language extensions can be simulated using a simpler core language. Indeed, there has even been a work package by the European Working Group on Answer Set Programming (WASP) that focused on this question [51]. These simulations are important as they allow using standard available answer set solvers for solving programs with extensions. Furthermore they allow us to focus on a small core theory when showing properties, while still retaining generality.

In recent years Fuzzy Answer Set Programming (FASP) has been proposed, which aims to extend ASP with the capacity to model continuous optimization problems in a similar manner. Many different formalisms have been proposed (see e.g. [6, 7, 40, 42, 55, 64, 65, 66]), which all differ in the language constructs they support. For example, some of these approaches feature negation-as-failure, others have classical negation, and again others allow arbitrary monotonic functions in rule bodies. Unfortunately, in contrast to ASP, there has been little work focusing on the language extensions of FASP. In this paper we investigate the expressivity of different constructs. In particular, we analyze how many of the features that appear in FASP variants can be simulated in a language that is considerably simpler. This creates a bridge between the desire to have a rich and expressive FASP language on one hand and the wish to have a small core language that is easy to implement and reason about on the other hand. The advantage of having a rich and expressive language is that it removes the burden from programmers to write the simulations by hand, making the language easier to use. The advantage of having a small core language is that (i) this makes it easier to reason about the language, (ii) it makes it easier to investigate links to other theories and (iii) it facilitates the implementation of the backend of a FASP solver.

The structure of the paper is as follows. In Section 2, we recall the necessary notions of fuzzy logic used throughout the paper. After this, in Section 3, we identify a core language for FASP, called CFASP, that is sufficient to express the following extensions:

1. **Constraints**. One of the important constructs in ASP are constraint rules such as rule *constr* in program P_{gc} above. Such a rule states that its *body* (the expression on the right of $\leftarrow$) can never be true in a valid solution of the problem under consideration. The constraint in the example above helps to exclude color assignments in which nodes are similarly colored. In Section 4, we extend CFASP with constraints, resulting in the language CFASP$^{\perp}$. We reveal the capabilities of these constraints and furthermore show that a well-known procedure for eliminating constraints in ASP can be generalized to the fuzzy case.
2. **Monotonically decreasing functions**. When generalizing ASP to a many-valued setting, various types of functions may serve as generalizations of logical connectives, ranging from t-norms and t-conorms to averaging operators, as well as problem-specific hedges. Some authors (e.g. [10]) therefore allow arbitrary functions whose partial mappings are increasing or decreasing. It is easy to see that this class covers all commonly used operators from fuzzy logic. In Section 5, we extend CFASP with decreasing functions, resulting in the language CFASP^{f}. We show that allowing some of the partial mappings to be decreasing does not actually increase expressivity. In particular, we show that by using the negation-as-failure primitive, such functions can be simulated using operators that are increasing in all arguments.
3. **Rule aggregation**. Some FASP formalisms, referred to as AFASP, feature an **aggregator**, which is an expression mapping rule satisfaction values to a single value. In Section 6, we show how this extension can be simulated using only rules.
4. **S-implicators**. All the AFASP approaches introduced in the literature limit rules to correspond to residual implicators. However, there might still be some contexts in which an S-implicator is more natural than a residual implicator. In Section 7, we extend AFASP with S-implicators, resulting in the language AFASP^{s}. We motivate the use of S-implicators and show how to simulate rules based on S-implicators in AFASP.
5. **Strong negation**. In ASP, two types of negation are used intertwiningly, called negation-as-failure and strong negation (also known as classical negation). In Section 8, we show how the simulation of

strong negation in classical ASP can be generalized to the fuzzy case.

Figure 1 shows how the above extensions of FASP are related. An arrow from language L_1 to L_2 indicates that L_1 can be simulated in L_2.

In Section 9, we discuss the related work and in Section 10 we present some concluding remarks.

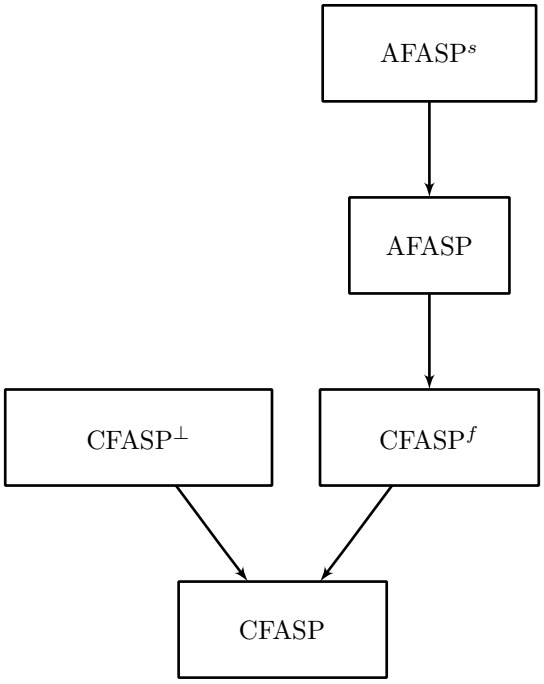

Figure 1: Diagram of the relationships between the different CFASP and AFASP languages.

2. Fuzzy Logic

Consider a complete lattice $\mathcal{L}$. We denote the ordering of $\mathcal{L}$ as $\leq_\mathcal{L}$ or as $\leq$ when there is no cause for confusion. Furthermore we denote its greatest (resp. least) element as $1_\mathcal{L}$ (resp. $0_\mathcal{L}$). When no confusion is possible, we just write 1 and 0. An $\mathcal{L}$-fuzzy set in a universe X is an $X \to \mathcal{L}$ mapping. We use $F = \{x_1^{l_1}, \ldots, x_n^{l_n}\}$ to denote an $\mathcal{L}$-fuzzy set F in a universe X, where for $i \in 1 \ldots n$ we have that $x_i \in X$, $l_i \in \mathcal{L}$ and $F(x_i) = l_i$. For $x \notin \{x_1, \ldots, x_n\}$ we have that $F(x) = 0_\mathcal{L}$. Note that any regular set A in a universe X can be considered as an $\mathcal{L}$-fuzzy set by defining for any $x \in X$ that $A(x) = 1_\mathcal{L}$ if $x \in A$ and $A(x) = 0_\mathcal{L}$ otherwise. The intersection of two $\mathcal{L}$-fuzzy sets in a universe X is the fuzzy set $F_1 \cap F_2$ defined for any $x \in X$ as $(F_1 \cap F_2)(x) = \inf(F_1(x), F_2(x))$. The inclusion of two $\mathcal{L}$-fuzzy sets in a universe X is defined as $F_1 \subseteq F_2$ iff for all x in X we have $F_1(x) \leq_\mathcal{L} F_2(x)$. The set of all $\mathcal{L}$-fuzzy sets over X is denoted as $\mathcal{F}_\mathcal{L}(X)$ or as $\mathcal{F}(X)$ when $\mathcal{L} = ([0,1], \leq)$. We call $([0,1], \leq)$-fuzzy sets fuzzy sets.

A **negator** $\sim$is a decreasing $\mathcal{L} \to \mathcal{L}$ function satisfying $\sim 0 = 1$ and $\sim 1 = 0$. This operator generalizes classic negation. If for each $x \in \mathcal{L}$ it holds that $\sim(\sim x) = x$, we call the negator **involutive**. A **t-norm** $\wedge$ is an increasing, commutative and associative $\mathcal{L}^2 \to \mathcal{L}$ function that satisfies $x \wedge 1 = x$. It generalizes classic conjunction. A **t-conorm** $\vee$ is an increasing, commutative and associative $\mathcal{L}^2 \to \mathcal{L}$ function that satisfies $x \vee 0 = x$. T-conorms generalize classic disjunction. An **implicator** $\to$ is an $\mathcal{L}^2 \to \mathcal{L}$ function that is decreasing in its first, and increasing in its second argument, satisfies $0 \to 0 = 1$ and for all $x \in \mathcal{L}$ satisfies $1 \to x = x$. T-norms whose partial mappings are sup-morphisms (i.e. $\sup_i(x_i \wedge y) = (\sup(x_i)_{i \in I}) \wedge y$) induce a **residual implicator** defined by $x \to y = \sup\{\lambda \in [0,1] \mid x \wedge \lambda \leq y\}$. Any such t-norm and its residual implicator satisfy the **residuation principle**, which states that $x \wedge z \leq y$ is equivalent to $z \leq x \to y$. Any

name	t-norm	t-conorm
minimum/maximum	$x \wedge_m y = \min(x, y)$	$x \vee_m y = \max(x, y)$
Łukasiewicz/Łukasiewicz	$x \wedge_l y = \max(0, x + y - 1)$	$x \vee_l y = \min(x + y, 1)$
product/bounded sum	$x \wedge_p y = x \cdot y$	$x \vee_p y = x + y - x \cdot y$

Table 1: Common t-norms and t-conorms over $([0, 1], \leq)$

t-norm	residual implicator	induced negator
minimum	$x \to_m y = \begin{cases} y & \text{if } x > y \\ 1 & \text{otherwise} \end{cases}$	$\sim_m x = \begin{cases} 0 & \text{if } x > 0 \\ 1 & \text{otherwise} \end{cases}$
Łukasiewicz	$x \to_l y = \min(1, 1 - x + y)$	$\sim_l x = 1 - x$
product	$x \to_p y = \begin{cases} y/x & \text{if } x > y \\ 1 & \text{otherwise} \end{cases}$	$\sim_p x = \sim_m x$

Table 2: Common residual pairs and induced negators over $([0, 1], \leq)$

implicator $\to$ furthermore induces a negator $\sim$ defined by $\sim x = x \to 0$. We summarize common t-norms, t-conorms, residual implicators and induced negators over the complete lattice $([0, 1], \leq)$ in Tables 1 and 2. The residual implicator of the minimum and product t-norm are called the **Gödel implicator**, respectively **Goguen implicator**. Next to the residual implicators, there is another common type of implicator called the **S-implicators**, which are built from a generalization of the classical logic formula $\neg x \vee y$. In this paper we will primarily use the **Kleene-Dienes** S-implicator, defined as $x \to_{kd} y = (\sim_l x \vee_m y)$.

3. The FASP Core Language

In this section we introduce a core language for FASP, called core FASP, which in the following sections will be shown to be sufficient to express many of the common extensions to FASP. First we present the language and afterwards we discuss the simulation techniques that will be used in the succeeding sections.

3.1. Language

Definition 1. *Consider a set A of atoms. A **literal** is either an atom $a \in A$, a value from a lattice $\mathcal{L}$, or a **naf-literal** of the form $\sim a$, where $\sim$ corresponds to a negator.*

Definition 2. *Given a set of atoms A, a **rule** over a complete lattice $\mathcal{L}$ is an object of the form*

$$r : a \leftarrow f(b_1, \ldots, b_n) \tag{1}$$

where a is an atom called the head, f is an increasing $\mathcal{L}^n \to \mathcal{L}$ function, b_i $(1 \leq i \leq n)$ are literals, r is the rule label and $\leftarrow$ corresponds to a residual implicator. We often refer to a rule of the form (1) only by its label r. For a rule r we denote its head as r_h, its body as r_b and the residual implicator corresponding to $\leftarrow$ as $\to_r$. The t-norm of which $\to_r$ is a residual implicator is denoted by $\wedge_r$.

The core programs are sets of rules.

Definition 3. *A **core FASP program** (short: CFASP program) over a complete lattice $\mathcal{L}$ is a set of rules over $\mathcal{L}$. We denote the set of atoms occurring in a core FASP program P as $\mathcal{B}_P$ and the lattice over which the program ranges as $\mathcal{L}_P$. Given an atom a we define $P_a = \{r \mid r \in P, r_h = a\}$. A core FASP program is called **simple** if it contains no literals of the form $\sim l$. An interpretation I of a core FASP program P is a $\mathcal{B}_P \to \mathcal{L}$ mapping. It is extended to constants from $\mathcal{L}$ by $I(l) = l$ for $l \in \mathcal{L}$, to literals of the form $\sim l$ by*

$I(\sim l) = \sim I(l)$, to bodies of rules by $I(f(b_1, \ldots, b_n)) = f(I(b_1), \ldots, I(b_n))$ and finally to a rule r of the form (1) by

$$I(r) = I(r_b) \rightarrow_r I(a) \tag{2}$$

An interpretation I is called a model of P iff for each r in P we have $I(r) = 1$.

The semantics of these programs are given by a certain subset of the models, called answer sets.

Definition 4. *Consider a simple CFASP program P. An interpretation I of P is called an **answer set** of P iff I is the minimal model of P.*

Equivalently, we can characterize answer sets as the least fixpoints of the following operator.

Definition 5. *Consider a simple CFASP program P. The **immediate consequence operator** Π associated with P is an operator mapping interpretations to interpretations defined for an interpretation I of P and $a \in \mathcal{B}_P$ as*

$$\Pi_P(I)(a) = \sup\{I(r_b) \mid r \in P_a\}$$

We denote the least fixpoint of the immediate consequence operator for a program P as Π_P^*. It is well known that this least fixpoint corresponds to the minimal model of P (see e.g. [7]) and that it can be obtained by repeatedly applying the immediate consequence operator, starting from the empty set, until a fixpoint is found. When programs are not simple, a transformation generalizing the Gelfond-Lifschitz transformation from [18] is used to define the semantics.

Definition 6. *Let P be a CFASP program and let I be an interpretation of P. The **reduct** of a literal l w.r.t. I is defined as $l^I = I(l)$ if l is of the form $\sim l$ and $l^I = l$ otherwise. The reduct of an expression $f(b_1, \ldots, b_n)$ is defined as $(f(b_1, \ldots, b_n))^I = f(b_1^I, \ldots, b_n^I)$. The reduct of a rule $r \in P$ of the form (1) is defined as $r^I = a \leftarrow (f(b_1, \ldots, b_n))^I$. The reduct of a program P is defined as $P^I = \{r^I \mid r \in P\}$.*

Definition 7. *Consider a CFASP program P. An interpretation I of P is called an **answer set** of P iff I is the answer set of P^I.*

Note that any answer set of a program P is also a minimal model of P. The converse proposition does not necessarily hold however [25].

Example 1. *Consider the following CFASP program P over $([0,1], \leq)$:*

$$\begin{array}{lllll} r_1: & a & \leftarrow & \max(b, c) \\ r_2: & b & \leftarrow & \sim_l c \\ r_3: & c & \leftarrow & 0.3 \end{array}$$

Consider the interpretation $I = \{a^{0.7}, b^{0.7}, c^{0.3}\}$ of P. To check whether it is an answer set we construct the reduct P^I:

$$\begin{array}{lllll} r_1: & a & \leftarrow & \max(b, c) \\ r_2: & b & \leftarrow & 0.7 \\ r_3: & c & \leftarrow & 0.3 \end{array}$$

Since I is the minimal model of P^I we find that I is indeed an answer set of P.

3.2. Simulation Technique: Value Fixing

In the simulations that we will discuss next it is often needed to "fix" the value of a certain atom in the body of a rule when the reduct is applied. For atoms in a naf-literal the reduct operation does this automatically, but for atoms that occur as the argument of an increasing function a special technique is needed. We call this technique *value fixing*. As an example, consider the following CFASP program P over $([0,1], \leq)$:

$$\begin{array}{lllll} r_1: & a & \leftarrow & \max(b, c) \\ r_2: & b & \leftarrow & \sim_l c \\ r_3: & c & \leftarrow & \sim_l b \\ r_4: & b & \leftarrow & 0.4 \end{array}$$

Now suppose that, given an interpretation I of P we want to ensure that in the reduct of P w.r.t. I the body of rule r_1 is equal to the expression $\max(I(b), c)$, i.e. that $(r_1^I)_b = \max(I(b), c)$. This is for example needed when the value of a should only be dependent upon the guessed value for b, which will occur frequently in the simulations occurring in the following sections. We can do this by (i) introducing a "fresh" atom not_b; (ii) adding a rule $n_b : not_b \leftarrow\sim_l b$ to P; and (iii) replacing the b in the body of rule r_1 with $\sim_l not_b$. This results in the following program P':

$$\begin{array}{rrcl} r'_1: & a & \leftarrow & \max(\sim_l not_b, c) \\ r_2: & b & \leftarrow & \sim_l c \\ r_3: & c & \leftarrow & \sim_l b \\ r_4: & b & \leftarrow & 0.4 \\ n_b: & not_b & \leftarrow & \sim_l b \end{array}$$

One can easily verify that for an interpretation I of P and its corresponding interpretation $I' = I \cup \{not_b^{\sim_l I(b)}\}$ of P' the reduct $P'^{I'}$ consists of the following rules:

$$\begin{array}{rrcl} r'^{I'}_1: & a & \leftarrow & \max(\sim_l I'(not_b), c) \\ r_2^{I'}: & b & \leftarrow & \sim_l I'(c) \\ r_3^{I'}: & c & \leftarrow & \sim_l I'(b) \\ r_4^{I'}: & b & \leftarrow & 0.4 \\ n_b^{I'}: & not_b & \leftarrow & \sim_l I'(b) \end{array}$$

By definition of I' we then have that $(r'_1)_b = \max(\sim_l I'(not_b), c) = \max(\sim_l (\sim_l I(b)), c) = \max(I(b), c)$.

In general the procedure goes as follows. Given a CFASP program P, a rule $r \in P$ and an atom $b \in r_b$. To fix the value of b in a certain position of the body of r in the reduct operation we proceed as follows:

1. Add a new rule $n_b : not_b \leftarrow\sim_i b$ to P, where not_b is a "fresh" literal and $\sim_i$ is an involutive negator.
2. Replace the position of b that needs to be fixed by $\sim_i not_b$.

Note that the above procedure works for any program with a truth lattice for which an involutive negator exists.

4. Constraints

4.1. Uses of Constraints

In classical answer set programming there are special rules called **constraints**. Constraints differ from regular rules by the omission of a head literal and are used to specify that in any valid solution, the body of the rule should not be satisfied. For example, in program P_{gc} from Section 1 the constraint *constr* specifies that two adjacent nodes should be differently colored. This is an important aspect of answer set programming and a necessary feature to elegantly describe many problem domains. Some FASP formalisms [25, 64] have a generalization of this feature. For example, in [25] rules of the following form are allowed:

$$constr : l \leftarrow f(b_1, \ldots, b_n) \tag{3}$$

where l is an element of some complete lattice $\mathcal{L}$ and f is an increasing $\mathcal{L}^n \rightarrow \mathcal{L}$ function. Due to the fact that $\leftarrow$ is interpreted as a residual implicator it follows that any model of a program incorporating a rule of the form (3), called a constraint, satisfies $I(f(b_1, \ldots, b_n)) \leq l$. We denote the language obtained by extending CFASP with rules of the form (3) above CFASP$^\perp$. The semantics of CFASP$^\perp$ are defined similarly to CFASP: an interpretation I of a program P is called an answer set iff I is a model of P and it is the least fixpoint of Π_{P^I}. Note that, in contrast to the CFASP semantics, we additionally require answer sets to be models. This is needed because in the CFASP case any answer set is a model of program, whereas this might not be the case if a program has constraints. Hence, the set of answer sets of a CFASP$^\perp$ program

P is a subset of the set of answer sets of the CFASP program $P \setminus \mathcal{C}_P$, where $\mathcal{C}_P$ is the set of all constraints of P.

Note that a constraint introduces an upper bound on the value of a body function. We can use this to ensure that the truth degree of certain atoms are constrained to a certain interval. Consider an atom a. If we wish to constrain a to the interval $[0.3, 0.8]$, we can add the following rules to a CFASP$^\bot$ program:

$$constr : 0.8 \leftarrow a$$
$$constr' : 0.7 \leftarrow \sim_l a$$

It is easy to see that any model M of these two rules satisfies $0.8 \geq M(a)$ and $0.3 \leq M(a)$, hence $M(a) \in [0.3, 0.8]$. In general we can define the following transformation:

Definition 8. *Let P be a CFASP$^\bot$ program over $([0,1], \leq)$, $a \in \mathcal{B}_P$ and assume we wish to constrain the value of a to the interval $[l, u]$ (with l and u in $[0,1]$). The **interval-constrained** version of P w.r.t. a and $[l, u]$, called P', has the following rule base:*

$$P' = P \cup \{low_a : (1 - l) \leftarrow \sim_l a\} \cup \{upp_a : u \leftarrow a\}$$

The following proposition shows that the interval-constrained version of a program indeed constrains the value of a literal in the desired way.

Proposition 1. *Let P be a CFASP$^\bot$ program over $([0,1], \leq)$ and $a \in \mathcal{B}_P$. Then if M' is an answer set of the interval-constrained version of P w.r.t. a and the interval $[l, u]$ (with l and u in $[0,1]$), called P', it holds that $M'(a) \in [l, u]$.*

4.2. Implementing Constraints

The program $C = \{c : p \leftarrow \sim p\}$ is well-known in answer set programming because it has no classical answer sets. In fact, any program for which rule c is the only rule with p in its head has no answer sets [2]. This peculiarity actually turns out to be useful in eliminating answer sets under certain conditions. For example, consider the following classical program G:

$$\begin{array}{lllll} r_1: & a & \leftarrow & \sim b \\ r_2: & b & \leftarrow & \sim a \end{array}$$

This program has answer sets $\{a\}$ and $\{b\}$. If we would like to eliminate the answer set in which b holds, we can add b to the body of rule c and add the resulting rule to the program:

$$\begin{array}{lllll} r_1: & a & \leftarrow & \sim b \\ r_2: & b & \leftarrow & \sim a \\ c_b: & p & \leftarrow & \sim p \wedge b \end{array}$$

Suppose now that A is an interpretation of $G \cup \{c_b\}$ such that $b \in A$. If $p \notin A$, then A is not a model of $G \cup \{c_b\}$ and thus A is not an answer set. If $p \in A$, then $c_b^A : p \leftarrow 0 \wedge b$. However, from this we can easily see that A is not the least fixpoint of $\Pi_{(G \cup \{c_b\})^A}$, hence A is not an answer set. This means $\{a\}$ is the only answer set of $G \cup \{c_b\}$ and the addition of rule c_b effectively eliminated answer set $\{b\}$.

Program C has *fuzzy* answer sets, however, meaning that its useful capacity to eliminate undesired answer sets has not directly been preserved in the fuzzy setting. Using the Gödel implicator together with the Łukasiewicz negation, for instance, it is not hard to see that $\{p^{0.5}\}$ is the unique answer set. Therefore, an adaptation of program C is needed to eliminate undesirable answer sets in the fuzzy case. To this end, consider the following program Z:

$$Z = \{r : p \leftarrow_m (\sim_m p > 0)\} \tag{4}$$

where rule r is defined over $([0,1], \leq)$. The body $\sim_m p > 0$ can be thought of as $f(\sim_m p)$ with f the $[0,1] \to \{0,1\}$ function defined as $f(x) = 1$ if $x > 0$ and $f(0) = 0$. This function f is increasing, so rule r is

a rule from a CFASP program. One can easily see that the only models of Z are $M_l = \{p^l\}$, $l \in]0,1]$. None of these models are answer sets, however, since for $l > 0$ we get

$$Z^{M_l} = \{r^{M_l} : p \leftarrow_m (0 > 0)\}$$

and $\Pi^*_{Z^{M_l}} = \{p^0\} \neq M_l$.

In the crisp case, the program C can be used to simulate constraints [2]; it turns out that our program Z can also be used to simulate constraints in CFASP programs. As an example, consider the CFASP$^\perp$ program P:

$$\begin{array}{rrll} r_1: & a & \leftarrow_m & \sim_l b \\ r_2: & b & \leftarrow_m & \sim_l a \\ c: & 0.5 & \leftarrow_m & a \end{array}$$

The only answer sets of this program are of the form $M_l = \{a^l, b^{1-l}\}$, with $l \in [0, 0.5]$, as rule c eliminates all solutions M where $M(a) > 0.5$. Now consider the CFASP program P':

$$\begin{array}{rrll} r_1: & a & \leftarrow_m & \sim_l b \\ r_2: & b & \leftarrow_m & \sim_l a \\ c': & c_{0.5} & \leftarrow_m & a \\ r_{0.5}: & c_{0.5} & \leftarrow_m & 0.5 \\ r'_{0.5}: & \perp & \leftarrow_m & (\sim_l \perp > 0) \wedge_m (c_{0.5} > 0.5) \end{array}$$

with $c_{0.5}$ and $\perp$ fresh atoms. Note that P' is constraint-free and that, for any $l \in [0, 0.5]$, $M'_l = M_l \cup \{c^{0.5}_{0.5}, \perp^0\}$ is an answer set of P'. Note that these are the only answer sets of P' as well. Indeed, suppose there is some fixpoint N of $\Pi_{P'}$ such that $N(a) > 0.5$. From rules c' and $r_{0.5}$ we obtain that $N(c_{0.5}) = N(a) > 0.5$, which implies that $N(\perp)$ needs to satisfy

$$\begin{aligned} N(\perp) &= \Pi_{P'}(N)(\perp) \\ &= \sup\{N(r_b) \mid r \in P'_\perp\} \\ &= N((r'_{0.5})_b) \\ &= N((\sim_l \perp > 0) \wedge_m (c_{0.5} > 0.5)) \\ &= N(\sim_l \perp > 0) \end{aligned}$$

The equality $N(\perp) = N(\sim_l \perp > 0)$ has no solution, however, as $N(\sim_l \perp > 0)$ takes on a value in $\{0, 1\}$ but for $N(\perp) = 0$ we get $N(\sim_l \perp > 0) = 1$ and for $N(\perp) = 1$ we get $N(\sim_l \perp > 0) = 0$. In general, one can verify that P and P' have corresponding answer sets, i.e. if M is an answer set of P then $M \cup \{c^{0.5}_{0.5}, \perp^0\}$ is an answer set of P and, conversely, if M' is an answer set of P', then $M' \cap \mathcal{B}_P$ is an answer set of P.

We now show that the construction used in the preceding example can be applied in general to CFASP$^\perp$ programs over the lattice $([0, 1], \leq)$. Formally, this transformation is defined as follows:

Definition 9. *Let P be a CFASP$^\perp$ program over the lattice $([0, 1], \leq)$ and let $\mathcal{K}_P = \{k \mid (k \leftarrow \alpha) \in \mathcal{C}_P\}$ be the set of constants appearing as the head of rules from $\mathcal{C}_P$ (the set of constraint rules in P). The corresponding CFASP program P' of P then contains the following rules:*

$$\begin{aligned} P' \quad = \quad & \{r'\colon a \leftarrow \alpha \mid (r\colon a \leftarrow \alpha) \in P \setminus \mathcal{C}_P\} \\ & \cup \{r'\colon c_k \leftarrow \alpha \mid (r\colon k \leftarrow \alpha) \in \mathcal{C}_P\} \\ & \cup \{r_k\colon c_k \leftarrow k \mid k \in \mathcal{K}_P\} \\ & \cup \{r'_k\colon \perp \leftarrow (\sim \perp > 0) \wedge (c_k > k) \mid k \in \mathcal{K}_P\} \end{aligned}$$

where $\wedge$ is an arbitrary t-norm, $\perp \notin \mathcal{B}_P$ and for all $k \in \mathcal{K}_P$ also $c_k \notin \mathcal{B}_P$.

The following propositions show that the answer sets of the CFASP$^\perp$ program P and its corresponding CFASP program P' coincide.

Proposition 2. *Let P be a CFASP$^{\perp}$ program and let P' be its corresponding CFASP program as defined by Definition 9. If M is an answer set of P, then $M' = M \cup \{c_k^k \mid k \in \mathcal{K}_P\} \cup \{\perp^0\}$ is an answer set of P', where $\mathcal{K}_P$ is defined as in Definition 9.*

Proposition 3. *Let P be a CFASP$^{\perp}$ program and let P' be its corresponding CFASP program as defined by Definition 9. If M' is an answer set of P', then $M = M' \cap \mathcal{B}_P$ is an answer set of P.*

5. Monotonically Decreasing Functions

A number of FASP frameworks not only allow functions that are increasing in rule bodies, but allow functions whose partial mappings are either monotonically increasing or decreasing (see for example [10, 25]). Functions with decreasing partial mappings in fact generalize negation-as-failure to functions of more than one argument: if $f(x_1, \ldots, x_n)$ decreases in its ith argument, the function increases when x_i decreases. Since x_i decreases when the maximal value that we can derive for x_i decreases, this means $f(x_1, \ldots, x_n)$ increases when the support for x_i decreases. This corresponds to the idea underlying negation-as-failure. However, it turns out that generalizing negation-as-failure to functions of more than one argument does not lead to a higher expressiveness, as we show in this section that any program with monotonically decreasing functions can be translated to a program in which the only decreasing functions are negators, i.e. to core FASP programs. First we define the syntax and semantics of programs with decreasing functions.

Definition 10. *A **general FASP rule** (short: CFASP^f rule) over a complete lattice $\mathcal{L}$ is an object of the form*

$$r : a \leftarrow f(b_1, \ldots, b_n; c_1, \ldots, c_m) \tag{5}$$

*where a, b_i and c_j are atoms and f is an $\mathcal{L}^{n+m} \to \mathcal{L}$ function that is increasing in its n first and decreasing in its m last arguments. The head, body and label of a rule are defined and denoted similar to CFASP rules. A **general FASP program** (short: CFASP^f program) over a complete lattice $\mathcal{L}$ is a set of CFASP^f rules over $\mathcal{L}$.*

Note that we only allow atoms, and not literals in CFASP^f programs. This is not a problem, however, as it is easy to see that literals of the form $\sim s$ are $\mathcal{L} \to \mathcal{L}$ functions that have no increasing arguments. Interpretations, the immediate consequence operator and models of CFASP^f programs and rules are defined similar to CFASP programs. Furthermore, a CFASP^f program is called simple if all rules are of the form $r : a \leftarrow f(b_1, \ldots, b_n;)$, i.e. if no decreasing functions occur in rule bodies. The semantics of CFASP^f programs depend on a new reduct definition, given as follows:

Definition 11. *Let $r : a \leftarrow f(b_1, \ldots, b_n; c_1, \ldots, c_m)$ be a CFASP^f rule over $\mathcal{L}$. The reduct of r w.r.t. some interpretation I, denoted r^I, is defined as*

$$r^I : a \leftarrow f'(b_1, \ldots, b_n)$$

where

$$f'(b_1, \ldots, b_n) = f(b_1, \ldots, b_n; I(c_1), \ldots, I(c_m))$$

The reduct of a general FASP program P over $\mathcal{L}$, denoted P^I, is the set of rules $\{r^I \mid r \in P\}$.

Answer sets of CFASP^f programs are then defined similar to those of CFASP programs.

Definition 12. *Let P be a CFASP^f program over $\mathcal{L}$. A model M of P is called an answer set of P iff $M = \Pi^*_{P^I}$.*

We can now show that any CFASP^f program can be simulated using a CFASP program. Intuitively the procedure works as follows. Given a rule of the form (5) above, we replace the function f in its body with a new function f' defined by $f'(b_1, \ldots, b_n, not_{c_1}, \ldots, not_{c_m};) = f(b_1, \ldots, b_n; \sim not_{c_1}, \ldots, \sim not_{c_m})$, where not_c is a new atom defined by the rule $n_c : not_c \leftarrow\sim c$, with $\sim$an involutive negator. In this way, we have replaced the decreasing function with an increasing function with literals, i.e. with the functions occurring in core FASP program rules. Note that we are using the value fixing method from Section 3.2 in this simulation.

Formally, this procedure is defined as follows.

Definition 13. *Let P be a CFASP^f program over a lattice $\mathcal{L}$. Then its corresponding CFASP program P' contains the following rules:*

$$\begin{aligned} P' \quad = \quad & \{r' : a \leftarrow f'(b_1, \ldots, b_n, not_{c_1}, \ldots, not_{c_m}) \mid (r : a \leftarrow f(b_1, \ldots, b_n; c_1, \ldots, c_m)) \in P\} \\ & \cup \{n_l : not_l \leftarrow \sim l \mid l \in \mathcal{N}_P\} \end{aligned}$$

where $\sim$ is an involutive negator on $\mathcal{L}$, $\mathcal{N}_P$ is the set of all atoms a that occur in a decreasing argument position of a function in the body of some rule in P, and f' is defined by $f'(b_1, \ldots, b_n, not_{c_1}, \ldots, not_{c_m}) = f(b_1, \ldots, b_n; \sim not_{c_1}, \ldots, \sim not_{c_m})$. Also for each $l \in \mathcal{B}_P$ it must hold that $not_l \notin \mathcal{B}_P$, i.e. the not_l atom is a "fresh" atom.

As an example, consider a CFASP^f program P with the following rules:

$$\begin{aligned} r_1 : \quad & a \quad \leftarrow_m \frac{1}{1 + b \cdot c} \\ r_2 : \quad & b \quad \leftarrow_m 0.8 \\ r_3 : \quad & c \quad \leftarrow_m 0.5 \end{aligned}$$

An answer set of P is $M = \{a^{10/14}, b^{0.8}, c^{0.5}\}$. If we apply the transformation of Definition 13 on P we obtain P' with rules:

$$\begin{aligned} r'_1 : \quad & a \quad \leftarrow_m f'(not_b, not_c) \\ r'_2 : \quad & b \quad \leftarrow_m 0.8 \\ r'_3 : \quad & c \quad \leftarrow_m 0.5 \\ n_b : \quad & not_b \quad \leftarrow_m \sim_l b \\ n_c : \quad & not_c \quad \leftarrow_m \sim_l c \end{aligned}$$

where $f'(not_b, not_c) = \dfrac{1}{1 + (\sim_l not_b) \cdot (\sim_l not_c)}$. We can then show that $M' = M \cup \{not_b^{\sim_l M(b)}, not_c^{\sim_l M(c)}\}$ is an answer set of P'. The reduct $P'^{M'}$ contains the following rules:

$$\begin{aligned} {r'}_1^{M'} : \quad & a \quad \leftarrow_m f'(not_b, not_c) \\ {r'}_2^{M'} : \quad & b \quad \leftarrow_m 0.8 \\ {r'}_3^{M'} : \quad & c \quad \leftarrow_m 0.5 \\ n_b^{M'} : \quad & not_b \quad \leftarrow_m 0.2 \\ n_c^{M'} : \quad & not_c \quad \leftarrow_m 0.5 \end{aligned}$$

where f' is defined as above. From the reduct we can see that $\Pi^*_{P'^{M'}}(b) = 0.8$, $\Pi^*_{P'^{M'}}(c) = 0.5$, $\Pi^*_{P'^{M'}}(not_b) = 0.2$ and $\Pi^*_{P'^{M'}}(not_c) = 0.5$. This leads to $\Pi^*_{P'^{M'}}(a) = 10/14$ and thus M' is the least fixpoint of $\Pi_{P'^{M'}}$, which is what we expected.

The following propositions show that this transformation preserves the answer set semantics.

Proposition 4. *Let P be a CFASP^f program and let P' be its corresponding CFASP program as defined by Definition 13. If M is an answer set of P, then $M' = M \cup \{not_l{}^{\sim M(l)} \mid l \in \mathcal{N}_P\}$ is an answer set of P'.*

Proposition 5. *Let P be a CFASP^f program and let P' be its corresponding CFASP program as defined by Definition 13. If M' is an answer set of P', then $M = M' \cap \mathcal{B}_P$ is an answer set of P.*

6. Aggregators

6.1. Simulation

In some applications, CFASP, CFASP$^\perp$ and CFASP^f can be too rigid because of their insistence to satisfy all rules completely. For example, consider the following CFASP$^\perp$ program P_{fgc} modeling a continuous graph coloring problem:

$$
\begin{array}{rrl}
gen_1: & white(X) & \leftarrow \sim_l black(X) \\
gen_2: & black(X) & \leftarrow \sim_l white(X) \\
sim_1: & sim(X,Y) & \leftarrow (white(X) \leftrightarrow white(Y)) \\
sim_2: & sim(X,Y) & \leftarrow (black(X) \leftrightarrow black(Y)) \\
constr: & 0 & \leftarrow (edge(X,Y) \wedge_m sim(X,Y))
\end{array}
$$

In this program the function $\leftrightarrow$ is defined as $x \leftrightarrow y = \min(x \rightarrow_l y, y \rightarrow_l x)$. Rules gen_1 and gen_2 intuitively generate a certain coloring of the graph, which can be rejected by constraint *constr* if two adjacent nodes are similarly colored. The two rules sim_1 and sim_2 define what it means for two nodes to be similar. For the graph depicted in Figure 2a we find that there are two answer sets, viz. $A_1 = \{edge(a,b)^1, edge(b,a)^1, sim(a,a)^1, sim(b,b)^1, sim(a,b)^0, sim(b,a)^0, white(a)^1, black(a)^0, black(b)^1, white(b)^0\}$ and $A_2 = \{edge(a,b)^1, edge(b,a)^1, sim(a,a)^1, sim(b,b)^1, sim(a,b)^0, sim(b,a)^0, black(a)^1, white(a)^0, white(b)^1, black(b)^0\}$. The graph depicted in Figure 2b has no answer sets however. This is not ideal, as in many cases we would like to find a solution, even if it is not optimal (i.e. satisfies all rules completely). Hence, we might find it tolerable if some nodes are colored similarly, as long as the similarity is not too high. For the graph in Figure 2b this could mean that a solution coloring node a completely white, node b completely black and node c white to degree 0.5 is still acceptable.

The solution proposed in [25, 64] is to allow rules to be partially fulfilled and use an aggregator to attach a score to answer sets, corresponding to their suitability as a model of the program. Formally an **aggregated FASP (AFASP) program** P is a tuple $\langle \mathcal{R}_P, \mathcal{A}_P \rangle$, where $\mathcal{R}_P$ is a CFASP program and $\mathcal{A}_P$ is an increasing[3] function mapping $\mathcal{R}_P \rightarrow \mathcal{L}$ functions, called **rule interpretations**, to a value in $\mathcal{L}$. The semantics of AFASP programs are defined using an adapted immediate consequence operator $\Pi_{P,\rho}$, relative to an AFASP program P and rule interpretation ρ of P, which is defined for an interpretation I of P and atom $a \in \mathcal{B}_P$ as follows:

$$\Pi_{P,\rho}(I)(a) = \sup\{I_s(r, \rho(r)) \mid r \in (\mathcal{R}_P)_a\} \tag{6}$$

where $I_s(r,w) = \inf\{l \in \mathcal{L}_P \mid (I(r_b) \rightarrow_r l) \geq w\}$ is called the **support** of a rule of P w.r.t. an interpretation and a weight $w \in \mathcal{L}_P$. In [25] we have shown that if rules are interpreted using a residual implicator we have $I_s(r,w) = I(r_b) \wedge_r w$. An interpretation I is an m-**answer set** ($m \in \mathcal{L}$) of an AFASP program P iff $I = \Pi^*_{P^I,\rho_I}$ and $\mathcal{A}_P(\rho_I) \geq m$, where for each $r \in P$ we have $\rho_I(r) = I(r)$ and $P^I = \langle (\mathcal{R}_P)^I, \mathcal{A}_P \rangle$.

For example, we can combine the CFASP program P'_{fgc}, which is the constraint-free version of the CFASP$^\perp$ program P_{fgc} above, with the aggregator $\mathcal{A}(\rho) = \alpha(\rho) \cdot \beta(\rho)$ where

$$
\begin{array}{rcl}
\alpha(\rho) & = & \prod\{\rho((gen'_1)_a) \cdot \rho((gen'_2)_a) \cdot \rho((sim'_1)_{a,b}) \cdot \rho((sim'_2)_{a,b}) \\
& & \quad \cdot \rho(r_0) \cdot \rho(r'_0) \mid a,b \in Nodes\} \geq 1 \\
\\
\beta(\rho) & = & 0.3 \cdot \rho(constr'_{a,b}) + 0.3 \cdot \rho(constr'_{b,a}) \\
& & + 0.1 \cdot \rho(constr'_{a,c}) + 0.1 \cdot \rho(constr'_{c,a}) \\
& & + 0.1 \cdot \rho(constr'_{b,c}) + 0.1 \cdot \rho(constr'_{c,b})
\end{array}
$$

to create the AFASP program $G_{fgc} = \langle P'_{fgc}, \mathcal{A} \rangle$. Note that with gen'_a we denote the grounding of rule gen' with node a. Intuitively, this aggregator denotes that the constraint concerning nodes a and b is most important, while the constraints concerning nodes a and c and nodes b and c are of equal importance. One can then easily verify that for graph 2b the following interpretation[4] $A_{0.6} = \{edge(a,b)^1, edge(b,a)^1, edge(a,c)^1, edge(c,a)^1, edge(b,c)^1, edge(c,b)^1, sim(a,a)^1, sim(b,b)^1, sim(c,c)^1, white(a)^1, black(b)^1, white(c)^{0.5}, black(c)^{0.5}, sim(a,c)^{0.5}, sim(c,a)^{0.5}, sim(b,c)^{0.5}, sim(c,b)^{0.5}\}$ is an 0.6-answer set of G_{fgc}. Hence, using AFASP we can find approximate answer sets.

At first sight, one might be tempted to think that a program with an aggregator can be replaced by a core FASP program P' such that $P' = \mathcal{R}_P \cup \{r'_{aggr} : aggr \leftarrow f((r_1)_h \leftarrow_{r_1} (r_1)_b, \ldots, (r_n)_h \leftarrow_{r_n} (r_n)_b)\}$,

[3] Where $\mathcal{R}_P \rightarrow \mathcal{L}$ functions are ordered pointwise, i.e. $\rho_1 \leq \rho_2$ iff for each $r \in \mathcal{R}_P$ we have $\rho_1(r) \leq \rho_2(r)$.

[4] Note that, as remarked in the preliminaries, we do not include atoms with truth value 0 – such as c and $\perp$ – in the enumeration of an interpretation.

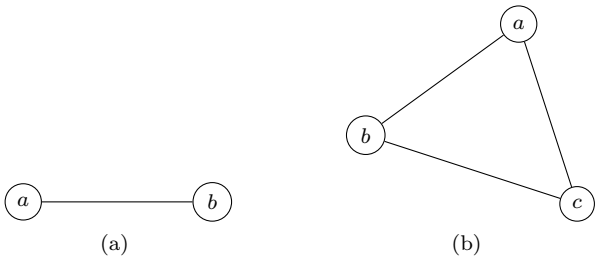

Figure 2: Example instances for the graph coloring problem.

where f corresponds to the function defined by the aggregator of P. The intended meaning is such that M is an m-answer set of P iff $M \cup \{aggr^m\}$ is an answer set of P'.

This trivial translation is not correct, however, as it does not correctly incorporate the notion of partial rule satisfaction. For example, consider the AFASP program P with rule base $\mathcal{R}_P = \{(r_1 : a \leftarrow 1), (r_2 : b \leftarrow 1)\}$ and with aggregator $\mathcal{A}_P(\rho) = \inf\{\rho(r) \mid r \in \mathcal{R}_P\}$. Using the transformation proposed above, we obtain the core FASP program $P' = \{(r'_1 : a \leftarrow 1), (r'_2 : b \leftarrow 1), (r'_{aggr} : aggr \leftarrow \inf((a \leftarrow 1), (b \leftarrow 1))\}$. For the 0.7-answer set $M = \{a^{0.7}, b^1\}$ of P the corresponding interpretation $M' = M \cup \{(r'_{aggr})^{0.7}\}$ is not an answer set of P', however, as P' only has one answer set, viz. $\{a^1, b^1, (r'_{aggr})^1\}$. This problem can be solved in the following way.

Definition 14. *Let P be an AFASP program with rule base $\mathcal{R}_P = \{r_1, \ldots, r_n\}$ over the lattice $\mathcal{L}$ and suppose an involutive negator $\sim$ exists over this lattice. Its corresponding CFASP^f program, denoted as P', contains the following rules:*

$$\begin{aligned} P' \quad = \quad & \{r' : a \leftarrow (\alpha \wedge_r r'_i) \mid (r : a \leftarrow \alpha) \in \mathcal{R}_P\} \\ & \cup \{n_a : not_a \leftarrow \sim a \mid a \in \mathcal{B}_P\} \\ & \cup \{r'_\rho : r'_i \leftarrow (\alpha' \rightarrow_r (\sim not_a)) \mid (r : a \leftarrow \alpha) \in \mathcal{R}_P\} \\ & \cup \{r'_{aggr} : aggr \leftarrow f(r'_{1i}, \ldots, r'_{ni})\} \end{aligned}$$

Furthermore, all literals not_a for $a \in \mathcal{B}_P$, r'_i for $r \in \mathcal{R}_P$ and $aggr$ are literals not occurring in $\mathcal{B}_P$. Also, the function f corresponds to $\mathcal{A}_P$ in the sense that $\mathcal{A}_P(\rho) = f(\rho(r_1), \ldots, \rho(r_n))$ and α' is obtained from α by replacing each naf-literal $\sim_a a$ in α with $\sim_a(\sim not_a)$, where $\sim_a$ is the negator associated with the naf-literal.

Note that due to the r'_ρ rules in Definition 14, we translate an AFASP program to a CFASP^f program, rather than a CFASP program. This is not a problem however, as we have shown in Section 5 that any CFASP^f program can be translated to a corresponding CFASP program.

Example 2. *Consider program P_1 with rule base $\mathcal{R}_{P_1}$:*

$$\begin{aligned} r_1 : \quad & a \quad \leftarrow_m \sim_l b \\ r_2 : \quad & b \quad \leftarrow_m 0.7 \end{aligned}$$

The aggregator of P_1 is $\mathcal{A}_{P_1}(\rho) = \inf\{\rho(r) \mid r \in \mathcal{R}_{P_1}\}$. Applying Definition 14, we obtain the CFASP^f program P'_1 with rules:

$$\begin{aligned} r'_1 : \quad & a \quad \leftarrow_m (\sim_l b) \wedge_m r'_{1i} \\ r'_2 : \quad & b \quad \leftarrow_m 0.7 \wedge_m r'_{2i} \\ n_a : \quad & not_a \quad \leftarrow_m \sim_l a \\ n_b : \quad & not_b \quad \leftarrow_m \sim_l b \\ r'_{1\rho} : \quad & r'_{1i} \quad \leftarrow_m ((\sim_l (\sim_l not_b)) \rightarrow_m (\sim not_a)) \\ r'_{2\rho} : \quad & r'_{2i} \quad \leftarrow_m (0.7 \rightarrow_m (\sim not_b)) \\ r'_{aggr} : \quad & aggr \quad \leftarrow_m \inf(r'_{1i}, r'_{2i}) \end{aligned}$$

Consider now the 1*-answer set* $M = \{a^{0.3}, b^{0.7}\}$ *of* P_1*. Our intention is that Definition 14 is constructed in such a way that* $M' = \{a^{0.3}, b^{0.7}, {r'_{1_i}}^1, {r'_{2_i}}^1, not_a^{0.7}, not_b^{0.3}, {r'_{aggr}}^1\}$ *is an answer set of* P'_1*. One can verify that this is indeed the case.*

Note that we are using the value fixing method from Section 3.2 in our simulation. This is needed to correctly preserve the semantics. To see this, consider the following alternative program P''_1:

$$\begin{array}{rrl} r'_1: & a & \leftarrow_m (\sim_l b) \wedge_m r'_{1_i} \\ r'_2: & b & \leftarrow_m 0.7 \wedge_m r'_{2_i} \\ r'_{1_\rho}: & r'_{1_i} & \leftarrow_m ((\sim_l b) \rightarrow_m a) \\ r'_{2_\rho}: & r'_{2_i} & \leftarrow_m (0.7 \rightarrow_m b) \\ r'_{aggr}: & aggr & \leftarrow_m \inf(r'_{1_i}, r'_{2_i}) \end{array}$$

Now consider the 1-answer set $M = \{a^{0.3}, b^{0.7}\}$ of P_1. We wish the aggregator-free version of P_1 to be constructed in such a way that $M' = M \cup \{r_i'^{M(r)} \mid r \in \mathcal{R}_P\} \cup \{aggr^{\mathcal{A}_P(\rho_M)}\}$ is an answer set of P''_1. Hence in the case of P''_1, we find that $M' = M \cup \{{r'_{1_i}}^1, {r'_{2_i}}^1\} \cup \{aggr^1\}$ should be an answer set of P''_1. However, there is an $M'' < M'$ such that M'' is a fixpoint of $\Pi_{P''^{M'}_1}$, which contradicts the fact that M' is an answer set of P''_1. Indeed, for $M'' = \{a^{0.2}, b^{0.7}\} \cup \{{r'_{1_i}}^{0.2}, {r'_{2_i}}^1\} \cup \{aggr^{0.2}\}$ it can be seen that M'' is a model of P'_1 and a fixpoint of $\Pi_{P''^{M'}_1}$ as follows. For a we obtain:

$$\Pi_{P''^{M'}_1}(M'')(a) = \sim_l M'(b) \wedge_m M''(r'_{1_i}) = (1 - 0.7) \wedge_m 0.2 = 0.2$$

Likewise we obtain that $\Pi_{P''^{M'}_1}(M'')(b) = 0.7$. Now for r'_{1_i} we obtain

$$\begin{aligned} \Pi_{P''^{M'}_1}(M'')(r'_{1_i}) &= M''(((\sim_l b) \rightarrow_m a)^{M'}) \\ &= M''(M'(\sim_l b) \rightarrow_m a) \\ &= (\sim_l M'(b)) \rightarrow_m M''(a) \\ &= 0.3 \rightarrow_m 0.2 \\ &= 0.2 \end{aligned}$$

Similarly we find that $\Pi_{P''^{M'}_1}(M'')(r'_{2_i}) = 1$. Lastly, for $aggr$ we obtain

$$\Pi_{P''^{M'}_1}(M'')(aggr) = M''(\inf(r'_{1_i}, r'_{2_i})) = \inf(0.2, 1) = 0.2$$

Hence M'' is a fixpoint of $\Pi_{P''^{M'}_1}$, contradicting that M' is an answer set of P''_1.

The problem the preceding example illustrates is that we must be able to "fix" the value of the literals r'_{1_i} and r'_{2_i} when we are taking the reduct relative to M'. The only way to ensure this is by eliminating all literals from the body of the r'_{1_ρ} and r'_{2_ρ} rules by means of the reduct procedure. Hence we must replace each positively occurring literal in the bodies of r'_{1_ρ} and r'_{2_ρ} by a negatively occurring literal; this is done by replacing a positively occurring literal a with $\sim_l not_a$, which preserves the same value, but will be replaced by the reduct operation.

The following propositions show that our translation preserves the semantics.

Proposition 6. *Let* P *be an AFASP program with* $\mathcal{R}_P = \{r_1, \ldots, r_n\}$ *and let* P' *be its corresponding CFASP*^{f} *program as defined by Definition 14. If* M *is an* m*-answer set of* P*, then* $M' = M \cup \{aggr^{\mathcal{A}_P(\rho_M)}\} \cup \{not_a^{\sim M(a)} \mid a \in \mathcal{B}_P\} \cup \{r_i'^{M(r)} \mid r \in \mathcal{R}_P\}$ *is an answer set of* P'*.*

Proposition 7. *Let* P *be an AFASP program and let* P' *be its corresponding CFASP*^{f} *program as defined by Definition 14. If* M' *is an answer set of* P'*, with* $m = M'(aggr)$*, then* $M' \cap \mathcal{B}_P$ *is an* m*-answer set of* P*.*

Sometimes it is convenient to extend AFASP programs with constraints or arbitrary functions that are monotonic in their partial mappings. We can easily translate programs with these extensions to regular AFASP programs however, as we will now show. First, define the sets AFASP$^{\perp}$ and AFASP^{f} as the set of AFASP programs where the rule base is a program in CFASP$^{\perp}$, respectively CFASP^{f}. Then we can define the following translations:

Definition 15. *Let P be an AFASP$^{\perp}$ program over the lattice $([0,1],\leq)$ and let $\mathcal{K}_P = \{k \mid (k \leftarrow \alpha) \in \mathcal{C}_P\}$ be the set of constants appearing as the head of rules from $\mathcal{C}_P$ (the set of constraint rules in $\mathcal{R}_P$). The corresponding AFASP program P' of P then consists of the rulebase $\mathcal{R}'_P$, i.e. the CFASP program corresponding to the CFASP$^{\perp}$ program $\mathcal{R}_P$ as defined by Definition 9, and the following aggregator:*

$$\mathcal{A}_{P'}(\rho) = \begin{cases} \mathcal{A}_P(\rho) & \text{if } \forall k \in \mathcal{K}_P \colon \rho(r_k) \geq 1 \wedge \rho(r'_k) \geq 1 \\ 0 & \text{otherwise} \end{cases}$$

where r_k and r'_k are as defined by Definition 9.

Definition 16. *Let P be an AFASP^{f} program over a lattice $\mathcal{L}$. The corresponding AFASP program P' consists of the rulebase $\mathcal{R}'_P$, i.e. the CFASP program corresponding to the CFASP^{f} program $\mathcal{R}_P$ as defined by Definition 13, and the following aggregator:*

$$\mathcal{A}_{P'}(\rho) = \begin{cases} \mathcal{A}_P(\rho) & \text{if } \forall l \in \mathcal{N}_P \colon \rho(n_l) \geq 1 \\ 0 & \text{otherwise} \end{cases}$$

where $\mathcal{N}_P$ and n_l are as defined by Definition 13.

The following propositions show that the above definitions preserve the semantics.

Proposition 8. *Let P be an AFASP$^{\perp}$ program and let P' be its corresponding AFASP program as defined by Definition 15. If M is an m-answer set of P for some $m \in]0,1]$, it holds that $M' = M \cup \{c_k^k \mid k \in \mathcal{K}_P\} \cup \{\perp^0\}$ is an m-answer set of P', where $\mathcal{K}_P$ is as defined by Definition 15.*

Proposition 9. *Let P be an AFASP$^{\perp}$ program and let P' be its corresponding AFASP program as defined by Definition 15. If M' is an m-answer set of P' for some $m \in]0,1]$, it holds that $M = M' \cap \mathcal{B}_P$ is an m-answer set of P.*

Proposition 10. *Let P be an AFASP^{f} program and let P' be its corresponding AFASP program as defined by Definition 16. If M is an m-answer set of P for some $m \in \mathcal{L}_P \setminus \{0\}$, it holds that $M' = M \cup \{not_l^{\sim M(l)} \mid l \in \mathcal{N}_P\}$ is an m-answer set of P', where $\mathcal{N}_P$ is as defined by Definition 16.*

Proposition 11. *Let P be an AFASP^{f} program and let P' be its corresponding AFASP program as defined by Definition 16. If M' is an m-answer set of P for some $m \in \mathcal{L}_P \setminus \{0\}$, it holds that $M = M' \cap \mathcal{B}_P$ is an m-answer set of P.*

6.2. Simulation Technique: Value Injection

In the simulations that follow in the remainder of this paper it is often necessary to "inject" the value of an atom in the aggregator. For example, consider the AFASP program P on the lattice $([0,1],\leq)$ with the following rule base:

$$\begin{aligned} r_1 \colon &\quad a \quad \leftarrow_m \sim_l b \\ r_2 \colon &\quad b \quad \leftarrow_m \sim_l a \end{aligned}$$

Now, suppose that we want to attach an aggregator that uses the truth value of atom a to P. Aggregators only have access to the truth values of rules, however, so we need to find a way to propagate the truth

value of a to the truth value of a certain rule. In the example we can do this by inserting a new constraint $i_a : 0 \leftarrow_l \sim_l a$ to P, resulting in the altered program P' with the following rule base:

$$\begin{array}{lrll} r_1: & a & \leftarrow_m & \sim_l b \\ r_2: & b & \leftarrow_m & \sim_l a \\ i_a: & 0 & \leftarrow_l & \sim_l a \end{array}$$

For any interpretation I of P' we have that $\rho_I(i_a) = (\sim_l I(a)) \rightarrow_l 0 = \sim_l (\sim_l I(a)) = I(a)$, since for all $x \in [0,1]$ it holds that $x \rightarrow_l 0 = \sim_l x$. Hence, we have successfully transferred the value of a into the degree to which a rule is satisfied.

In general, for an AFASP program P, if we need to use the value of an atom $a \in \mathcal{B}_P$ in the aggregator of P, we can proceed as follows:

1. Add a new constraint to $\mathcal{R}_P$ of the form $i_a : 0 \leftarrow_i \sim_i a$, where $\leftarrow_i$ corresponds to an implicator over $\mathcal{L}_P$ for which an involutive negator exists with the property that for any $x \in \mathcal{L}_P$ we have that $x \rightarrow_i 0 = \sim_i 0$.
2. In the aggregator, refer to i_a everytime the value of a is needed.

Note that this procedure works for any program that uses a truth lattice for which there exists an involutive negator $\sim_i$ that is induced by an implicator $\rightarrow_i$ (i.e. for which it holds that for any $x \in \mathcal{L}_P$ we have that $x \rightarrow_i 0 = \sim_i 0$). It is clear that in $[0,1]$ the Łukasiewicz implication satisfies this criterium. In general every MV-algebra satisfies this requirement by definition [19].

7. S-implicators

Occasionally, when using aggregators, the ability to use other types of implicators in rules could be useful. If the Kleene-Dienes implicator were used, for example, then $r : a \leftarrow f(b_1, \ldots, b_n)$ would only evaluate to 1 if either the body evaluates to 0, or the head evaluates to 1. This means that as soon as $I(f(b_1, \ldots, b_n)) > 0$, the rule is triggered and the head is taken to be completely true.

Example 3. *Consider the following rules, encoding that we want to have a barbecue, unless the weather is bad:*

$$\begin{array}{lrll} r_1: & bad\_weather & \leftarrow_{kd} & rain \\ r_2: & bad\_weather & \leftarrow_l & \sim sunshine \\ r_3: & bbq & \leftarrow_l & \sim bad\_weather \end{array}$$

where rain is the degree to which it is raining and sunshine is the expected amount of sunshine. Because the Kleene–Dienes implicator is used in the first rule, a barbecue is out of the question even if it rains only a little bit (e.g. drizzle). If it is not raining, our motivation for having a barbecue depends linearly on the amount of sunshine, hence a Łukasiewicz implicator is used.

However, as the rules in the programs we discussed so far are restricted to residual implicators, we are not directly able to write a rule as in Example 3 in the AFASP programs introduced in Section 6. Now consider the language AFASP^s, which is AFASP extended with S-implicator rules. To extend the semantics of AFASP programs with S-implicator rules, we need to determine their support. The following proposition shows how this support can be computed:

Proposition 12. *Let $\mathcal{L}$ be a lattice, $\sim$ an involutive negator over $\mathcal{L}$, $\wedge$ a t-norm over $\mathcal{L}$, $\rightarrow_r$ the residual implicator of $\wedge$, $\vee$ the t-conorm defined by $x \vee y = \sim(\sim x \wedge \sim y)$ and let $\leftarrow_s$ be the S-implicator induced by $\sim$ and $\vee$. Then for any interpretation I of a rule $r : a \leftarrow_s \alpha$ and $w \in \mathcal{L}$ it holds that:*

$$I_s(r, w) = \sim(\sim w \leftarrow_r I(\alpha))$$

Using the support calculated above, the semantics of an AFASP^s program can be defined similar to Section 6. We can transform an AFASP^s program to an equivalent AFASP program.

Definition 17. *Let P be an AFASP^s program with rule base $\mathcal{R}_P = \mathcal{R}^r_P \cup \mathcal{R}^s_P$ such that $\mathcal{R}^r_P$ is the set of rules constructed from residual implicators and $\mathcal{R}^s_P$ is the set of rules constructed from S-implicators. Furthermore assume that $\mathcal{L}_P$ is such that there is an implicator $\leftarrow_i$ that induces an involutive negator $\sim_i$. Then the rulebase of the AFASP^f program P' corresponding to P is defined as:*

$$\begin{aligned}\mathcal{R}_{P'} \;=\; & \mathcal{R}^r_P \\ & \cup \{r' : a \leftarrow\!\sim_i (not_{w_{r'}} \leftarrow_r \alpha) \mid r : a \leftarrow_s \alpha \in \mathcal{R}^s_P\} \\ & \cup \{r'_w : w_{r'} \leftarrow (\sim_i not_a \leftarrow_s \alpha) \mid r : a \leftarrow_s \alpha \in \mathcal{R}^s_P\} \\ & \cup \{n_a : not_a \leftarrow\!\sim_i a \mid r : a \leftarrow_s \alpha \in \mathcal{R}^s_P\} \\ & \cup \{r'_{nw_{r'}} : not_{w_{r'}} \leftarrow\!\sim_i w_{r'} \mid r : a \leftarrow_s \alpha \in \mathcal{R}^s_P\} \\ & \cup \{r'_c : 0_{\mathcal{L}} \leftarrow_i \sim_i w_{r'} \mid r : a \leftarrow_s \alpha \in \mathcal{R}^s_P\}\end{aligned}$$

where $\leftarrow_r$ for each $r : a \leftarrow_s \alpha \in \mathcal{R}^s_P$ is the residual implicator induced by the same t-norm as $\leftarrow_s$. The literals $w_{r'}$, not_a and $not_{w_{r'}}$ are fresh literals not contained in $\mathcal{B}_P$. Furthermore, $\leftarrow_i$ is a residual implicator that induces the involutive negator $\sim_i$ and the implicator $\leftarrow$ is an arbitrary residual implicator. The aggregator of P' is defined as:

$$\mathcal{A}_{P'}(\rho) = \begin{cases} (\mathcal{A}_P)(\rho') & \text{if } \rho(r') \wedge \rho(r'_w) \wedge \rho(n_a) \wedge \rho(r'_{nw_{r'}}) \geq 1_{\mathcal{L}} \\ 0_{\mathcal{L}} & \text{otherwise} \end{cases}$$

where $\rho'(r) = \rho(r'_c)$ for any $r \in \mathcal{R}^s_P$ and $\rho'(r) = \rho(r)$ for any $r \in \mathcal{R}_P \setminus \mathcal{R}^s_P$.

The limitation on the lattice states that the lattice must have one negator defined using $x \to_i 0$ that satisfies $(x \to_i 0) \to_i 0 = x$. This constraint is needed to properly define the support of the rule with an S-implicator, as is also apparent from Proposition 12. Furthermore it is needed to use the value injection method from Section 6.2 and ensures that we can use the value fixing method from Section 3.2 in this simulation.

Also note that the program defined above still contains S-implicators. However, they appear in the rule bodies, and not as the implication associated to a rule. Hence, we have reduced the semantics of a program with mixed S-implication and residual implication rules to a program solely consisting of the latter. The occurrence of the S-implicators in the rule bodies is responsible for the program being an AFASP^f program. This is not a problem, as in Section 6 we have shown that such a program can be simulated using an AFASP program. Furthermore note that for any interpretation I of P and corresponding interpretation I' of P' we have $\rho_I(r'_c) = I(w_{r'})$. In this way the aggregator expression obtains the same value for any interpretation of P' as it does for P. Further note that the construction with not_a is necessary to be able to fix the value of not_a w.r.t. a certain reduct as in Section 6.

As an example, consider the AFASP^s program from Example 3. The corresponding AFASP program contains the following rule base

$$\begin{aligned}
r_2 :&\quad bad_weather &&\leftarrow_l \sim sunshine \\
r_3 :&\quad bbq &&\leftarrow_l \sim bad_weather \\
{r_1}' :&\quad bad_weather &&\leftarrow_l \sim_l (not_{w_{r'_1}} \leftarrow_m rain) \\
{r_1}'_w :&\quad w_{r'_1} &&\leftarrow_l (\sim_l not_{bad_weather} \leftarrow_{kd} rain) \\
{r_1}'_{not_{bad_weather}} :&\quad not_{bad_weather} &&\leftarrow_l (\sim_l bad_weather) \\
{r_1}'_{nw_{r'_1}} :&\quad not_{w_{r'_1}} &&\leftarrow_l \sim_l w_{r'_1} \\
{r_1}'_c :&\quad 0 &&\leftarrow_l \sim_l w_{r'_1}
\end{aligned}$$

Suppose we add some facts that tell us that it is raining to a degree of 0.2 and it is sunny to a degree of 0.7. The 1-answer set we obtain for the program from Example 3 is $A = \{rain^{0.2}, sunny^{0.7}, bad\_weather^1, bbq^0\}$. One can verify that the 1-answer set of the S-implicator free version is $A' = A \cup \{w^1_{r'_1}\} \cup \{not^0_{bad\_weather}\} \cup \{not^0_{w_{r'_1}}\}$.

The following propositions show that the answer sets of the AFASP^s program coincide with those of the corresponding AFASP program. Hence AFASP^s programs can be translated to equivalent CFASP programs using the results from Section 6.

Proposition 13. *Let P be an AFASP^s program and let P' be the corresponding AFASP program as defined by Definition 17. If M is an m-answer set ($m \in \mathcal{L}_P$ and $m > 0_\mathcal{L}$) of P, it holds that $M' = M \cup \{w_{r'}^{\rho_M(r)} \mid r \in \mathcal{R}_P^s\} \cup \{not_a^{\sim_i M(a)} \mid a \in \mathcal{B}_P, \exists r \in \mathcal{R}_P^s : r_h = a\} \cup \{not_{w_{r'}}^{\sim_i \rho_M(r)} \mid r \in \mathcal{R}_P^s\}$ is an m-answer set of P', where $\sim_i$ is as defined by Definition 17.*

Proposition 14. *Let P be an AFASP^s program and let P' be the corresponding AFASP program as defined by Definition 17. If M' is an m-answer set ($m \in \mathcal{L}_{P'}$ and $m > 0_\mathcal{L}$) of P', it holds that $M = M' \cap \mathcal{B}_P$ is an m-answer set of P.*

8. Strong Negation

In classic ASP, besides negation-as-failure, there is a second form of negation called **strong negation**[5]. This form of negation is used when explicit derivation of negative information is needed. The resulting semantic difference can be very important. For example, if we wish to state that it is safe to cross the train tracks when no train is coming we write the following program when using negation-as-failure (from [2]):

$$cross \quad \leftarrow \quad \sim train$$

This however means that when the information about a train coming is absent, we cross the tracks, which is not the safest thing to do. With strong negation, the problem is written as:

$$cross \quad \leftarrow \quad \neg train$$

where $\neg train$ is a special literal that can appear in the head of rules. As the value of $\neg train$ is derived using the rules of the program and not derived by the absence of information about $train$, we only cross the tracks when we can explicitly derive that no train is coming. Of course, when both a and $\neg a$ appear in the head of rules there is the possibility of inconsistency. The usual semantics for ASP determine that whenever the standard definition would lead to an answer set of a program P in which both a and $\neg a$ occur, by definition, the only answer set of P is given by $Lit(P) = \mathcal{B}_P \cup \{\neg a \mid a \in \mathcal{B}_P\}$ (see [2]). A program in which this occurs is inconsistent and, as in classical propositional logic, anything can be derived from an inconsistent program.

In [64] the inconsistency problem is solved by attaching to each interpretation I of a program P and literal $a \in \mathcal{B}_P$ a score of consistency $I_c(a) = \sim_c(I(a) \wedge_c I(\neg a))$, where $\sim_c$ is a negator and $\wedge_c$ a t-norm. The interpretation I is then called **x-consistent**, with $x \in \mathcal{L}_P$, iff $\mathcal{A}_c(I_c) \geq x$, where $\mathcal{A}_c$ is the consistency aggregator that maps $\mathcal{B}_P$ and I_c to a global consistency score for I. This consistency aggregator, which differs from the regular aggregator, is required to be increasing when the consistencies for literals increase.

It is well-known that ASP programs with strong negation can be translated to equivalent programs without strong negation by substituting a new literal a' and adding the constraint $\leftarrow a, a'$ for each $\neg a \in Lit(P)$. The resulting program has no consistent answer sets iff program P has the unique answer set $Lit(P)$.

We can generalize the procedure for eliminating strong negation in classical ASP to fuzzy programs and embed strong negation in AFASP. In particular, to implement the strong negation approach of [64] into AFASP, we proceed as follows.

Definition 18. *Let P be an AFASP program with strong negation, i.e. with $\neg a \in \mathcal{B}_P$ for some literals $a \in \mathcal{B}_P$ and let $\sim_c$, $\wedge_c$ and $\mathcal{A}_c$ be the negator, t-norm and aggregator expression determining the consistency score of P w.r.t. some interpretation. Furthermore assume that we can define an implicator $\rightarrow_i$ in $\mathcal{L}_P$ that induces an involutive negator $\sim_i$. Then P', the strong-negation free version of P is defined as follows:*

1. $\mathcal{B}_{P'} = (\mathcal{B}_P \setminus \{\neg a \mid a \in \mathcal{B}_P\}) \cup \{a' \mid \neg a \in \mathcal{B}_P\}$
2. $\mathcal{R}_{P'} = \{r : a \leftarrow \alpha' \mid r : a \leftarrow \alpha \in \mathcal{R}_P, a \in \mathcal{B}_P \setminus \{\neg a \mid a \in \mathcal{B}_P\}\}$
   $\cup \{r : a' \leftarrow \alpha' \mid r : \neg a \leftarrow \alpha \in \mathcal{R}_P, a \in \mathcal{B}_P \setminus \{\neg a \mid a \in \mathcal{B}_P\}\}$
   $\cup \{c_a : 0 \leftarrow_i (\sim_i(\sim_c(a \wedge_c a'))) \mid a \in \mathcal{B}_P \setminus \{\neg a \mid a \in \mathcal{B}_P\}\}$

[5]This negation is often referred to as **classical negation** as well, not to be confused with negation over $\{0, 1\}$.

3. $\mathcal{A}_{P'}(\rho) = (\mathcal{A}_c(\{a^{c_a} \mid a \in \mathcal{B}_P\}), \mathcal{A}_P(\rho))$

where for a rule $r : a \leftarrow \alpha \in \mathcal{R}_P$ we define α' as the expression obtained by replacing each $\neg a$ for the corresponding a'. Last, for each $a \in \mathcal{B}_P$ it must hold that $a' \notin \mathcal{B}_P$, i.e. a' is a "fresh" literal.

Technically, we first replace all classically negated literals with a fresh variable. Then we "inject" the value of $I_c(a)$ for a literal a into the aggregator of the new program using the method explained in Section 6.2 (this is done in the c_a rules). Last, we create the new aggregator as a tuple of the consistency degree and the old aggregator, allowing us to order answer sets using both measures. Note that for an AFASP program P this aggregator obtains a value in $\mathcal{L}_P^2$, which is a total lattice.

Example 4. *Consider an AFASP program P with rule base $\mathcal{R}_P$:*

$$\begin{array}{lrll} r_1: & a & \leftarrow_m & \neg b \\ r_2: & \neg b & \leftarrow_m & 0.2 \\ r_3: & \neg a & \leftarrow_m & 0.4 \end{array}$$

The rule aggregator is defined as $\mathcal{A}_P(\rho) = \min(1, \rho(r_1) + \rho(r_2) + \rho(r_3))$ and the consistency aggregator as $\mathcal{A}_c(I_c) = \inf\{I_c(a) \mid a \in \mathcal{B}_P\}$. For all literals the consistency negator $\sim_c$ is $\sim_l$ and the consistency t-norm $\wedge_c$ is $\wedge_m$. According to Definition 18, the strong negation free version of P is a program P' with rule base $\mathcal{R}_{P'}$:

$$\begin{array}{lrll} r_1: & a & \leftarrow_m & b' \\ r_2: & b' & \leftarrow_m & 0.2 \\ r_3: & a' & \leftarrow_m & 0.4 \\ c_a: & 0 & \leftarrow_l & \sim_l(\sim_l(a \wedge_m a')) \\ c_b: & 0 & \leftarrow_l & \sim_l(\sim_l(b \wedge_m b') \end{array}$$

The aggregator of P' is $\mathcal{A}_{P'} = (\mathcal{A}_c(\{a^{c_a}, b^{c_b}\}), \mathcal{A}_P)$. Now, consider an interpretation $I = \{a^{0.2}, (\neg b)^{0.2}, (\neg a)^{0.4}\}$ of P and the corresponding interpretation $I' = \{a^{0.2}, b'^{0.2}, a'^{0.4}\}$ of P'. Computing I_c we obtain that $I_c(a) = 1 - (I(a) \wedge_m I(\neg a)) = 1 - (0.2 \wedge_m 0.4) = 0.8$ and likewise $I_c(b) = 1 - (I(b) \wedge_m I(\neg b)) = 1 - (0 \wedge_m 0.2) = 1$. Computing $\rho_{I'}$ we easily obtain $\rho_{I'}(r_1) = \rho_{I'}(r_2) = \rho_{I'}(r_3) = 1$; for c_a and c_b we obtain $\rho_{I'}(c_a) = \sim_l(\sim_l(a \wedge_m a') \rightarrow_l 0) = 1 - (I_c(a) \rightarrow_l 0) = 1 - (1 - I_c(a)) = I_c(a)$ and likewise for c_b we get $\rho_{I'}(c_b) = I_c(b)$. The evaluation of $\mathcal{A}_{P'}$ with $\rho_{I'}$ then gives $\mathcal{A}_{P'}(\rho_{I'}) = (\mathcal{A}_c(\{a^{I_c(a)}, b^{I_c(b)}\}), \mathcal{A}_P(\rho_{I'})) = (\mathcal{A}_c(I_c), \mathcal{A}_P(\rho_{I'}))$. Hence, the aggregator indeed maps an interpretation to a tuple containing the consistency degree and the rule aggregation score.

The following proposition links our definition to the strong negation approach of [64].

Proposition 15. *Let P be an AFASP program with strong negation and let P' be its strong-negation free version. Then an interpretation I' of P' is an (x, y)-answer set of P iff the corresponding interpretation I of P is an x-consistent y-answer set in the sense of [64].*

9. Relationship to Existing Approaches

The study of extensions of ASP has received plenty of attention over the past years, including the efforts of the European Working Group on Answer Set Programming (WASP) [51]. The main objectives of such a study are (1) researching the complexity and additional expressivity certain extensions bring; (2) investigating whether extensions can be compiled to a core language that is easy to implement, or is already implemented. Certain interesting links have been brought to light in this research. For example, it has been shown that nested expressions can be translated to disjunctive logic programs [52] and that aggregates can be translated to normal logic programs [53]. Next to these general extensions to ASP, the translation of other frameworks to ASP has also been studied. For example DLV supports abduction with penalization [54] through its frontend by compiling this framework to a logic program with weak constraints [5]. For preferences in ASP a common implementation method is to use a meta-formalism that can be compiled into

a generate program that finds answer sets and a check program that checks whether the generated answer sets are optimal [3, 4]. Though the preference extensions have a higher complexity, this method ensures that programs with preferences can still be solved using off the shelf ASP solvers such as Smodels [58] and DLV [34].

The combination of answer set programming and logic programming with uncertainty theories has also received a lot of attention. Among others, there have been extensions of logic programming using probabilistic reasoning [9, 17, 37, 38, 47, 48, 61], possibilistic reasoning [1, 49, 50], fuzzy reasoning [6, 20, 39, 40, 41, 42, 43, 55, 61, 64, 65, 66], and more general many-valued or uncertainty reasoning [7, 8, 10, 11, 12, 15, 16, 26, 27, 28, 29, 31, 32, 33, 35, 36, 46, 57, 59, 60, 62, 63]. Roughly, one can divide these approaches in two classes, viz. *implication-based* (IB) and *annotation-based* (AB) frameworks [60]. In the implication-based setting a rule is generally of the form

$$a \overset{w}{\leftarrow} \alpha$$

where a is an atom, α is a body expression[6], and $w \in \mathcal{L}$, with $\mathcal{L}$ the lattice used for truth values. Intuitively, such a rule denotes that in any model of the program the truth degree of the implication $\alpha \rightarrow a$ must be greater than or equal to the weight w. In the annotation-based approaches one considers *annotations*, which are either constants from the truth lattice $\mathcal{L}$, variables ranging over this truth lattice, or functions over elements of this truth lattice applied to annotations. A rule is then of the form

$$a : \mu \leftarrow b_1 : \mu_1, \ldots, b_n : \mu_n$$

where $a, a_1, \ldots, a_n$ are atoms and $\mu, \mu_1, \ldots, \mu_n$ are annotations. Intuitively, an annotated rule denotes that if the certainty of each b_i is at least μ_i, then the certainty of a is at least μ. The links between these two approaches are well-studied in e.g. [12, 27, 32, 33]. In this section, we give an overview of these related approaches and study their links with CFASP.

9.1. Fuzzy and Many-Valued Logic Programming Without Partial Rule Satisfaction

Many proposals for fuzzy and many-valued logic programming have rules that need to be completely fulfilled. In this category one finds most annotation-based approaches, e.g. [6, 26, 27, 55, 60, 63] and some implication-based approaches where the weight of each rule is 1, e.g. [10, 11, 12, 29]. Some of these proposals only contain monotonic functions in rules (e.g. the AB approach from [6, 27] and the IB approaches from [11, 12, 29]), while others feature negation (e.g. the AB approaches from [26, 55, 60, 63]) or even arbitrary decreasing functions (e.g. the IB approach from [10, 59]). Since the semantics of the IB approaches correspond to the semantics of CFASP the results presented in this paper show that the approaches with arbitrary decreasing functions can actually be simulated in the approaches with negation. Stronger even, they can also simulate constraints, AFASP, S-implicators and classical negation.

This furthermore means that CFASP also inherits the modelling power that is already present in some of these proposals. For example, from the embeddings shown in [9, 12], and using the fact that we can embed [10] in our approach using the simulation of decreasing functions, we inherit the capacity to translate Generalized Annotated Logic Programs [27], Probabilistic Deductive Databases [30], Possibilistic Logic Programming [14], Hybrid Probabilistic Logic Programs [13][7] and Fuzzy Logic Programming [65] to CFASP.

9.2. Weighted Rule Satisfaction Approaches

Some IB approaches to fuzzy and many-valued logic programming feature partial rule fulfillment by adding "weights" to rules (e.g. [7, 8, 15, 32, 33, 35, 36, 39, 40, 42, 43, 44, 45, 57]). These weights are specified manually and they reflect the minimum degree of fulfillment required for a rule. Formally, such a rule takes on the form of

$$a \overset{w}{\leftarrow} \alpha$$

[6] We define this in more detail further on.

[7] Note that the translation process in [9] is exponential in the size of the program, but, as the authors point out, this is to be expected as reasoning in these programs in most cases is exponential.

where a is an atom, α is a body expression (explained further on), $\leftarrow$ corresponds to a residual implicator over $[0,1]$ that is denoted as $\rightarrow_r$ and w is a value of $[0,1]$. For convenience we denote the t-norm of which $\rightarrow_r$ is the residual implicator as $\wedge_r$. Furthermore we will use r_h and r_b to refer to the head, resp. the body of a rule r as usual, and r_w to refer to the weight w. In the case of [15, 39, 40, 42, 43, 57], the bodies of rules are of the form $b_1 \wedge_1 \ldots \wedge_{n-1} b_n$, where b_i $(1 \leq i \leq n)$ is an atom and $\wedge_j$ $(1 \leq j \leq n-1)$ is a t-norm. In some cases these bodies are augmented with negation-as-failure literals, such as in [39, 40, 42, 43, 44, 45]. Other approaches, such as [7, 8, 32, 33, 35, 36], have rule bodies of the form $f(b_1, \ldots, b_n)$, where b_i $(1 \leq i \leq n)$ are atoms and f is a monotonically increasing $\mathcal{L} \rightarrow \mathcal{L}$ function. Some of these approaches feature negation-as-failure under the well-founded semantics [35, 36]. Furthermore, [7] allows a combination of multiple lattices to be used for rules. This last feature is obtained in CFASP by using the cartesian product of all these lattices as the lattice for program rules and using the corresponding projections to extend the operators used to this product lattice.

The semantics of a program consisting of weighted rules without negation-as-failure is defined in two equivalent ways in the literature. We will take [40] and [7] as examples of these two methods, but the following discussion equally applies to all the approaches mentioned earlier, barring some minor syntactical issues. In the case of [40], an interpretation M is called a **model** of a program P when for all r in P it holds that $M(r_h) \geq M(r_b \wedge_r r_w)$. **Answer sets** of these programs are then defined as minimal models. In [7], answer sets are defined as the least fixpoints of an **immediate consequence operator**, defined for a program P, interpretation I of P and atom $l \in \mathcal{B}_P$, as:

$$\Pi_P(I)(l) = \sup\{I(r_b \wedge_r r_w) \mid r \in P_l\}$$

It can be shown that these two types of semantics coincide (see e.g. [39]). We can simulate programs with weighted rules in CFASP as well. For example, consider the following program P:

$$a \xleftarrow{0.7}_m 0.9$$
$$b \xleftarrow{0.5}_m a$$

where rules with $\leftarrow_m$ are associated with the Gödel implicator. The (unique) answer set of P is $A = \{a^{0.7}, b^{0.5}\}$. The corresponding CFASP program is P':

$$\begin{array}{lrcl} r_1: & a & \leftarrow & 0.9 \wedge_m 0.7 \\ r_2: & b & \leftarrow & a \wedge_m 0.5 \end{array}$$

One can easily verify that the unique answer set of P' is also A. In general we can just replace a weighted rule of the form $a \xleftarrow{w} \alpha$ with a CFASP rule $r : a \leftarrow \alpha \wedge_r w$. For a formal proof we refer to [22].

Last, for programs with weighted rules that have negation-as-failure and monotonically increasing functions the semantics are defined using a Gelfond-Lifschitz reduct that is similar to CFASP. Hence, the translation above also works for programs with negation-as-failure, which moreover means that these languages have the same expressive power as CFASP.

9.3. Aggregated Fuzzy Answer Set Programming

In [64], an aggregated fuzzy answer set programming framework is introduced. It differs from the AFASP language in Section 6 by using unfounded sets instead of fixpoints to define the semantics and by placing more restrictions on the syntax. Syntactically, in [64], a rule over a lattice $\mathcal{L}$ is of the form $r : a \leftarrow b_1 \wedge \ldots \wedge b_n \wedge \sim c_1 \wedge \ldots \wedge \sim c_m$, where a, b_i $(1 \leq i \leq n)$, c_j $(1 \leq j \leq m)$ are atoms, $\wedge$ is a t-norm on $\mathcal{L}$ and $\sim$is a negator on $\mathcal{L}$. A program is a collection of rules with the same t-norm and negator. Hence, in a single program only a single t-norm and negator can be used.

Semantically, in [64] an answer set M has a degree k, which, as in Section 6, reflects the value of an *aggregator* function that combines the degree of satisfaction of the rules in the program into a value in the lattice underlying the program. Furthermore, an answer set is defined in [64] as a model that is free from *unfounded sets*. Intuitively, the concept of unfounded set provides a direct formalization of "badly motivated" conclusions.

Formally, a set X of atoms is called *unfounded* w.r.t. an interpretation I of a program P iff for all $a \in X$, every rule $r : a \leftarrow \alpha \in P$ satisfies either

(i) $X \cap \alpha \neq \emptyset$, or

(ii) $I_s(a, \rho_I(r)) < I(a)$, or

(iii) $I(r_b) = 0$

Intuitively, condition (i) above describes a circular motivation while (ii) asserts that a is overvalued w.r.t. r. Condition (iii) is needed to ensure that the semantics are a proper generalization of the classical semantics. Answer sets to a degree k are then defined in [64] as k-models that are free from unfounded sets, i.e. a k-model M is a k-answer set iff $supp(M) \cap X = \emptyset$ for any unfounded set X, where we recall that $supp(M) = \{a \mid a \in \mathcal{B}_P, M(a) > 0\}$. A nice feature is that this single definition covers any program P, regardless of whether it is positive, or has constraint rules. The downside is that generalizing this definition to programs with arbitrary monotonic functions is non-trivial.

In [24] it is shown that, when a total ordering is used in the lattice, the semantics of [64] correspond to the semantics that we use for AFASP in Section 6. Hence, due to the syntactical differences CFASP is more expressive than the framework introduced in [64]. When the ordering used is not total, however, this equivalence is no longer valid. For example, consider the lattice $(\mathbb{B} \times \mathbb{B}, \leq)$ such that $(1, 1)$ is the top element of the lattice, $(0, 0)$ is the bottom element and $(0, 0) \leq (0, 1) \leq (1, 1)$ and $(0, 0) \leq (1, 0) \leq (1, 1)$. Now consider an AFASP program P, with $\mathcal{A}_P(\rho) = \inf\{\rho(r) \mid r \in P\}$, over this lattice:

$$r_1 : a \leftarrow (1, 0)$$
$$r_2 : a \leftarrow (0, 1)$$

According to the property of residual implicators that $x \rightarrow y = 1_{\mathcal{L}}$ iff $x \leq y$, any interpretation I of P that satisfies rule r_1 to the degree $(1, 1)$ must obey $I(a) \geq (1, 0)$. Likewise any interpretation I that satisfies r_2 to the degree $(1, 1)$ must obey $I(a) \geq (0, 1)$. Hence the only 1-model of P is $I = \{a^{(1,1)}\}$. However, according to rule (ii) above $\{a\}$ is an unfounded set, which means that under the unfounded semantics I is not an answer set of P. On the other hand, $I = \Pi^*_{P^I, \rho_I}$, and thus I is an answer set of P according to the fixpoint semantics. If we consider rules as constraints that need to be fulfilled, the fixpoint semantics correspond better to our intuition.

10. Concluding Remarks

In this paper we have introduced a core language for FASP, motivated by the usefulness of the basic core languages that exist for ASP. Furthermore, we have shown that this core language is sufficiently expressive to cover many common extensions to FASP that have been suggested in the literature. Specifically we have proven that constraints, monotonically decreasing functions, aggregators, S-implicators and strong negation can be expressed by this core language. We have also shown that in the case of constraints and strong negation these simulations bear a great resemblance to the simulations that exist for classical ASP, which provides additional insights into the connection between classical ASP and FASP.

Our analysis is important both from a theoretical and practical point of view. From the theoretical perspective, our core language makes reasoning over FASP simpler, while our simulation results show that the theoretical results are still strong enough to cover a whole range of FASP programs. From the practical perspective our results show that to implement a solver we only need to implement one for the core language. The extensions can then be added in the front end of the solver, rather than needing to cope with it in the back end. For programs that only have t-norms in rule bodies we can use the results in [21] to implement this core language. For programs that only have linear functions in rule bodies we can use the results from [56].

Last, the fact that many extensions of FASP can be compiled to this reduced core does not mean that these extensions are useless: many of the simulations provided in this paper are cumbersome. To make FASP an intuitive language that is easy to model in, these extensions are thus of crucial importance.

Acknowledgment

We would like to thank the anonymous reviewers for their useful suggestions and remarks. Jeroen Janssen is funded by a research project from the Flemish Fund for Scientic Research (FWO-Vlaanderen). Steven Schockaert is a post-doctoral fellow of the Flemish Fund for Scientic Research (FWO-Vlaanderen).

Appendix A. Proofs

Appendix A.1. Proofs of Section 4.1

Proposition 1. *Let P be a CFASP$^{\perp}$ program over $([0,1],\leq)$ and $a \in \mathcal{B}_P$. Then if M' is an answer set of the interval-constrained version of P w.r.t. a and the interval $[l,u]$ (with l and u in $[0,1]$), called P', it holds that $M'(a) \in [l,u]$.*

Proof of Proposition 1. Since M' is a model of P', we know that $M'(upp_a) \geq 1$ and $M'(low_a) \geq 1$. Hence, $M'(upp_a) = 1$ and $M'(low_a) = 1$. From the residuation principle we then obtain both that $u \geq M'(a)$ and $M'(a) \geq l$, hence $M'(a) \in [l,u]$. □

Appendix A.2. Proofs of Section 4.2

First, we introduce a number of technical lemmas for the proof of Proposition 2.

Lemma 1. *Let P be a CFASP$^{\perp}$ program and let P' be the corresponding CFASP program as defined by Definition 9. If M is an answer set of P and $M' = M \cup \{c_k^k \mid k \in \mathcal{K}_P\} \cup \{\perp^0\}$, for any interpretation I of P' it holds that*

$$(\Pi_{P'^{M'}}(I)) \cap \mathcal{B}_P = \Pi_{P^M}(I \cap \mathcal{B}_P)$$

Proof. First, suppose $a \notin \mathcal{B}_P$. From this assumption we know that $((\Pi_{P'^{M'}}(I)) \cap \mathcal{B}_P)(a) = 0$. Furthermore, by definition of Π_{P^M} we know that $(\Pi_{P^M}(I \cap \mathcal{B}_P))(a) = \sup\{(I \cap \mathcal{B}_P)(r_b) \mid r \in P_a^M\}$. Since $P_a^M = \emptyset$ due to $a \notin \mathcal{B}_P$ we thus must conclude that $(\Pi_{P^M}(I \cap \mathcal{B}_P))(a) = 0$, and thus $((\Pi_{P'^{M'}}(I)) \cap \mathcal{B}_P)(a) = (\Pi_{P^M}(I \cap \mathcal{B}_P))(a)$.

Now, suppose $a \in \mathcal{B}_P$. By definition of $\Pi_{P'^{M'}}$ we have

$$(\Pi_{P'^{M'}}(I) \cap \mathcal{B}_P)(a) = \sup\{I(r'_b) \mid r' \in {P'}_a^{M'}\}$$

By definition of the reduct of a program we obtain

$$(\Pi_{P'^{M'}}(I) \cap \mathcal{B}_P)(a) = \sup\{I((r'_b)^{M'}) \mid r' \in P'_a\} \tag{A.1}$$

Any $r \in P_a$ is by definition a non-constraint rule and thus is mapped to $r' \in P'_a$ such that $r = r'$. Furthermore the only rules in P'_a are rules that correspond to some $r \in P_a$, leading to the conclusion that $P_a = P'_a$. Since $M = M' \cap \mathcal{B}_P$ we can see that for an expression α built from the literals in $\mathcal{B}_P$ we have both $\alpha^M = \alpha^{M'}$ and $M(\alpha^M) = M'(\alpha^{M'})$. Hence, combining this with the foregoing we can see that for each $r \in P_a$ and corresponding $r' \in P'_a$ we have

$$I(r_b^M) = I((r'_b)^{M'}) \tag{A.2}$$

Combining (A.1) with (A.2) we obtain

$$(\Pi_{P'^{M'}}(I) \cap \mathcal{B}_P)(a) = \sup\{I(r_b^M) \mid r \in P_a\}$$

By the definition of the reduct of a program this is equivalent to

$$(\Pi_{P'^{M'}}(I) \cap \mathcal{B}_P)(a) = \sup\{I(r_b) \mid r \in P_a^M\}$$

As the atoms occurring in r_b of any $r \in P_a^M$ are all atoms of $\mathcal{B}_P$ this means that

$$(\Pi_{P'^{M'}}(I) \cap \mathcal{B}_P)(a) = \sup\{I \cap \mathcal{B}_P(r_b) \mid r \in P_a^M\}$$

By the definition of Π_{P^M} it then follows that

$$(\Pi_{P'^{M'}}(I) \cap \mathcal{B}_P)(a) = (\Pi_{P^M}(I \cap \mathcal{B}_P))(a)$$

□

Lemma 2. *Let P be a CFASP$^\perp$ program. If M is a model of P, for any rule $r \in P_k$ (with $k \in \mathcal{K}_P$ and $\mathcal{K}_P$ as defined by Definition 9) it holds that*

$$M(r_b) \leq k$$

Proof. This follows easily by the observation that $M(r_b) \rightarrow_r k \geq 1$ as M is a model of P and the fact that $M(r_b) \rightarrow_r k \geq 1$ can only be the case when $M(r_b) \leq k$ as $\rightarrow_r$ is a residual implicator. □

Lemma 3. *Let P be a CFASP$^\perp$ program and let P' be the corresponding CFASP program as defined by Definition 9. If M is a model of P and $M' = M \cup \{c_k^k \mid k \in \mathcal{K}_P\} \cup \{\perp^0\}$, then for any interpretation I of P' such that $I \subseteq M'$ for all $k \in \mathcal{K}_P$ it holds that:*

$$\Pi_{P'^{M'}}(I)(c_k) = k$$

Proof. Suppose $k \in \mathcal{K}_P$. First we show that for all $r' \in ({P'}^{M'}_{c_k})$ we have $I(r'_b) \leq k$. By combining this with the definition of $\Pi_{P'^{M'}}$ we then obtain the stated. Suppose $r' \in ({P'}^{M'}_{c_k})$. The case for $r'_b = k$ is trivial, so assume $r'_b \neq k$. Since $I \subseteq M'$ we know from the fact that all functions in ${P'}^{M'}$ are monotonic that

$$I({r'_b}^{M'}) \leq M'({r'_b}^{M'})$$

Since $M(\alpha^M) = M(\alpha)$ for any rule body α this is equivalent to

$$I({r'_b}^{M'}) \leq M'(r'_b)$$

As $r' \in ({P'}^{M'}_{c_k})$ by definition of P' there must be a corresponding $r \in \mathcal{C}_P$ such that $r_h = k$ and $r_b = r'_b$. As $M = M' \cap \mathcal{B}_P$ and all atoms occurring in r'_b are atoms in $\mathcal{B}_P$ we thus obtain

$$I({r'_b}^{M'}) \leq M(r_b)$$

Using Lemma 2 on rule r, which is a constraint rule, we obtain

$$I({r'_b}^{M'}) \leq k$$

Now from the definition of $\Pi_{P'^{M'}}(I)(c_k)$ we obtain

$$\Pi_{P'^{M'}}(I)(c_k) = \sup\{I(r_b) \mid r \in {P'}^{M'}_{c_k}\}$$

From the former and the definition of r_k we know that k is the supremum of $\{I(r_b) \mid r \in {P'}^{M'}_{c_k}\}$ and thus

$$\Pi_{P'^{M'}}(I)(c_k) = k$$

□

Lemma 4. *Let P be a CFASP$^\perp$ program and let P' be the corresponding CFASP program as defined by Definition 9. If M is an answer set of P and $M' = M \cup \{c_k^k \mid k \in \mathcal{K}_P\} \cup \{\perp^0\}$, then M' is a fixpoint of $\Pi_{P'^{M'}}$.*

Proof. Observe that $\mathcal{B}_{P'}$ can be partitioned in $\mathcal{B}_P \cup \{c_k \mid k \in \mathcal{K}_P\} \cup \{\bot\}$. We will consider these partitions separately. First, for a certain $k \in \mathcal{K}_P$ we know by Lemma 3 that $\Pi_{P'^{M'}}(M')(c_k) = k = M'(c_k)$. As $M'(\bot) = 0$, it also easily follows by definition of $P'^{M'}$ that $\Pi_{P'^{M'}}(M')(\bot) = 0 = M'(\bot)$. For $\mathcal{B}_P$, by definition $M = M' \cap \mathcal{B}_P$ holds; hence by Lemma 1, the definition of M' and the fact that M is a fixpoint we obtain

$$\begin{aligned}\Pi_{P'^{M'}}(M') \cap \mathcal{B}_P &= \Pi_{P^M}(M' \cap \mathcal{B}_P) \\ &= \Pi_{P^M}(M) \\ &= M \\ &= M' \cap \mathcal{B}_P\end{aligned}$$

Hence $\Pi_{P'^{M'}}(a) = M'(a)$ for all $a \in \mathcal{B}_P$. □

Proposition 2. *Let P be a CFASP$^\bot$ program and let P' be its corresponding CFASP program as defined by Definition 9. If M is an answer set of P, then $M' = M \cup \{c_k^k \mid k \in \mathcal{K}_P\} \cup \{\bot^0\}$ is an answer set of P', where $\mathcal{K}_P$ is defined as in Definition 9.*

Proof of Proposition 2. We have to show that M' is the least fixpoint of $\Pi_{P'}$. First, suppose that $M = \emptyset$, then it is easy to see that $M' = M \cup \{c_k^k \mid k \in \mathcal{K}_P\} \cup \{\bot^0\}$ is the least fixpoint of $\Pi_{P'^{M'}}$ and thus an answer set of P'. Second, for $M \neq \emptyset$, let $\langle J_i \mid i \text{ an ordinal}\rangle$ and $\langle J_i' \mid i \text{ an ordinal}\rangle$ be the sequence corresponding to the computation of the least fixpoint of Π_{P^M}, resp. $\Pi_{P'^{M'}}$. We show that

$$J_i' = J_i \cup \{c_k^k \mid k \in \mathcal{K}_P\} \cup \{\bot^0\} \tag{A.3}$$

for any ordinal $i > 0$ from which the proposition readily follows by transfinite induction.

First note that for any ordinal i it must hold that $J_i' \subseteq M'$ due to $\Pi^*_{P'^{M'}} \subseteq M'$ by Lemma 4 and the fact that $J_i' \subseteq \Pi^*_{P'^{M'}}$. If i is a successor ordinal (including 1), then (A.3) follows from Lemmas 3, 1 and the observation that $I((r_k')_b) = 0$ for any I with $I(c_k) = k$.

If i is a limit ordinal then, by definition, $J_i' = \bigcup_{j<i} J_j'$ which, using the induction hypothesis, yields $J_i' = \bigcup_{j<i} J_j \cup \{c_k^k \mid k \in \mathcal{K}_P\} \cup \{\bot^0\}$ from which (A.3) follows immediately. □

To prove Proposition 3, we first introduce the following lemma.

Lemma 5. *Let P be a CFASP$^\bot$ program and let P' be the corresponding CFASP program as defined by Definition 9. If M' is an answer set of P', then*

$$(\forall k \in \mathcal{K}_P \colon M'(c_k) = k) \wedge M'(\bot) = 0$$

with $\mathcal{K}_P$ as defined by Definition 9.

Proof. For any c_k, with $k \in \mathcal{K}_P$, it must hold that $M'(c_k) \geq k$ as $M'(r_k) = 1$ due to M' being a model of P'. Likewise $M'(r_k') = 1$. However, $M'(c_k) > k$ for such a k is impossible since the definition of P' would imply that

$$\begin{aligned}M'(\bot) &= \Pi_{P'^{M'}}(M')(\bot) \\ &= \sup\{M'((r_{k'}')_b) \mid k' \in \mathcal{K}_P\}\end{aligned}$$

Now for any $k' \in \mathcal{K}_P$ such that $\neg(M'(c_{k'}) > k')$ we have by definition of $r_{k'}'$ that $M'((r_{k'}')_b) = 0$. For the earlier chosen k we obtain that $M'((r_k')_b) = (\sim M'(\bot) > 0)$. It is easy to see that $M'(\bot) = (\sim M'(\bot) > 0)$ has no solution however, contradicting that M' is an answer set if such a k exists. This means $M'(c_k) = k$ for any $k \in \mathcal{K}_P$ and thus also

$$M'(\bot) = M'((\sim\bot > 0) \wedge (c_k > k)) = 0$$

completing the proof. □

Proposition 3. *Let P be a CFASP$^{\perp}$ program and let P' be its corresponding CFASP program as defined by Definition 9. If M' is an answer set of P', then $M = M' \cap \mathcal{B}_P$ is an answer set of P.*

Proof of Proposition 3. Using Definition 9 and Lemma 5 it is easy to verify that ${P'}^{M'}$ contains the following rules, where $M = M' \cap \mathcal{B}_P$:

$$\begin{aligned} {P'}^{M'} \quad = \quad & \{r' : a \leftarrow \alpha \mid (r : a \leftarrow \alpha) \in (P \setminus \mathcal{C}_P)^M\} \\ & \cup \{r' : c_k \leftarrow \alpha \mid (r : k \leftarrow \alpha) \in {\mathcal{C}_P}^M\} \\ & \cup \{r_k : c_k \leftarrow k \mid k \in \mathcal{K}_P\} \\ & \cup \{r'_k : \perp \leftarrow (1 > 0) \wedge (c_k > k) \mid k \in \mathcal{K}_P\} \end{aligned}$$

By construction of P' and M we know that for each $r \in P$ we have $M(r) = M'(r')$. Thus M is a model of P.

Second, M is clearly a fixpoint of Π_{P^M} as only the rules from $\{r' : a \leftarrow \alpha \mid (r : a \leftarrow \alpha) \in (P \setminus \mathcal{C}_P)^M\}$, which are identical to those in $(P \setminus \mathcal{C}_P)^M$, are used in computing $\Pi_{P'^{M'}}(M) = M$. Hence $\Pi^*_{P^M} \subseteq M$. Suppose now that $\Pi^*_{P^M} \subset M$, then for $M'' = \Pi^*_{P^M} \cup \{c_k^k \mid k \in \mathcal{K}_P\} \cup \{\perp^0\}$ it must hold that $M'' \subset M'$ due to Lemma 5. Using Lemma 3 we know that $\Pi_{P'^{M'}}(M'')(c_k) = k = M''(c_k)$; furthermore we can easily see that $\Pi_{P'^{M'}}(M'')(\perp) = 0 = M''(\perp)$ and for $a \in \mathcal{B}_P$ that $\Pi_{P'^{M'}}(M'')(a) = M''(a)$ from the definition of P' and M''. This means that M'' is a fixpoint of $\Pi_{P'^{M'}}$, which contradicts the fact that M' is the least fixpoint of $\Pi_{P'^{M'}}$. Thus $M = \Pi^*_{P^M}$, meaning M is an answer set of P. □

Appendix A.3. Proofs of Section 5

Proposition 4. *Let P be a CFASP^f program and let P' be its corresponding CFASP program as defined by Definition 13. If M is an answer set of P, then $M' = M \cup \{not_l{}^{\sim M(l)} \mid l \in \mathcal{N}_P\}$ is an answer set of P'.*

Proof of Proposition 4. We have to show that M' is the least fixpoint of $\Pi_{P'^{M'}}$. First, note that for any $l \in \mathcal{N}_P$ by definition of P' and M'

$$\Pi_{P'^{M'}}(M')(not_l) = {\sim} M'(l) = {\sim} M(l) = M'(not_l) \tag{A.4}$$

Now, for $a \in \mathcal{B}_P$ each rule $r : a \leftarrow f(b_1, \ldots, b_n; c_1, \ldots, c_m)$ in P is replaced by $r : a \leftarrow f'(b_1, \ldots, b_n, not_{c_1}, \ldots, not_{c_m})$, with f' as in Definition 13. Hence since $M'(not_l) = {\sim} M(l)$ we obtain that

$$M(f(b_1, \ldots, b_n; c_1, \ldots, c_m)^M) = M'(f'(b_1, \ldots, b_n, not_{c_1}, \ldots, not_{c_m})^{M'}) \tag{A.5}$$

and also

$$M(r) = M'(r') \tag{A.6}$$

since $\sim$is involutive, $M = M' \cap \mathcal{B}_P$ and the atoms occurring in P are all atoms of $\mathcal{B}_P$. As M is a fixpoint of Π_{P^M}, it then follows from (A.4), (A.5), (A.6) and by definition of M' that M' must be a fixpoint of $\Pi_{P'^{M'}}$ (one only has to work out the definition of $\Pi_{P'^{M'}}$ together with the formerly mentioned equations to see this).

Suppose now there is some $M'' \subset M'$ that is also a fixpoint of $\Pi_{P'^{M'}}$. By definition of P' it is easy to see that for each $l \in \mathcal{N}_P$ it then must hold that $M''(not_l) = {\sim} M'(l) = {\sim} M(l)$. From this it follows that for each $r \in P$ we obtain $(M'' \cap \mathcal{B}_P)((r_b)^M) = M''(r'_b)$ by definition of P' and thus we obtain for each $a \in \mathcal{B}_P$:

$$\begin{aligned} \Pi_{P^M}(M'' \cap \mathcal{B}_P)(a) &= \sup\{M'' \cap \mathcal{B}_P(\alpha^M) \mid r : a \leftarrow \alpha \in P\} \\ &= \sup\{M''(\alpha') \mid r' : a \leftarrow \alpha' \in {P'}^{M'}\} \\ &= \Pi_{P'^{M'}}(M'')(a) \\ &= M''(a) \end{aligned}$$

Hence $M'' \cap \mathcal{B}_P$ is a fixpoint of Π_{P^M}. However, since for each $l \in \mathcal{N}_P$ we have $M''(not_l) = M'(not_l)$ and as $M'' \subset M'$, it holds that $M'' \cap \mathcal{B}_P \subset M$. This however contradicts the fact that M is the least fixpoint of Π_{P^M}, showing no such M'' can exist and thus M' is the least fixpoint of $\Pi_{P'^{M'}}$. □

Proposition 5. *Let P be a CFASP^f program and let P' be its corresponding CFASP program as defined by Definition 13. If M' is an answer set of P', then $M = M' \cap \mathcal{B}_P$ is an answer set of P.*

Proof of Proposition 5. By definition of P' it is not hard to see that for each rule $r \in P$ we have

$$M((r_b)^M) = M'(r'_b) \tag{A.7}$$

We show that M is the least fixpoint of Π_{P^M}. From (A.7) we can easily see that M must be a fixpoint of Π_{P^M}. Now suppose some M'' (with $M'' \subset M$) is also a fixpoint of Π_{P^M}. Then we can construct $M''' = M'' \cup \{not_l^{\sim M(l)} \mid l \in \mathcal{N}_P\}$. It is not hard to see that for each rule $r \in P$:

$$M''((r_b)^M) = M'''(r'_b)$$

Hence, as M'' is a fixpoint of Π_{P^M}, we obtain that M''' is also a fixpoint of $\Pi_{P'^{M'}}$, contradicting the fact that M' is an answer set of P'. □

Appendix A.4. Proofs of Section 6

We first include some general propositions on the AFASP framework that will be useful below.

Proposition 16. *Let I be an interpretation of a rule $r : a \leftarrow \alpha$ (over the lattice $\mathcal{L}$). Then $I_s(r, w) = I(\alpha) \wedge_r w$, with $w \in \mathcal{L}$.*

Proof. See [23]. □

Proposition 17. *Let I_1 and I_2 be interpretations of a rule $r : a \leftarrow \alpha$ with α a positive expression (over the lattice $\mathcal{L}$) such that $I_1 \subseteq I_2$. Then $(I_1)_s(r, w) \leq (I_2)_s(r, w)$.*

Proof. Using Proposition 16 and the monotonicity of t-norms we can easily see that:

$$(I_1)_s(r, w) = I_1(\alpha) \wedge_r w \leq I_2(\alpha) \wedge_r w = (I_2)_s(r, w)$$

□

Proposition 18. *Let P be a simple AFASP program and ρ a rule interpretation of this program. The immediate consequence operator $\Pi_{P,\rho}$ is monotonically increasing, i.e. for every two interpretations I_1 and I_2 it holds that*

$$I_1 \subseteq I_2 \Rightarrow \Pi_{P,\rho}(I_1) \subseteq \Pi_{P,\rho}(I_2)$$

Proof. Can be easily seen by the definition of the immediate consequence operator, Proposition 16 and the monotonicity of t-norms. □

Proposition 19. *Let M be an m-answer set ($m \in \mathcal{L}_P$) of an AFASP program P. Then M is a fixpoint of Π_{P,ρ_M}.*

Proof. Suppose $a \in \mathcal{B}_P$, then using the definition of Π_{P,ρ_M}, Proposition 16, the definition of the reduct, and since M is a fixpoint of Π_{P^M,ρ_M} (by the definition of answer set), we obtain:

$$\begin{aligned}
\Pi_{P,\rho_M}(M)(a) &= \sup\{M_s(r, \rho_M(r)) \mid r \in (\mathcal{R}_P)_a\} \\
&= \sup\{M(r_b) \wedge_r \rho_M(r) \mid r \in (\mathcal{R}_P)_a\} \\
&= \sup\{M(r_b^M) \wedge_r \rho_M(r^M) \mid r \in (\mathcal{R}_P)_a\} \\
&= \sup\{M_s(r, \rho_M(r)) \mid r \in (\mathcal{R}_P)_a^M\} \\
&= M(a)
\end{aligned}$$

□

Using the above propositions we can prove the results of this section.

Proposition 6. *Let P be an AFASP program with $\mathcal{R}_P = \{r_1, \ldots, r_n\}$ and let P' be its corresponding CFASP^f program as defined by Definition 14. If M is an m-answer set of P, then $M' = M \cup \{aggr^{\mathcal{A}_P(\rho_M)}\} \cup \{not_a^{\sim M(a)} \mid a \in \mathcal{B}_P\} \cup \{r_i'^{M(r)} \mid r \in \mathcal{R}_P\}$ is an answer set of P'.*

Proof of Proposition 6. We show that M' is the least fixpoint of $\Pi_{P'^{M'}}$. First we show that it is a fixpoint of $\Pi_{P'^{M'}}$. We consider four cases:

1. For $aggr \in \mathcal{B}_{P'}$ we obtain by definition of P' and M' that
$$\Pi_{P'^{M'}}(M')(aggr) = M'(f(r'_{1i}, \ldots, r'_{ni})) = \mathcal{A}_P(\rho_M) = M'(aggr)$$
2. For $a \in \mathcal{B}_P$ and corresponding $not_a \in \mathcal{B}_{P'}$ we obtain using the definition of M' that
$$\Pi_{P'^{M'}}(M')(not_a) = M'((\sim a)^{M'}) = M'(\sim M'(a)) = M'(\sim M(a)) = M'(not_a)$$
3. For $r : a \leftarrow \alpha \in \mathcal{R}_P$ and corresponding $r'_i \in \mathcal{B}_{P'}$ we obtain using the definition of M' and the definition of P' that
$$\begin{aligned}\Pi_{P'^{M'}}(M')(r'_i) &= M'((\alpha' \rightarrow_r (\sim not_a))^{M'}) \\ &= M'(\alpha') \rightarrow_r M'(\sim not_a) \\ &= M(\alpha) \rightarrow_r \sim(\sim M(a)) \\ &= M(\alpha) \rightarrow_r M(a) \\ &= M(r) \\ &= M'(r'_i)\end{aligned}$$
4. For $a \in \mathcal{B}_P$ we obtain, using the definition of P', the fact that for any expression α and interpretation of α we have $I(\alpha^I) = I(\alpha)$, the definition of the reduct of a program and the fact that M is a fixpoint of Π_{P^M, ρ_M} that
$$\begin{aligned}\Pi_{P'^{M'}}(M')(a) &= \sup\{M'((\alpha)^{M'}) \wedge_r M'(r'_i) \mid (r : a \leftarrow \alpha) \in P_a\} \\ &= \sup\{M(\alpha^M) \wedge_r \rho_M(r) \mid (r : a \leftarrow \alpha) \in P_a\} \\ &= \sup\{M(\alpha^M) \wedge_r \rho_M(r^M) \mid (r : a \leftarrow \alpha) \in P_a\} \\ &= \sup\{M(\alpha) \wedge_r \rho_M(r) \mid (r : a \leftarrow \alpha) \in P_a^M\} \\ &= \sup\{M_s(r, \rho_M(r)) \mid (r : a \leftarrow \alpha) \in P_a^M\} \\ &= M(a) \\ &= M'(a)\end{aligned}$$

Hence we can conclude that M' is a fixpoint of $\Pi_{P'^{M'}}$. Now suppose there is an interpretation $M'' \subset M'$ of P' such that M'' is also a fixpoint of $\Pi_{P'^{M'}}$. For $a \in \mathcal{B}_P$ we then obtain by definition of P' that
$$\Pi_{P'^{M'}}(M'')(not_a) = M''((\sim a)^{M'}) = M''(\sim M'(a)) = \sim M'(a)$$
Hence $M''(not_a) = M'(not_a)$ for each $a \in \mathcal{B}_P$. For $(r : a \leftarrow \alpha) \in \mathcal{R}_P$ we obtain by definition of P', the fact that in $(\alpha')^{M'}$ there are no naf-literals and the fact that implicators are increasing in their first and decreasing in their second argument that
$$\begin{aligned}\Pi_{P'^{M'}}(M'')(r'_i) &= M''((\alpha' \rightarrow_r (\sim not_a))^{M'}) \\ &= M''(M'(\alpha') \rightarrow_r (\sim M'(not_a))) \\ &= M''(M(\alpha) \rightarrow_r (\sim(\sim(M(a))))) \\ &= M''(M(\alpha) \rightarrow_r M(a)) \\ &= M(r) = M'(r'_i)\end{aligned}$$

From this and the definition of P' it then also easily follows that $M''(aggr) = M'(aggr)$. Hence as $M'' \subset M'$, from the foregoing it follows that $M'' \cap \mathcal{B}_P \subset M$. Now for each $a \in \mathcal{B}_P$ we can show that

$$\begin{aligned}
\Pi_{P^M,\rho_M}(M'')(a) &= \sup\{M''(\alpha^M) \wedge_r \rho_M(r) \mid (r : a \leftarrow \alpha) \in \mathcal{R}_P\} \\
&= \sup\{M''(\alpha^{M'}) \wedge_r \rho_M(r) \mid (r : a \leftarrow \alpha) \in \mathcal{R}_P\} \\
&= \sup\{M''(\alpha^{M'}) \wedge_r M''(r'_i) \mid (r : a \leftarrow \alpha) \in \mathcal{R}_P\} \\
&= \sup\{(M''(\alpha^{M'}) \wedge_r M''(r'_i)) \wedge_{r'} \rho_{M'}(r') \mid (r : a \leftarrow \alpha) \in \mathcal{R}_P\} \\
&= \sup\{M''_s(r', \rho_{M'}(r')) \mid (r : a \leftarrow \alpha) \in \mathcal{R}_P\} \\
&= \Pi_{P'^{M'}}(M'')(a) \\
&= M''(a)
\end{aligned}$$

meaning $M'' \cap \mathcal{B}_P$ is a fixpoint of Π_{P^M}, contradicting the fact that M is the least fixpoint of Π_{P^M}. Hence such an M'' cannot exist and M' is the least fixpoint of $\Pi_{P'^{M'}}$. □

Proposition 7. *Let P be an AFASP program and let P' be its corresponding CFASP^f program as defined by Definition 14. If M' is an answer set of P', with $m = M'(aggr)$, then $M' \cap \mathcal{B}_P$ is an m-answer set of P.*

Proof of Proposition 7. We have to show that for $M = M' \cap \mathcal{B}_P$ we have $\mathcal{A}_P(\rho_M) \geq m$ for any $m \leq M'(aggr)$ and that M is the least fixpoint of Π_{P^M,ρ_M}. First we show that $\mathcal{A}_P(\rho_M) \geq m$ for any $m \leq M'(aggr)$. Suppose $m \in \mathcal{L}_P$ such that $m \leq M'(aggr)$. Since M' is a fixpoint of $\Pi_{P'^{M'}}$ from the definition of P' we can easily see that for any $a \in \mathcal{B}_P$ we must have $M'(not_a) = \sim M'(a) = \sim M(a)$ as there is only one rule with not_a in the head and likewise that for any $(r : a \leftarrow \alpha) \in \mathcal{R}_P$ and corresponding $r'_i \in \mathcal{B}_{P'}$ we must have $M'(r'_i) = M'(\alpha \rightarrow_r \sim not_a) = \rho_M(r)$. Furthermore it follows that $M'(aggr) = M'(f(r'_{1i}, \ldots, r'_{ni}))$ as again there is only one rule with $aggr$ in the head and thus as $M'(f(r'_{1i}, \ldots, r'_{ni})) = \mathcal{A}_P(\rho_M)$ by construction of P' that $\mathcal{A}_P(\rho_M) \geq m$ as $M'(aggr) \geq m$.

Second we show that M is the least fixpoint of Π_{P^M,ρ_M}. First we show that M is a fixpoint of Π_{P^M,ρ_M}. Suppose $a \in \mathcal{B}_P$, then from the fact that M' is a fixpoint of $\Pi_{P'^{M'}}$, the definition of M, the fact that for any expression α we have $M(\alpha^M) = M(\alpha)$ and the foregoing part of the proof we know that:

$$\begin{aligned}
M'(a) &= \Pi_{P'^{M'}}(M')(a) \\
&= \sup\{M'((\alpha \wedge_r r'_i)^{M'}) \mid (r : a \leftarrow \alpha) \in \mathcal{R}_P\} \\
&= \sup\{M'(\alpha^{M'}) \wedge_r M'(r'_i) \mid (r : a \leftarrow \alpha) \in \mathcal{R}_P\} \\
&= \sup\{M(\alpha^M) \wedge_r \rho_M(r) \mid (r : a \leftarrow \alpha) \in \mathcal{R}_P\} \\
&= \sup\{M(\alpha^M) \wedge_r \rho_M(r^M) \mid (r : a \leftarrow \alpha) \in \mathcal{R}_P\} \\
&= \sup\{M_s(r, \rho_M(r)) \mid (r : a \leftarrow \alpha) \in (\mathcal{R}_P)^M\} \\
&= \Pi_{P^M,\rho_M}(M)(a)
\end{aligned}$$

Hence M is a fixpoint of Π_{P^M,ρ_M}. Now suppose there is some $M'' \subset M$ such that M'' is also a fixpoint of Π_{P^M,ρ_M}. Consider then $M''' = M'' \cup \{aggr^{M'(aggr)}\} \cup \{not_a^{\sim M(a)} \mid a \in \mathcal{B}_P\} \cup \{r_i'^{\rho_M(r)} \mid (r : a \leftarrow \alpha) \in \mathcal{R}_P\}$. Obviously $M''' \subset M'$ by construction. We show that M''' is a fixpoint of $\Pi_{P'^{M'}}$ contradicting the assumption that M' is an answer set of P'. Since $\mathcal{B}_{P'}$ can be partitioned in the four sets $\{not_a \mid a \in \mathcal{B}_P\}$, $\{r'_i \mid (r : a \leftarrow \alpha) \in \mathcal{R}_P\}$, $\{aggr\}$ and $\mathcal{B}_P$ we consider the elements in these four partitions separately:

1. For $a \in \mathcal{B}_P$ and the corresponding $not_a \in \mathcal{B}_{P'}$ we obtain

$$\Pi_{P'^{M'}}(M''')(not_a) = M'''((\sim a)^{M'}) = M'''(\sim M'(a)) = \sim M'(a) = M'''(not_a)$$

2. For $(r : a \leftarrow \alpha) \in \mathcal{R}_P$ and the corresponding r'_i we obtain

$$\begin{aligned}
\Pi_{P'^{M'}}(M'''')(r'_i) &= M''''((\alpha' \rightarrow_r (\sim not_a))^{M'}) \\
&= M''''((M'(\alpha') \rightarrow_r (\sim(\sim(M'(a))))) \\
&= M'(\alpha') \rightarrow_r M'(a) \\
&= M'(\alpha) \rightarrow_r M'(a) \\
&= M(\alpha) \rightarrow_r M(a) \\
&= \rho_M(r) \\
&= M''''(r'_i)
\end{aligned}$$

3. For $aggr$ we obtain

$$\begin{aligned}
\Pi_{P'^{M'}}(M'''')(aggr) &= M''''(f(r'_{1i}, \ldots, r'_{ni})) \\
&= f(\rho_M(r_1), \ldots, \rho_M(r_n)) \\
&= \mathcal{A}_P(\rho_M) \\
&= M'(aggr) \\
&= M''''(aggr)
\end{aligned}$$

4. Suppose $a \in \mathcal{B}_P$. Since M'' is a fixpoint of Π_{P^M,ρ_M} and $M'' = M'''' \cap \mathcal{B}_P$ it follows that $M'''' \cap \mathcal{B}_P$ is a fixpoint of Π_{P^M,ρ_M}. From this we obtain:

$$\begin{aligned}
\Pi_{P'^{M'}}(M'''')(a) &= \sup\{M''''((\alpha \wedge_r r'_i)^{M'}) \mid r : a \leftarrow \alpha \in \mathcal{R}_P\} \\
&= \sup\{M''''(\alpha^{M'}) \wedge_r M''''(r'_i) \mid r : a \leftarrow \alpha \in \mathcal{R}_P\} \\
&= \sup\{M''''(\alpha^M) \wedge_r \rho_M(r) \mid r : a \leftarrow \alpha \in \mathcal{R}_P\} \\
&= \sup\{M''''_s(r^M, \rho_M(r)) \mid r : a \leftarrow \alpha \in \mathcal{R}_P\} \\
&= \Pi_{P^M,\rho_M}(M'''')(a) \\
&= M''''(a)
\end{aligned}$$

Hence M'''' is a fixpoint of $\Pi_{P'^{M'}}$. This is however impossible as M' is an answer set of P' and thus the least fixpoint of $\Pi_{P'^{M'}}$. Thus no such M'' can exist and M is the least fixpoint of Π_{P^M}. □

Proposition 8. *Let P be an AFASP$^\perp$ program and let P' be its corresponding AFASP program as defined by Definition 15. If M is an m-answer set of P for some $m \in]0, 1]$, it holds that $M' = M \cup \{c^k_k \mid k \in \mathcal{K}_P\} \cup \{\perp^0\}$ is an m-answer set of P', where $\mathcal{K}_P$ is as defined by Definition 15.*

Proof of Proposition 8. Suppose M is an m-answer set of P for some $m \in]0, 1]$. Then by definition of answer sets we know that $M = \Pi^*_{P^M,\rho_M}$ and $\mathcal{A}_P(\rho_M) \geq m$. By construction of M' we immediately obtain that $\mathcal{A}_{P'}(\rho_{M'}) = \mathcal{A}_P(\rho_M) \geq m$. Hence, if we show that $M' = \Pi^*_{P'^{M'},\rho_{M'}}$ the stated follows.

Since $\mathcal{A}_{P'}(\rho_{M'}) > 0$ and by construction of P' we know that

$$\rho_{M'}(r_k) = \rho_{M'}(r'_k) = 1 \tag{A.8}$$

Now, consider program C defined as $C = \{r : a \leftarrow \alpha \wedge \rho_M(r) \mid r : a \leftarrow \alpha \in \mathcal{R}_P\}$, with $\wedge$ an arbitrary t-norm. By definition of the immediate consequence operator it is easy to see that

$$\Pi_{P^M,\rho_M} = \Pi_{C^M} \tag{A.9}$$

Now consider C', the constraint free version of C obtained using the procedure outlined in Section 4. By construction of C, C' and P' and from (A.8) it is easy to see that

$$\Pi_{P'^{M'},\rho_{M'}} = \Pi_{C'^{M'}} \tag{A.10}$$

From Proposition 2 we know that if M is an answer set of C, it follows that M' is an answer set of C'. Since M is an answer set of P, it follows that $M = \Pi^*_{P^M,\rho_M}$ and thus by (A.9) that $M = \Pi^*_{C^M}$. From this it also follows that M is a model of C, and thus that M is an answer set of C. By Proposition 2 we thus obtain that M' is an answer set of C'. Using (A.10) and the fact that the former implies $M' = \Pi^*_{C'^{M'}}$ we obtain that $M' = \Pi^*_{P'^{M'},\rho_{M'}}$, from which the stated follows. □

Proposition 9. *Let P be an AFASP$^{\perp}$ program and let P' be its corresponding AFASP program as defined by Definition 15. If M' is an m-answer set of P' for some $m \in]0,1]$, it holds that $M = M' \cap \mathcal{B}_P$ is an m-answer set of P.*

Proof of Proposition 9. Is similar to Proposition 8. □

Proposition 10. *Let P be an AFASP^{f} program and let P' be its corresponding AFASP program as defined by Definition 16. If M is an m-answer set of P for some $m \in \mathcal{L}_P \setminus \{0\}$, it holds that $M' = M \cup \{not_l^{\sim M(l)} \mid l \in \mathcal{N}_P\}$ is an m-answer set of P', where $\mathcal{N}_P$ is as defined by Definition 16.*

Proof of Proposition 10. Is similar to Proposition 8. □

Proposition 11. *Let P be an AFASP^{f} program and let P' be its corresponding AFASP program as defined by Definition 16. If M' is an m-answer set of P for some $m \in \mathcal{L}_P \setminus \{0\}$, it holds that $M = M' \cap \mathcal{B}_P$ is an m-answer set of P.*

Proof of Proposition 11. Is similar to Proposition 8. □

Appendix A.5. Proofs of Section 7

Lemma 6. *Let $\mathcal{L}$ be a lattice, $\sim$ an involutive negator over $\mathcal{L}$, $\wedge$ a t-norm over $\mathcal{L}$, $\rightarrow_r$ the residual implicator of $\wedge$, $\vee$ the t-conorm defined by $x \vee y = \sim(\sim x \wedge \sim y)$ and let $\leftarrow_s$ be the S-implicator induced by $\sim$ and $\vee$. Then for any interpretation I of a rule $r : a \leftarrow_s \alpha$ and $w \in \mathcal{L}$ it holds that:*

$$I(a \leftarrow_s \alpha) \geq w \equiv I(a \leftarrow_r \sim(\sim w \leftarrow_r \alpha)) \geq 1$$

Proof. Using the definition of $\leftarrow_s$, the residuation principle and the relationships between $\sim$, $\wedge$ and $\vee$ stated in Lemma 6, we obtain:

$$\begin{aligned}
(I(a \leftarrow_s \alpha)) \geq w &\equiv (I(a) \leftarrow_s I(\alpha)) \geq w \\
&\equiv (I(a) \vee \sim I(\alpha)) \geq w \\
&\equiv (\sim(\sim I(a) \wedge I(\alpha))) \geq w \\
&\equiv (\sim I(a) \wedge I(\alpha)) \leq (\sim w) \\
&\equiv (\sim I(a)) \leq (\sim w \leftarrow_r I(\alpha)) \\
&\equiv (I(a)) \geq (\sim(\sim w \leftarrow_r I(\alpha))) \\
&\equiv (I(a) \leftarrow_r \sim(\sim w \leftarrow_r I(\alpha))) \geq 1 \\
&\equiv (I(a \leftarrow_r \sim(\sim w \leftarrow_r \alpha))) \geq 1
\end{aligned}$$

□

Proposition 12. *Let $\mathcal{L}$ be a lattice, $\sim$ an involutive negator over $\mathcal{L}$, $\wedge$ a t-norm over $\mathcal{L}$, $\vee$ the t-conorm defined by $x \vee y = \sim(\sim x \wedge \sim y)$ and let $\leftarrow_s$ be the S-implicator induced by $\sim$ and $\vee$. Then for any interpretation I of a rule $r : a \leftarrow_s \alpha$ and $w \in \mathcal{L}$ it holds that:*

$$I_s(r, w) = \sim(\sim w \leftarrow_r I(\alpha))$$

Proof of Proposition 12. Using the definition of $\leftarrow_s$, the definition of $I_s(r, w)$, the residuation principle, the relationships between $\sim$, $\wedge$ and $\vee$ stated in Proposition 12 and using Lemma 6, we obtain:

$$\begin{aligned}
I_s(r, w) &= \inf\{y \in \mathcal{L} \mid I(y \leftarrow_s \alpha) \geq w\} \\
&= \inf\{y \in \mathcal{L} \mid I(y \leftarrow_r \sim(\sim w \leftarrow_r \alpha)) \geq 1\} \\
&= \inf\{y \in \mathcal{L} \mid (y \leftarrow_r \sim(\sim w \leftarrow_r I(\alpha))) \geq 1\} \\
&= \inf\{y \in \mathcal{L} \mid y \geq \sim(\sim w \leftarrow_r I(\alpha)) \wedge 1\} \\
&= \inf\{y \in \mathcal{L} \mid y \geq \sim(\sim w \leftarrow_r I(\alpha))\} \\
&= \sim(\sim w \leftarrow_r I(\alpha))
\end{aligned}$$

□

To show Proposition 13, we introduce a few technical lemmas.

Lemma 7. *Let P be an AFASP^s program and let P' be its corresponding AFASP program as defined by Definition 17. Then if M is a fixpoint of Π_{P^M,ρ_M} of P, it holds for the interpretation $M' = M \cup \{w_{r'}^{\rho_M(r)} \mid r \in \mathcal{R}_P^s\} \cup \{not_a^{\sim M(a)} \mid a \in \mathcal{B}_P, \exists r \in \mathcal{R}_P^s \colon r_h = a\} \cup \{not_{w_{r'}}^{\sim \rho_M(r)} \mid r \in \mathcal{R}_P^s\}$ of P' that*

1. $\forall r \in \mathcal{R}_P^r \colon \rho_{M'}(r) = \rho_M(r)$
2. $\forall r \in \mathcal{R}_P^s \colon \rho_{M'}(r') = 1_\mathcal{L}$
3. $\forall r \in \mathcal{R}_P^s \colon \rho_{M'}(r'_w) = 1_\mathcal{L}$
4. $\forall r \in \mathcal{R}_P^s \colon \rho_{M'}(n_a) = 1_\mathcal{L}$
5. $\forall r \in \mathcal{R}_P^s \colon \rho_{M'}(r'_{nw_{r'}}) = 1_\mathcal{L}$
6. $\forall r \in \mathcal{R}_P^s \colon \rho_{M'}(r'_c) = \rho_M(r)$

where r', r'_w, n_a, and r'_c are defined as in Definition 17.

Proof. We consider these cases separately:

1. Trivial from the definition of P' and M'.
2. When $r : a \leftarrow_s \alpha \in \mathcal{R}_P^s$, from Proposition 12 we have for the corresponding $r' \in \mathcal{R}_{P'}$ by definition of M' that

$$\begin{aligned}
\rho_{M'}(r') &= M'(a) \leftarrow M'(\sim_i(not_{w_{r'}} \leftarrow_r \alpha)) \\
&= M(a) \leftarrow M'(\sim_i(\sim_i \rho_M(r) \leftarrow_r \alpha)) \\
&= M(a) \leftarrow \sim_i(\sim_i \rho_M(r) \leftarrow_r M'(\alpha)) \\
&= M(a) \leftarrow \sim_i(\sim_i \rho_M(r) \leftarrow_r M(\alpha)) \\
&= M(a) \leftarrow M_s(r, \rho_M(r)) \\
&= 1_\mathcal{L}
\end{aligned}$$

   The last step follows from the fact that $x \leq y \equiv x \rightarrow y = 1$ for any residual implicator $\rightarrow$ and from the fact that M is a fixpoint of Π_{P^M,ρ_M}.
3. When $r : a \leftarrow_s \alpha \in \mathcal{R}_P^s$, using the definition of M' we have for the corresponding $r'_w \in \mathcal{R}_{P'}$ that

$$\begin{aligned}
\rho_{M'}(r'_w) &= M'(w_{r'} \leftarrow (\sim_i not_a \leftarrow_s \alpha)) \\
&= M'(w_{r'}) \leftarrow (\sim_i M'(not_a) \leftarrow_s M'(\alpha)) \\
&= \rho_M(r) \leftarrow (\sim_i(\sim_i M(a)) \leftarrow_s M(\alpha)) \\
&= \rho_M(r) \leftarrow (M(a) \leftarrow_s M(\alpha)) \\
&= 1_\mathcal{L}
\end{aligned}$$

4. When $r \in \mathcal{R}_P^s$ with $r_h = a$, we have for the corresponding $n_a \in \mathcal{R}_{P'}$ that

$$\begin{aligned}\rho_{M'}(n_a) &= M'(not_a \leftarrow\!\sim_i a)\\ &= M'(not_a) \leftarrow\!\sim_i M'(a)\\ &= \sim_i M(a) \leftarrow\!\sim_i M(a)\\ &= 1_{\mathcal{L}}\end{aligned}$$

5. When $r : a \leftarrow_s \alpha \in \mathcal{R}_P^s$ we obtain for the corresponding $r'_{nw_{r'}} \in \mathcal{R}_{P'}$ that

$$\rho_{M'}(r'_{nw_{r'}}) = M'(not_{w_{r'}} \leftarrow\!\sim_i w_{r'}) = \sim_i \rho_M(r) \leftarrow\!\sim_i \rho_M(r) = 1_{\mathcal{L}}$$

6. When $r \in \mathcal{R}_P^s$ we obtain for the corresponding $r'_c \in \mathcal{R}_{P'}$ that

$$\rho_{M'}(r'_c) = M'(0_{\mathcal{L}} \leftarrow_i \sim_i w_{r'}) = \sim_i(\sim_i M'(w_{r'})) = \rho_M(r)$$

□

Lemma 8. *Let P be an AFASP^s program with M a fixpoint of Π_{P^M,ρ_M} and let $M' = M \cup \{w_{r'}^{\rho_M(r)} \mid r \in \mathcal{R}_P^s\} \cup \{not_a^{\sim M(a)} \mid a \in \mathcal{B}_P, \exists r \in \mathcal{R}_P^s \colon r_h = a\} \cup \{not_{w_{r'}}^{\sim \rho_M(r)} \mid r \in \mathcal{R}_P^s\}$ be an interpretation of its corresponding AFASP program P', as defined by Definition 17. Then any interpretation I of P and I' of P' such that $I(a) = I'(a)$ for all $a \in \mathcal{B}_P$ satisfies*

$$\begin{aligned}\Pi_{P'^{M'},\rho_{M'}}(I') = &\Pi_{P^M,\rho_M}(I) \cup \{w_{r'}^{\rho_M(r)} \mid r \in \mathcal{R}_P^s\} \cup \{not_a^{\sim_i M(a)} \mid a \in \mathcal{B}_P, \exists r \in \mathcal{R}_P^s \colon r_h = a\}\\ &\cup \{not_{w_{r'}}^{\sim_i \rho_M(r)} \mid r \in \mathcal{R}_P^s\}\end{aligned}$$

with $\sim_i$ as in Definition 17.

Proof. Note that by Definition 17 one can immediately see that $\mathcal{B}_{P'}$ consists of four partitions, i.e. $\mathcal{B}_{P'} = \mathcal{B}_P \cup \{w_{r'} \mid r \in \mathcal{R}_P^s\} \cup \{not_a \mid a \in \mathcal{B}_P, \exists r \in \mathcal{R}_P^s \colon r_h = a\} \cup \{not_{w_{r'}} \mid r \in \mathcal{R}_P^s\}$. We consider the elements of these partitions separately.

1. Suppose $r : a \leftarrow_s \alpha \in \mathcal{R}_P^s$ and consider the corresponding $w_{r'} \in \mathcal{B}_{P'}$, then by definition of $\Pi_{P'^{M'},\rho_{M'}}$ and Proposition 16 we have

$$\Pi_{P'^{M'},\rho_{M'}}(I')(w_{r'}) = \sup\{I'(r'_b) \wedge_{r'} \rho_{M'}(r') \mid r' \in P'^{M'}_{w_{r'}}\}$$

   By Definition 17 we know that $P'^{M'}_{w_{r'}} = \{(r'_w)^{M'}\}$. Furthermore, by Lemma 7 we know $\rho_{M'}(r'_w) = 1_{\mathcal{L}}$, which, together with Definition 17, leads to

$$\Pi_{P'^{M'},\rho_{M'}}(I')(w_{r'}) = I'(((\sim not_a) \leftarrow_s \alpha)^{M'})$$

   Now by definition of the reduct and M', combined with the fact that $\sim_i$ is involutive, this leads to

$$\Pi_{P'^{M'},\rho_{M'}}(I')(w_{r'}) = I'(M'(a) \leftarrow_s M'(\alpha)) = \rho_{M'}(r)$$

   Now by the construction of M' we obtain

$$\Pi_{P'^{M'},\rho_{M'}}(I')(w_{r'}) = \rho_{M'}(r) = \rho_M(r)$$

2. Suppose $r : a \leftarrow_s \alpha \in \mathcal{R}_P^s$ and consider the corresponding $not_a \in \mathcal{B}_{P'}$. As $P'^{M'}_{not_a} = \{(n_a)^{M'}\}$, we obtain by the definition of $\Pi_{P'^{M'},\rho_{M'}}$ and Proposition 16 that

$$\Pi_{P'^{M'},\rho_{M'}}(I')(not_a) = I'((r'^{M'}_{not_a})_b) \wedge_{n_a} \rho_{M'}(n_a)$$

By Lemma 7 we know that $\rho_{M'}(n_a) = 1_{\mathcal{L}}$, hence by definition of n_a we get

$$\Pi_{P'^{M'},\rho_{M'}}(I')(not_a) = I'((\sim a)^{M'})$$

By the definition of the reduct and the fact that $M(a) = M'(a)$ for $a \in \mathcal{B}_P$ this means

$$\Pi_{P'^{M'},\rho_{M'}}(I')(not_a) = \sim M(a) = M'(not_a)$$

3. Suppose $r : a \leftarrow_s \alpha \in \mathcal{R}_P^s$ and consider the corresponding $not_{w_{r'}} \in \mathcal{B}_{P'}$. This case is entirely analogous to the case for not_a.
4. Finally, for $a \in \mathcal{B}_P$ we consider two cases:
   (a) Suppose $r : a \leftarrow \alpha \in \mathcal{R}_P^r$, then it is easy to see by definition of P', Proposition 16 and Lemma 7 that
   $$I_s(r^M, \rho_M(r)) = I'_s(r^{M'}, \rho_{M'}(r)) \tag{A.11}$$
   (b) Suppose $r : a \leftarrow_s \alpha \in \mathcal{R}_P^s$ with corresponding $r' : a \leftarrow \sim(not_{w_{r'}} \leftarrow_r \alpha)$ in $\mathcal{R}_{P'}$. By definition of reduct, we obtain that $r'^{M'} : a \leftarrow \sim(M'(not_{w_{r'}}) \leftarrow_r \alpha^{M'})$. Hence by definition of M' we obtain $r'^{M'} : a \leftarrow \sim(\sim\rho_M(r) \leftarrow_r \alpha^{M'})$. Combining this with Proposition 16, Proposition 12, Lemma 7 and the fact that for $a \in \mathcal{B}_P \cap \mathcal{B}_{P'}$ it holds that $I(a) = I'(a)$, we obtain:
   $$\begin{aligned} I'_s(r'^{M'}, \rho_{M'}(r')) &= \rho_{M'}(r') \wedge_{r'} I'(\sim(\sim\rho_M(r) \leftarrow_r \alpha^{M'})) \\ &= \sim(\sim\rho_M(r) \leftarrow_r I'(\alpha^{M'})) \\ &= \sim(\sim\rho_M(r^M) \leftarrow_r I'(\alpha^{M'})) \\ &= I_s(r^M, \rho_M(r^M)) \end{aligned}$$

Combining 4a and 4b we easily obtain that

$$\Pi_{P^M,\rho_M}(I) = \Pi_{P'^{M'},\rho_{M'}}(I') \cap \mathcal{B}_P$$

Hence, by combining these cases the stated follows. □

Proposition 13. *Let P be an AFASP^s program and let P' be the corresponding AFASP program as defined by Definition 17. If M is an m-answer set ($m \in \mathcal{L}_P$ and $m > 0_{\mathcal{L}}$) of P, it holds that $M' = M \cup \{w_{r'}^{\rho_M(r)} \mid r \in \mathcal{R}_P^s\} \cup \{not_a^{\sim_i M(a)} \mid a \in \mathcal{B}_P, \exists r \in \mathcal{R}_P^s \colon r_h = a\} \cup \{not_{w_{r'}}^{\sim_i \rho_M(r)} \mid r \in \mathcal{R}_P^s\}$ is an m-answer set of P', where $\sim_i$ is as defined by Definition 17.*

Proof of Proposition 13. First, by Lemma 7 and Definition 17 it is easy to see that $\mathcal{A}_{P'}(\rho_{M'}) \geq m$.

Second, we show that M' is an answer set of P', i.e. that it is the least fixpoint of $\Pi_{P'^{M'},\rho_{M'}}$. From Lemma 8 we can readily see that M' is a fixpoint of $\Pi_{P'^{M'},\rho_{M'}}$. Suppose now there is an $M'' \subset M'$ such that M'' is also a fixpoint of $\Pi_{P'^{M'},\rho_{M'}}$. We show by contradiction that such an M'' cannot exist. First, note that due to Lemma 8 and the fact that both M' and M'' are fixpoints of $\Pi_{P'^{M'},\rho_{M'}}$ it must hold that for all $l \in \mathcal{B}_{P'} \setminus \mathcal{B}_P$ we have $M''(l) = M'(l)$. Hence by Lemma 8 this means $M'' \cap \mathcal{B}_P \subset M' \cap \mathcal{B}_P$ and thus $M'' \cap \mathcal{B}_P \subset M$. However, by Lemma 8 and the fact that M'' is a fixpoint of $\Pi_{P'^{M'},\rho_{M'}}$ we have that $M'' \cap \mathcal{B}_P$ must be a fixpoint of Π_{P^M,ρ_M}, contradicting the fact that M is the least fixpoint Π_{P^M,ρ_M} due to M being an answer set of P. □

To show Proposition 14, we introduce the following Lemma.

Lemma 9. *Suppose P is an AFASP^s program and let P' be its corresponding AFASP program as defined by Definition 17. Then for any m-answer set M' of P', with $0_{\mathcal{L}} < m$, $m \in \mathcal{L}_P$ and $M = M' \cap \mathcal{B}_P$, it holds for all $r : a \leftarrow_s \alpha \in \mathcal{R}_P^s$ that*

1. $M'(not_a) = {\sim} M'(a)$
2. $M'(not_{w_{r'}}) = {\sim} M'(w_{r'})$
3. $M'(w_{r'}) = \rho_M(r)$

Proof. Since M' is a fixpoint of $\Pi_{P'^{M'},\rho_{M'}}$ these cases follow easily from the definition of $\Pi_{P'^{M'},\rho_{M'}}$, Proposition 16, the definition of P' and the fact that $P'^{M'}_{not_a} = \{(n_a)^{M'}\}$, $P'^{M'}_{not_{w_{r'}}} = \{(r'_{nw_{r'}})^{M'}\}$ and $P'^{M'}_{w_{r'}} = \{(r'_w)^{M'}\}$. □

Proposition 14. *Let P be an AFASP^s program and let P' be the corresponding AFASP program as defined by Definition 17. If M' is an m-answer set ($m \in \mathcal{L}_{P'}$ and $m > 0_{\mathcal{L}}$) of P', it holds that $M = M' \cap \mathcal{B}_P$ is an m-answer set of P.*

Proof of Proposition 14. First we show that $\mathcal{A}_P(\rho_M) \geq m$. For a rule $r \in \mathcal{R}^r_P$, by definition of P' it holds trivially that $\rho_M(r) = \rho_{M'}(r)$. Further, for each rule $r \in \mathcal{R}^s_P$ there is a corresponding rule $r'_c : 0_{\mathcal{L}} \leftarrow_i {\sim}_i(w_{r'})$ in P'. From Lemma 9 we know that $M'(w_{r'}) = \rho_M(r)$ and thus, as ${\sim}_i x = x \rightarrow_i 0_{\mathcal{L}}$ and ${\sim}_i({\sim}_i x) = x$ that $\rho_{M'}(r'_c) = \rho_M(r)$. Hence, as $\mathcal{A}_{P'}(\rho_{M'}) \geq m$ and $m > \min(P)$, this means $\mathcal{A}_P(\rho_M) \geq m$ by definition of $\mathcal{A}_{P'}$.

Second we show that M is the least fixpoint of Π_{P^M,ρ_M}. First we show that it is a fixpoint. From the definition of Π_{P^M,ρ_M} and Proposition 16 we obtain for $a \in \mathcal{B}_P$ that

$$\Pi_{P^M,\rho_M}(M)(a) = \sup\{M_s(r^M, \rho_M) \mid (r : a \leftarrow \alpha) \in \mathcal{R}_P\}$$

We consider two cases: $(r : a \leftarrow \alpha) \in \mathcal{R}^r_P$ and $(r : a \leftarrow \alpha \in \mathcal{R}^s_P)$.

1. If $(r : a \leftarrow_r \alpha) \in \mathcal{R}^r_P$, then in P' we have an equivalent rule and thus combining this with the former remark that $\rho_M(r) = \rho_{M'}(r)$ for such rules we obtain

$$M_s(r^M, \rho_M(r)) = M'_s(r^M, \rho_{M'}(r))$$

2. If $(r : a \leftarrow_s \alpha) \in \mathcal{R}^s_P$, then we have a corresponding rule $r' : a \leftarrow {\sim}(not_{w_{r'}} \leftarrow_r \alpha)$ in $\mathcal{R}_{P'}$. We can show that $M_s(r^M, \rho_M(r)) = M'_s(r'^{M'}, \rho_{M'}(r))$ for this rule using Proposition 16, the fact that $\rho_{M'}(r') \geq 1_{\mathcal{L}}$ since $\mathcal{A}_{P'}(\rho_{M'}) > 0_{\mathcal{L}}$, the fact that $I(\beta^I) = I(\beta)$ for any interpretation I, Lemma 9, Proposition 12 and the definition of the reduct:

$$\begin{aligned}
M'_s(r'^{M'}, \rho_{M'}(r)) &= M'((\sim(not_{w_{r'}} \leftarrow_r \alpha))^{M'}) \wedge_{r'} \rho_{M'}(r') \\
&= M'(\sim(not_{w_{r'}} \leftarrow_r \alpha)) \wedge_{r'} 1_{\mathcal{L}} \\
&= {\sim}(\sim(M'(w_{r'})) \leftarrow_r M'(\alpha)) \\
&= {\sim}(\sim(\rho_M(r)) \leftarrow_r M'(\alpha)) \\
&= {\sim}(\sim(\rho_M(r)) \leftarrow_r M(\alpha)) \\
&= {\sim}(\sim(\rho_M(r)) \leftarrow_r M(\alpha^M)) \\
&= M_s(r^M, \rho_M(r))
\end{aligned}$$

From 1 and 2 we thus obtain for $a \in \mathcal{B}_P$ that

$$\begin{aligned}
\Pi_{P^M,\rho_M}(M)(a) &= \sup\{M_s(r^M, \rho_M(r)) \mid r \in P_a\} \\
&= \sup\{M'_s(r^{M'}, \rho_{M'}(r)) \mid r \in P'_a\} \\
&= \Pi_{P'^{M'},\rho_{M'}}(M')(a) \\
&= M'(a) \\
&= M(a)
\end{aligned}$$

Hence, M is a fixpoint of Π_{P^M,ρ_M}. Now, suppose that M is not the least fixpoint of Π_{P^M,ρ_M}, then some $M'' = \Pi^*_{P^M,\rho_M} \subset M$ must exist. Consider then

$$M''' = M'' \cup \{w_{r'}^{\rho_M(r)} \mid r \in \mathcal{R}_P^s\} \cup \{not_a^{\sim M(a)} \mid a \in \mathcal{B}_P, \exists r \in \mathcal{R}_P^s \colon r_h = a\} \cup \{not_{w_{r'}}^{\sim \rho_M(r)} \mid r \in \mathcal{R}_P^s\}$$

It is clear from the construction of M''' that $M''' \subset M'$. Now, using Lemma 9 we know from the construction of M'' and M''' that

$$\Pi_{P'^{M'},\rho_{M'}}(M''') = M'''$$

Hence, M''' is a fixpoint of $\Pi_{P'^{M'},\rho_{M'}}$, which contradicts the fact that M' is the least fixpoint of $\Pi_{P'^{M'},\rho_{M'}}$. □

Appendix A.6. Proofs of Section 8

Proposition 15. *Let P be an AFASP program with strong negation and let P' be its strong-negation free version. Then an interpretation I' of P' is an (x, y)-answer set of P iff the corresponding interpretation I of P is x-consistent in the sense of [64].*

Proof of Proposition 15. Obvious from the construction of P'. □

References

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

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

[3] G. Brewka. Complex preferences for answer set optimization. In *Proceedings of the Ninth International Conference on the Principles of Knowledge Representation and Reasoning (KR2004)*, pages 213–223, 2004.

[4] G. Brewka, I. Niemelä, and M. Truszczyński. Answer set optimization. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, pages 867–872, 2003.

[5] F. Buccafurri, N. Leone, and P. Rullo. Enhancing disjunctive datalog by constraints. *IEEE Transactions on Knowledge and Data Engineering*, 12(5):845–860, 2000.

[6] T. H. Cao. Annotated fuzzy logic programs. *Fuzzy Sets & Systems*, 113(2):277–298, 2000.

[7] C. V. Damásio, J. Medina, and M. Ojeda-Aciego. Sorted multi-adjoint logic programs: termination results and applications. In *Proceedings of the 9th European Conference on Logics in Artificial Intelligence (JELIA'04)*, pages 260–273, 2004.

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

[9] C. V. Damásio and L. M. Pereira. Hybrid probabilistic logic programs as residuated logic programs. In *Proceedings of the 7th European Workshop on Logics in Artificial Intelligence (JELIA'00)*, pages 57–72, 2000.

[10] C. V. Damásio and L. M. Pereira. Antitonic logic programs. In *Proceedings of the 6th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'01)*, pages 379–392, 2001.

[11] C. V. Damásio and L. M. Pereira. Monotonic and residuated logic programs. In *Proceedings of the 6th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU'01)*, pages 748–759, 2001.

[12] C. V. Damásio and L. M. Pereira. Sorted monotonic logic programs and their embeddings. In *Proceedings of Information Processing and Management of Uncertainty (IPMU04)*, pages 807–814, 2004.

[13] A. Dekhtyar and V. S. Subrahmanian. Hybrid probabilistic programs. In *Proceedings of the Fourteenth International Conference on Logic Programming (ICLP'97)*, pages 391–405, 1997.

[14] D. Dubois, J. Lang, and H. Prade. Towards possibilistic logic programming. In *Proceedings of the Eigth International Conference on Logic Programming (ICLP'91)*, pages 581–595, 1991.

[15] M. H. v. Emden. Quantitative deduction and its fixpoint theory. *Journal of Logic Programming*, 30(1):37–53, 1986.

[16] M. Fitting. Bilattices and the semantics of logic programming. *Journal of Logic Programming*, 11(2):91–116, 1991.

[17] N. Fuhr. Probabilistic datalog: implementing logical information retrieval for advanced applications. *Journal of the American Society for Information Science*, 51(2):95–110, 2000.

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

[19] P. Hájek. *Metamathematics of Fuzzy Logic (Trends in Logic)*. 2001.

[20] M. Ishizuka and N. Kanai. Prolog-elf incorporating fuzzy logic. In *Proceedings of the 9th international joint conference on Artificial intelligence (IJCAI'85)*, pages 701–703, 1985.

[21] J. Janssen, S. Heymans, D. Vermeir, and M. De Cock. Compiling fuzzy answer set programs to fuzzy propositional theories. In *Proceedings of the 24th International Conference on Logic Programming (ICLP08)*, 2008.

[22] J. Janssen, S. Schockaert, D. Vermeir, and M. De Cock. Aggregated fuzzy answer set programming. To appear in Annals of Mathematics and Artificial Intelligence.

[23] J. Janssen, S. Schockaert, D. Vermeir, and M. De Cock. Aggregated fuzzy answer set programming. Submitted.
[24] J. Janssen, S. Schockaert, D. Vermeir, and M. De Cock. Reducing fuzzy answer set programming to model finding in fuzzy logics. Submitted.
[25] J. Janssen, S. Schockaert, D. Vermeir, and M. De Cock. General fuzzy answer set programs. In *Proceedings of the 8th International Workshop on Fuzzy Logic and Applications (WILF 2009)*, pages 352–359, 2009.
[26] M. Kifer and A. Li. On the semantics of rule-based expert systems with uncertainty. In *Proceedings of the 2nd International Conference on Database Theory (ICDT'88)*, pages 102–117, 1988.
[27] M. Kifer and V. S. Subrahmanian. Theory of generalized annotated logic programming and its applications. *Journal of Logic Programming*, 12(3&4):335–367, 1992.
[28] L. V. S. Lakshmanan. An epistemic foundation for logic programming with uncertainty. In *Proceedings of the 14th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'94)*, pages 89–100, 1994.
[29] L. V. S. Lakshmanan and F. Sadri. Modeling uncertainty in deductive databases. In *Proceedings of the 5th International Conference on Database and Expert Systems Applications (DEXA'94)*, pages 724–733, 1994.
[30] L. V. S. Lakshmanan and F. Sadri. Probabilistic deductive databases. In *Proceedings of the 1994 International Symposium on Logic programming (ILPS'94)*, pages 254–268, Cambridge, MA, USA, 1994. MIT Press.
[31] L. V. S. Lakshmanan and F. Sadri. Uncertain deductive databases: a hybrid approach. *Information Systems*, 22(9):483–508, 1997.
[32] L. V. S. Lakshmanan and N. Shiri. A parametric approach to deductive databases with uncertainty. *IEEE Transactions on Knowledge and Data Engineering*, 13(4):554–570, 2001.
[33] V. S. Lakshmanan. *Towards a generalized theory of deductive databases with uncertainty.* PhD thesis, Concordia University, 1997.
[34] N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello. The dlv system for knowledge representation and reasoning. *ACM Transactions on Computational Logic*, 7(3):499–562, 2006.
[35] Y. Loyer and U. Straccia. The well-founded semantics in normal logic programs with uncertainty. In *Proceedings of the 6th International Symposium on Functional and Logic Programming (FLOPS'02)*, pages 152–166, 2002.
[36] Y. Loyer and U. Straccia. The approximate well-founded semantics for logic programs with uncertainty. In *Proceedings of the 28th International Symposium on Mathematical Foundations of Computer Science (MFCS'03)*, pages 541–550, 2003.
[37] T. Lukasiewicz. Probabilistic logic programming. In *Proceedings of the 13th European Conference on Artificial Intelligence (ECAI'98)*, pages 388–392, 1998.
[38] T. Lukasiewicz. Many-valued disjunctive logic programs with probabilistic semantics. In *Proceedings of the 5th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'99)*, pages 277–289, 1999.
[39] T. Lukasiewicz. Fuzzy description logic programs under the answer set semantics for the semantic web. In *Proceedings of the Second International Conference on Rules and Rule Markup Languages for the Semantic Web (RuleML'06)*, pages 89–96, 2006.
[40] T. Lukasiewicz and U. Straccia. Tightly integrated fuzzy description logic programs under the answer set semantics for the semantic web. In *Proceedings of the First International Conference on Web Reasoning and Rule Systems (RR'07)*, pages 289–298, 2007.
[41] T. Lukasiewicz and U. Straccia. Top-k retrieval in description logic programs under vagueness for the semantic web. In *Proceedings of the 1st international conference on Scalable Uncertainty Management (SUM'07)*, pages 16–30, 2007.
[42] N. Madrid and M. Ojeda-Aciego. Towards a fuzzy answer set semantics for residuated logic programs. In *Proceedings of the 2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT'08)*, pages 260–264, 2008.
[43] N. Madrid and M. Ojeda-Aciego. On coherence and consistence in fuzzy answer set semantics for residuated logic programs. In *Proceedings of the 8th International Workshop on Fuzzy Logic and Applications (WILF'09)*, pages 60–67, 2009.
[44] N. Madrid and M. Ojeda-Aciego. Measuring instability in normal residuated logic programs: discarding information. *Communications in Computer and Information Science*, 80:128–137, 2010.
[45] N. Madrid and M. Ojeda-Aciego. On the measure of instability in normal residuated logic programs. In *Proceedings of FUZZ-IEEE'10*, 2010.
[46] A. Nerode, J. B. Remmel, and V. S. Subrahmanian. Annotated nonmonotonic rule systems. *Theoretical Computer Science*, 171(1-2):111–146, 1997.
[47] R. Ng and V. S. Subrahmanian. A semantical framework for supporting subjective and conditional probabilities in deductive databases. *Journal of Automated Reasoning*, 10(2):191–235, 1993.
[48] R. Ng and V. S. Subrahmanian. Stable semantics for probabilistic deductive databases. *Information and Computation*, 110(1):42–83, 1994.
[49] P. Nicolas, L. Garcia, and I. Stéphan. Possibilistic stable models. In *Nonmonotonic Reasoning, Answer Set Programming and Constraints*, Dagstuhl Seminar Proceedings, 2005.
[50] P. Nicolas, L. Garcia, I. Stéphan, and C. Lefèvre. Possibilistic uncertainty handling for answer set programming. *Annals of Mathematics and Artificial Intelligence*, 47(1-2):139–181, 2006.
[51] I. Niemelä. WASP WP3 report: Language extensions and software engineering for ASP.
[52] D. Pearce, V. Sarsakov, T. Schaub, H. Tompits, and S. Woltran. A polynomial translation of logic programs with nested expressions into disjunctive logic programs: Preliminary report. In *Proceedings of the 18th International Conference on Logic Programming (ICLP'02)*, pages 405–420, 2002.
[53] N. Pelov, M. Denecker, and M. Bruynooghe. Translation of aggregate programs to normal logic programs. In *Answer Set Programming: Advances in Theory and Implementation, CEUR Workshop Proceedings*, pages 29–42, 2003.

[54] S. Perri, F. Scarcello, and N. Leone. Abductive logic programs with penalization: semantics, complexity and implementation. *Theory and Practic of Logic Programming*, 5(1-2):123–159, 2005.
[55] E. Saad. Extended fuzzy logic programs with fuzzy answer set semantics. In *Proceedings of the 3rd International Conference on Scalable Uncertainty Management (SUM'09)*, pages 223–239, 2009.
[56] S. Schockaert, J. Janssen, D. Vermeir, and M. De Cock. Answer sets in a fuzzy equilibrium logic. In *Proceedings of Rule Reasoning 2009 (RR2009)*, pages 135–149, 2009.
[57] E. Y. Shapiro. Logic programs with uncertainties: a tool for implementing rule-based systems. In *Proceedings of the Eighth international joint conference on Artificial intelligence (IJCAI'83)*, pages 529–532, 1983.
[58] P. Simons. *Extending and Implementing the Stable Model Semantics*. PhD thesis, Helsinki University of Technology, 2000.
[59] U. Straccia. Query answering in normal logic programs under uncertainty. In *In 8th European Conferences on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU-05)*, pages 687–700, 2005.
[60] U. Straccia. Annotated answer set programming. In *Proceedings of the 11th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU'06)*, 2006.
[61] U. Straccia. Managing uncertainty and vagueness in description logics, logic programs and description logic programs. In *Reasoning Web: 4th International Summer School 2008*, pages 54–103, 2008.
[62] U. Straccia, M. Ojeda-Aciego, and C. V. Damásio. On fixed-points of multivalued functions on complete lattices and their application to generalized logic programs. *SIAM Journal on Computing*, 38(5):1881–1911, 2009.
[63] V. S. Subrahmanian. Amalgamating knowledge bases. *ACM Transactions on Database Systems*, 19(2):291–331, 1994.
[64] D. Van Nieuwenborgh, M. De Cock, and D. Vermeir. An introduction to fuzzy answer set programming. *Annals of Mathematics and Artificial Intelligence*, 50(3-4):363–388, 2007.
[65] P. Vojtás. Fuzzy logic programming. *Fuzzy Sets and Systems*, 124(3):361–370, 2001.
[66] G. Wagner. Negation in fuzzy and possibilistic logic programs. In *Uncertainty Theory in Artificial Intelligence Series*, pages 113–128, 1998.