mathematics

Article

Extending Fuzzy Linguistic Logic Programming with Negation †

Van Hung Le

Faculty of Information Technology, Hanoi University of Mining and Geology, Duc Thang, Bac Tu Liem, Hanoi 100803, Vietnam; levanhung@humg.edu.vn

† This paper is an extended version of our paper published in the 11th International Conference on Computational Collective Intelligence, ICCCI 2019, Hendaye, France, 4–6 September 2019.

Abstract: Fuzzy linguistic logic programming (FLLP) is a framework for representation and reasoning with linguistically expressed human knowledge. In this paper, we extend FLLP by allowing negative literals to appear in rule bodies, resulting in normal logic programs. We study the stable model semantics and well-founded semantics of such programs and their relation. The two kinds of semantics are adapted from those of classical ones based on the Gelfond–Lifschitz transformation and van Gelder's alternating fixpoint approach, respectively. To our knowledge, until now, there has been no work on the well-founded semantics of normal programs in any fuzzy logic programming (FLP) framework based on Vojtáš's FLP. Moreover, the relation between the two kinds of semantics is usually studied using a bilattice setting of the truth domain. However, our truth domains do not possess a complete knowledge-ordering lattice and, thus, do not have a bilattice structure. The two kinds of semantics possess properties similar to those of the classical case. Every stable model contains the well-founded (partial) model, and the well-founded total model coincides with the unique stable model, but not vice versa. Since the well-founded semantics is closely related to the stable model semantics, it can help compute stable models more efficiently.

Keywords: fuzzy logic programming; stable model semantics; well-founded semantics; linguistic truth value; hedge algebra; linguistic hedge

MSC: 68T30

Citation: Le, V.H. Extending Fuzzy Linguistic Logic Programming with Negation. *Mathematics* **2022**, *10*, 3105. https://doi.org/10.3390/math10173105

Academic Editor: Gia Sirbiladze

Received: 12 July 2022
Accepted: 23 August 2022
Published: 29 August 2022

1. Introduction

In the real world, there are situations where information might not be stated in a quantitative form, but rather in a qualitative one, e.g., by linguistic terms. This may arise for different reasons [1]. In some cases, due to its nature, the information may be unquantifiable and thus can be given only in linguistic terms; for instance, when accessing the "comfort" or "design" of a car, we might be led to use linguistic terms such as "good", "medium", and "bad" [2]. In other cases, precise quantitative information may not be given since either it is unavailable or the cost of computation is too high, so a linguistic "approximate value" might be acceptable; for example, when the rotation speed of an electric motor is accessed, linguistic terms, e.g., "very large", "large", and "medium", might be utilized instead of numerical values [3]. Furthermore, since humans primarily use words to describe phenomena and things, to reason, and to make decisions, human knowledge is usually expressed linguistically and employs fuzzy concepts and hedges. Hence, it is essential to have a knowledge representation framework that can directly work with linguistic terms and hedges to give answers to queries. Such a framework can provide a computational approach to human reasoning and can be a foundation for applications dealing with linguistic information.

FLLP is introduced in [4] as a logic programming (LP) framework without negation for representation and reasoning with linguistically expressed human knowledge, where the truth of vague sentences is stated linguistically and various linguistic hedges can be used simultaneously to express different levels of emphasis. To our knowledge, FLLP is the only FLP framework that can directly compute with linguistic terms and hedges in query answering and allow hedge connectives to appear in rule bodies to modify fuzzy predicates. Note that linguistic hedges play a very important twofold role in Zadeh's fuzzy logic, namely in the modification of fuzzy concepts and in the generation of linguistic values of linguistic variables [5].

In FLLP, each fact or rule is evaluated by some linguistic truth value, and linguistic hedges can operate as unary connectives in rule bodies. For example, the evaluation "(A student is good if he/she is *rather* smart and studies *very* hard) is *highly true*" can be represented by the rule:

$$good(X) \leftarrow \wedge(R\ smart(X), V\ hard\_working(X)).HT$$

where R, V, H, and T stand for *rather*, *very*, *highly*, and *true*, respectively.

FLLP has a counterpart for most of the notions and results of classical definite LP [4,6], e.g., declarative semantics, procedural semantics, and fixpoint semantics. In particular, the procedural semantics and fixpoint semantics [4] can give answers to queries by directly computing with linguistic terms. However, the latter is exhaustive and not goal-oriented, and the former is goal-oriented but may lead to an infinite loop. Therefore, in [7,8], two tabulation proof procedures for query answering, which are goal-oriented, sound, complete, and terminated, are provided. Of the procedures, the deterministic is more efficient than the non-deterministic and computes all and only the most general answers for a query.

It can be observed that permitting the representation and manipulation of negation is a key feature for real-world applications. In classical LP, Horn programs are extended to *normal* logic programs by allowing negative literals to occur in rule bodies for expressing explicit negative information. For example, in the real world, we usually do "jumping to conclusions" when the available information is incomplete [9]. For instance, if we know that Tweety is a bird, we jump to the conclusion that it can fly. This default assumption, i.e., that birds can fly, can be represented by the following rule:

$$fly(X) \leftarrow bird(X)$$

Later, if we have more information, it might turn out that the conclusion is no longer true and must be withdrawn. For instance, if we afterwards know that Tweety is a penguin, the conclusion is withdrawn. This use of logic is called *belief revision*. One method of belief revision in LP is using abnormality relations and negation. For the case of the penguin Tweety, we can change the rule to the one below [9]:

$$fly(X) \leftarrow bird(X) \wedge \neg abnormal_{fly,bird}(X) \tag{1}$$

and add the following rule to the program:

$$abnormal_{fly,bird}(X) \leftarrow penguin(X) \tag{2}$$

Thus, the complete, revised program consists of rules (1), (2) and the following facts:

$$\begin{aligned} bird(tweety) &\leftarrow \\ penguin(tweety) &\leftarrow \end{aligned}$$

The *well-founded semantics* [10] and the *stable model semantics* [11] are probably the most widely studied kinds of semantics for classical normal logic programs. The motivation behind the former is to extend the least model semantics of Horn programs to normal logic programs, i.e., keeping the property of the uniqueness of an intended/canonical model. Generally, this means moving beyond the setting of two-valued interpretations. In contrast, the objective behind the latter is to keep intended models two-valued. However, this means the existence and uniqueness properties of intended models could no longer be ensured.

More precisely, all normal programs have a well-founded *partial* model, which can be interpreted as a three-valued model in Fitting's sense [12], whereas a normal program can have multiple/one/no stable (total) models. Moreover, the well-founded model can be seen as an approximation of all stable models (if they exist) [13]. Every stable model contains the well-founded (partial) model. Consequently, a well-founded total model coincides with the unique stable model, but not vice versa.

In the literature of FLP, which is an extension of classical LP with many-valued logic (MVL)/fuzzy logic in the narrow sense (FLn)/mathematical fuzzy logic (MFL) for handling vagueness, there are few works addressing the issue of non-monotonic negation. Cornejo et al. [14] study the stable model semantics of multi-adjoint normal logic programs and its properties. These results are extensions of those of normal residuated logic programs in [15]. Loyer and Straccia [16] define the well-founded semantics (therein called the *compromise semantics*) for normal logic programs in a parametric implication-based framework. Nevertheless, to our knowledge, there is no work studying both kinds of semantics and their relation within an FLP framework.

Another approach to defining the semantics of normal logic programs is based on the notion of *bilattice* [17,18]. A bilattice is a structure $\langle \mathcal{B}, \preceq_t, \preceq_k \rangle$, where $\preceq_t$ and $\preceq_k$ are the *truth order* and *knowledge order* on $\mathcal{B}$, respectively. The extension of the stable model semantics and the well-founded semantics of classical normal logic programs to the context of bilattices is due to Fitting [19,20]. In this approach, Loyer and Straccia [21,22] propose the *approximate well-founded semantics* for normal logic programs in an FLP framework based on the notion of an *approximate interpretation*, which assigns to an atom A a logical (*interval*) value $[a; b]$, meaning that the exact truth value of A is between a and b.

In this paper, we extend FLLP by permitting negative literals to appear in rule bodies, resulting in normal logic programs. We study both the stable model semantics and the well-founded semantics of such programs and their relation in terms of fixpoints of several operators. It is shown that the two kinds of semantics possess properties similar to those of the classical case. Therefore, the study conducted in this work makes the following contributions: (1) enriching the theory of FLLP with negation to extend its applicability in real-world applications; (2) studying the two most widely studied kinds of semantics of normal programs, namely the stable model semantics and the well-founded semantics (note that this is the first time the well-founded semantics is proposed for an FLP framework among those based on Vojtáš's FLP [23]); and (3) studying the relation between the two kinds of semantics, which can help compute stable models more efficiently.

The remainder of the paper is organized as follows: Section 2 provides a literature review, while Section 3 gives an overview of FLLP without negation. Section 4 presents FLLP with negation, the stable model semantics and the well-founded semantics of normal logic programs, and their relation. Section 5 concludes the paper and gives several directions for future work.

2. Literature Review

2.1. Managing Vagueness in Logic Programming

In order to handle vagueness, classical LP is often extended with MVL/FLn/MFL. There have been many frameworks (see, for instance, [24]) proposed in this approach. In the *implication-based* (IB) approach [25], a truth value, which can be called the *degree of confidence* or the *weight*, is attached to each rule. The underlying logic is often *truth-functional*; that is, the truth value of a compound formula is computed using truth functions of connectives in the formula. Most of the works deal with logic programs without negation. More precisely, a rule is of the form:

$$A \leftarrow @(B_1, \ldots, B_n).q \tag{3}$$

where $A, B_1, \ldots, B_n, n \geq 0$, are atoms, @ is an operator aggregating atoms, and q is the truth degree of the rule. Given a fuzzy interpretation I of B_is by truth values in a truth domain, the truth value of A is computed by evaluating the truth value of the rule body using the truth function $@^{\bullet}$ and then somehow "propagating" it to the rule head.

One of the most well-known frameworks in the IB approach is Vojtáš's FLP [23]. The truth domain is the unit interval [0,1] with its usual ordering. A fuzzy logic program is a finite set of fuzzy rules of the form (3) and facts, which are graded atoms of the form $(A.a)$. In a rule, @ is an *aggregation* connective whose truth function is an aggregation operator. The aggregation operators can cover all kinds of fuzzy conjunctions and disjunctions. Rules can use different implication connectives $\leftarrow_i$. The truth function of an implication connective is an *implicator* which is *residual* to a *conjunctor*; i.e., they satisfy the residuation property (9). A conjunctor has the same properties as a t-norm except that commutativity and associativity are not assumed [26].

Monotonic logic programming (MLP) and residuated logic programming (RLP) [27] differ from Vojtáš's FLP in the following ways: (a) they extend the truth space to either a complete upper semilattice $\mathcal{L} = \langle \mathcal{T}, \preceq \rangle$, where the *carrier* set $\mathcal{T}$ is not necessary to be the unit interval [0,1], for the case of MLP, or a complete residuated lattice, in which there is a pair of binary operations $(\leftarrow, \otimes)$ forming an adjoint (residual) pair, for the case of RLP; (b) they follow an algebraic approach to both syntax and semantics of logic programs [28]. That is, monotonic logic programs (resp., residuated logic programs) are constructed from the abstract syntax induced by an *implication algebra*, defined with an R-implication $\leftarrow$ on $\mathcal{L}$, (resp., by an *residuated algebra*, defined with a residual pair $(\leftarrow, \otimes)$) and a set of propositional symbols; (c) the aggregation operator @ is only required to be monotonic; (d) the languages are propositional ones; and (e) there is only one implication connective used in logic programs.

Multi-adjoint logic programming (MALP) [29] is an extension of RLP in the sense that: (i) it is possible to use a number of different implication connectives in rules as in Vojtáš's FLP; (ii) the algebraic requirements on residuated lattices are weaker (more precisely, the condition that $(\mathcal{T}, \otimes, \top)$ is a commutative monoid in a residuated lattice is replaced by a weaker condition $\top \otimes q = q \otimes \top = q$, where $\top$ is the top element of $\mathcal{L}$, in a multi-adjoint lattice); and (iii) some sufficient conditions for the continuity of the immediate consequence operator T_P are established. In fact, if all operators @ in rule bodies are continuous in all arguments, and all adjoint conjunctors $\otimes$ are continuous in the rule body argument, then T_P is continuous. MALP is extended to a first-order language in [30]. In this setting, MALP is an extension of Vojtáš's FLP in terms of the truth space and becomes one of the most general FLP frameworks.

FLLP extends or modifies Vojtáš's FLP on the following aspects[4]: (a) utilizing linguistic truth domains instead of the unit interval [0,1]; (b) allowing rule bodies to use hedge connectives to modify fuzzy predicates; (c) adding an admissible rule to deal with hedge connectives in procedural semantics; (d) giving detailed proofs of the mgu lemma, the lifting lemma, and the completeness theorem for the case of general (i.e., non-ground) queries; (e) allowing rule bodies to use linguistic aggregation operators [6,31]; (f) proving counterparts of the results of classical definite LP, e.g., the model intersection property and the characterization of the least Herbrand model of a logic program by the infimum of the set of all its Herbrand models [6]; (g) proving a Pavelka-style completeness [32] for ground atoms [6]; (h) providing a sound and complete terminating goal-oriented deterministic query-answering procedure, which computes all and only the most general answers to a query [7]; and (i) in this work, extending logic programs with negation. It can be seen that the purely linguistic property and hedge connective usage are unique features of FLLP among FLP frameworks. In particular, hedge connectives have no counterpart in the other FLP frameworks. Therefore, FLLP is not a special case of any FLP framework even though our LTD is just a chain.

The frameworks in [33–35], among others, can also be classified into the IB approach.

2.2. Normal Logic Programs for Handling Vagueness

In the literature of LP dealing with vagueness, there are few works addressing the issue of *non-monotonic negation*.

Cornejo et al. [14] study the stable model semantics of multi-adjoint normal logic programs (MANLP) and prove the existence of stable models for MANLPs whose truth domain is a convex compact set of a Euclidean space and the operators in rule bodies are continuous. Moreover, they show that the uniqueness of stable models is ensured for a special kind of MANLPs on the subinterval lattice $(\mathcal{C}([0,1]), \leq)$. These results extend those of normal residuated logic programs (NRLP) [15]. The stable model semantics of NRLPs is also developed based on the Gelfond–Lifschitz transformation [11]. However, the well-founded semantics is not discussed in these works.

Loyer and Straccia [16], inspired by van Gelder's alternating fixpoint approach [36], define the well-founded semantics (therein called *compromise semantics*) for normal logic programs in a parametric IB framework. For classical Datalog programs with negation, the proposed semantics is identical to the well-founded semantics [10].

Another approach to defining the semantics of normal logic programs is based on the notion of a *bilattice* [17,18]. A bilattice is a structure $\langle \mathcal{B}, \preceq_t, \preceq_k \rangle$, where $\mathcal{B}$ is a non-empty truth set, $\preceq_t$ and $\preceq_k$ are the *truth order* and *knowledge order*, respectively. If $x \preceq_k y$ and $x \preceq_t y$, then y represents "more information" and "more truth", respectively, than x. Each bilattice has a negation $\neg$, which reverses the $\preceq_t$ ordering, leaves unchanged the $\preceq_k$, and verifies $\neg\neg x = x$. The simplest non-trivial bilattice, called $\mathcal{FOUR}$, is due to Belnap, Jr. [37], who introduced a logic intended to handle incomplete and/or inconsistent information. $\mathcal{FOUR}$ illustrates many basic properties of bilattices. In essence, $\mathcal{FOUR}$ expands the classical truth set $\{f, t\}$ to $\{f, t, \bot, \top\}$, where $\bot$ and $\top$ stand for *unknown* and *inconsistent*, respectively.

The extension of the notions of a stable model and a well-founded model to the context of bilattices is due to Fitting [19,20]. Fitting proposes a binary immediate consequence operator $\Psi_P(I, J)$, which has two input interpretations I and J over a bilattice; the former and the latter are used to assign meaning to positive literals and negative literals, respectively. It is shown that Ψ_P is monotone and, if the de Morgan laws hold, continuous in both arguments under $\preceq_k$; Ψ_P is monotone and, if the de Morgan laws hold, continuous in the first argument and anti-monotone in the second argument under $\preceq_t$. After that, fitting follows the idea of the Gelfond–Lifschitz transformation, i.e., fixing the interpretation for negative literals and then computing the least model under $\preceq_t$ of the resulting positive program. To this end, Fitting further defines the operator Ψ'_P which can be computed by iterating $\Psi_P(x, J)$ from the least interpretation under $\preceq_t$. It is shown that Ψ'_P is monotone and, if the de Morgan laws hold, continuous under $\preceq_k$ and anti-monotone under $\preceq_t$. Every fixpoint of Ψ'_P is a *stable model* of P. The set of fixpoints of Ψ'_P is a complete lattice under $\preceq_k$. Thus, Ψ'_P has a least fixpoint under $\preceq_k$, which is the *well-founded model* of P. The well-founded model can be obtained by iterating Ψ'_P from the least interpretation under $\preceq_k$. In this approach, Loyer and Straccia [21,22], within a framework extending *parametric deductive databases with uncertainty* [25] with negation, propose the *approximate well-founded semantics* for normal logic programs based on the notion of an *approximate interpretation*, assigning to an atom A a logical (*interval*) value $[a; b]$.

Answer set programming (ASP) [38–40] is a form of declarative programming oriented towards hard search problems. In addition to rules and facts, ASP programs can have *constraints*, which are rules with no atom in the head. ASP transforms the stable model semantics into a practical computational methodology and is a formalism for modeling and solving combinatorial optimization problems. Computing stable models is a fundamental automated reasoning task of *answer-set solvers* [41]. Fuzzy answer set programming (FASP) [42,43] is an expansion of ASP with MVL/FLn/MFL for modeling continuous optimization problems. The truth values associated with atoms and rules are also called their *weights*. In FASP, which is called core FASP in [44], the weight of each rule is 1. In aggregated FASP [44,45], rules can have a weight other than 1. Cornejo et al. [46] provides a bidirectional translation between core FASP programs and MANLP programs preserving their semantics. This translation allows the expressiveness of MANLP to be combined with the compactness and simplicity of core FASP. Therefore, the existence of stable models of

MANLP programs implies that of answer sets of corresponding core FASP programs. Extended MALP (EMALP) [47] is an extension of MANLP where constraints can be included in logic programs. A procedure converting EMALP programs to constraint-free EMALP ones, i.e., MANLP programs, preserving the stable model semantics is also proposed.

In LP and ASP, there is a well-known method to compute the stable model semantics of normal logic programs, called "*guess-and-check*" [9,48,49]. It consists of two steps: (1) guessing a candidate for a stable model/answer set of a normal logic program and (2) checking whether the candidate is actually a stable model/answer set of the program.

To our knowledge, there is no work on the well-founded semantics of normal logic programs of any FLP framework based on Vojtáš's FLP, although it is one of the most widely studied kinds of semantics in the literature. Therefore, in this work, we study the well-founded semantics of normal logic programs of FLLP, and it may be adapted for other FLP frameworks. We also adapt the stable model semantics of classical LP to FLLP based on the usual Gelfond–Lifschitz transformation. We then study the relation between the two kinds of semantics. In fact, in the literature, this relation is usually studied using a bilattice setting of the truth domain. However, our truth domains do not possess a complete-knowledge ordering lattice and, thus, do not have a bilattice structure. Hence, we cannot utilize that method. Since the well-founded semantics is closely related to the stable model semantics, the well-founded semantics may help reduce the number of candidates in Step (1) of the above guess-and-check method. That is, the well-founded semantics might help compute stable models more efficiently.

3. Preliminaries

3.1. Linguistic Truth Domains

In hedge algebra (HA) theory [50,51], values of the linguistic variable *Truth*, for instance, *Very True* and *Very Rather False*, can be considered as being generated from a set of atomic terms $\mathcal{G} = \{False, True\}$ using elements from a set of linguistic hedges $\mathcal{H} = \{Very, Rather, \dots\}$ as unary operators. There exists a natural ordering among those values, for example, *True* $<$ *Very True*, where $x \le y$ says that x is less than or equal to y in terms of truth. Thus, a value set $\mathcal{X}$ of *Truth* is a partially ordered set and can be characterized by an HA $\underline{X} = (\mathcal{X}, \mathcal{G}, \mathcal{H}, \le)$.

Hedges either increase or decrease the meaning of terms, and hence they can be regarded as *order operators*; i.e., for all $x \in \mathcal{X}$ and for all $h \in \mathcal{H}$, we have either $hx \ge x$ or $hx \le x$. It is denoted by $h \le k$ if h modifies terms less than or equal to k; i.e., for all $x \in \mathcal{X}$, either $kx \le hx \le x$ or $x \le hx \le kx$. Since the sets $\mathcal{H}$ and $\mathcal{X}$ are disjoint, we may use the same notation $\le$ for different order relations on $\mathcal{H}$ and $\mathcal{X}$. Let V, R, H, S, A, M, c^+, and c^- stand for *Very*, *Rather*, *Highly*, *Slightly*, *Approximately*, *More or less*, *True*, and *False*, respectively. We denote $h < k$ if $h \le k$ and $h \ne k$. We have $R < S$ since, e.g., $Sc^+ < Rc^+ < c^+$ and $c^- < Rc^- < Sc^-$, and $H < V$ since, e.g., $c^+ < Hc^+ < Vc^+$ and $Vc^- < Hc^- < c^-$.

$\mathcal{H}$ can be divided into two disjoint subsets: a set of stressing/intensifying hedges $\mathcal{H}^+ = \{h \mid hc^+ > c^+\} = \{h \mid hc^- < c^-\}$ and a set of depressing/weakening hedges $\mathcal{H}^- = \{h \mid hc^+ < c^+\} = \{h \mid hc^- > c^-\}$. For example, the set $\mathcal{H} = \{V, H, A, M, R, S\}$ is split into $\mathcal{H}^+ = \{V, H\}$ and $\mathcal{H}^- = \{A, M, R, S\}$ since, e.g., $Vc^+ > Hc^+ > c^+, c^+ < Ac^+$ and $c^+ < Mc^+$. Hedges in each of the sets $\mathcal{H}^+$ and $\mathcal{H}^-$ can be either comparable, for example, V and H, or incomparable, for instance, A and M. An HA $\underline{X} = (\mathcal{X}, \mathcal{G}, \mathcal{H}, \le)$ is said to be a *linear* HA if both sets $\mathcal{H}^+$ and $\mathcal{H}^-$ are linearly ordered. For example, the HA $\underline{X} = (\mathcal{X}, \mathcal{G}, \{V, H, R, S\}, \le)$ is a linear HA since in $\mathcal{H}^+ = \{V, H\}$, we have $V > H$, and in $\mathcal{H}^- = \{R, S\}$, we have $S > R$. For a linear HA $\underline{X} = (\mathcal{X}, \mathcal{G}, \mathcal{H}, \le)$, the truth value set $\mathcal{X}$ is linearly ordered [4]. As shown in [4], in order to adequately define the truth functions of hedge connectives, we restrict ourselves to linear linguistic truth domains.

A *linguistic truth domain* (LTD) $\overline{X}$ generated from a linear HA $\underline{X} = (\mathcal{X}, \mathcal{G}, \mathcal{H}, \le)$ is the linearly ordered set $\overline{X} = \mathcal{X} \cup \{0, W, 1\}$, where 0, W, and 1, respectively, represent *Absolutely False*, the *middle truth value*, and *Absolutely True*. They are the least, the neutral and the greatest elements of $\overline{X}$, respectively, and for all $x \in \{0, W, 1\}$ and for all $h \in \mathcal{H}$, we have $hx = x$ [4,7].

In order to adequately define Łukasiewicz operations [32], we only consider finite truth domains. An *l-limit* HA, where l is a positive integer, is a linear HA whose all values have a length of at most $l+1$, i.e., at most, l hedges can be applied to the atomic term. An LTD generated from an l-limit HA is finite [4,7].

Example 1. *The LTD* $\overline{X} = \{v_0 = 0, v_1 = Vc^-, v_2 = Hc^-, v_3 = c^-, v_4 = Rc^-, v_5 = Sc^-, v_6 = W, v_7 = Sc^+, v_8 = Rc^+, v_9 = c^+, v_{10} = Hc^+, v_{11} = Vc^+, v_{12} = 1\}$ *is generated from the 1-limit HA* $\underline{X} = (\mathcal{X}, \{c^-, c^+\}, \{V, H, R, S\}, \leq)$*, where truth values are listed in ascending order.*

3.2. Truth Functions of Hedge Connectives

Let the *identity* hedge $I \notin \mathcal{H}$ be defined by for all $x \in \mathcal{X}$, $Ix = x$. It can be seen that I is the least element in both sets $\mathcal{H}^+ \cup \{I\}$ and $\mathcal{H}^- \cup \{I\}$ [4,7].

An *extended order relation* $\leq_e$ on $\mathcal{H} \cup \{I\}$ is defined as follows: for all $h, k \in \mathcal{H} \cup \{I\}$, $h \leq_e k$ if one of the following conditions holds:

(i) $h \in \mathcal{H}^-, k \in \mathcal{H}^+$;

(ii) $h, k \in \mathcal{H}^+ \cup \{I\}$ and $h \leq k$;

(iii) $h, k \in \mathcal{H}^- \cup \{I\}$ and $h \geq k$.

It is denoted by $h <_e k$ if $h \leq_e k$ and $h \neq k$.

Let $\underline{X} = (\mathcal{X}, \{c^+, c^-\}, \mathcal{H}, \leq)$ be a linear HA. For every hedge $h \in \mathcal{H} \cup \{I\}$, its *truth function*, a mapping from $\overline{X}$ to $\overline{X}$, is denoted by $h^\bullet$. For all $h, k \in \mathcal{H} \cup \{I\}$, their truth functions satisfy the following conditions [7,52,53]:

$$\forall x, y \in \overline{X}, \text{if } x \geq y, h^\bullet(x) \geq h^\bullet(y) \tag{4}$$

$$\forall x \in \overline{X}, \text{if } h \leq_e k, h^\bullet(x) \geq k^\bullet(x) \tag{5}$$

$$\forall x \in \{0, W, 1\}, h^\bullet(x) = x \tag{6}$$

$$\forall x \in \overline{X}, I^\bullet(x) = x \tag{7}$$

$$h^\bullet(hc^+) = c^+ \tag{8}$$

Every truth function of a hedge is non-decreasing and preserves 0 and 1. Furthermore, a truth function of a stressing hedge $h \in \mathcal{H}^+$ is subdiagonal (i.e., $h^\bullet(x) \leq x$), and that of a depressing hedge $h \in \mathcal{H}^-$ is superdiagonal (i.e., $h^\bullet(x) \geq x$) [52]. According to Condition (8), if the truth value of "Lucia is *young*" is *very true*, that of "Lucia is *very young*" is *true* [54–56]. In [4], it is shown that truth functions of hedges always exist.

Example 2. *Consider the* 1*-limit HA in Example 1 and its LTD. An example of truth functions of hedges on the LTD is given in Table 1, where the value of a truth function* $h^\bullet$*, occurring in the first row, of a value* x*, occurring in the first column, is in the corresponding cell. For instance,* $H^\bullet(Vc^+) = Hc^+$ *and* $R^\bullet(Vc^-) = Hc^-$*. Below, we utilize the LTD, and these truth functions of hedges for our examples.*

Table 1. Truth functions of hedge connectives.

	$V^\bullet$	$H^\bullet$	$R^\bullet$	$S^\bullet$
0	0	0	0	0
Vc^-	Vc^-	Vc^-	Hc^-	c^-
Hc^-	Vc^-	Hc^-	c^-	Rc^-
c^-	Vc^-	Hc^-	Rc^-	Sc^-
Rc^-	Hc^-	c^-	Rc^-	Sc^-
Sc^-	c^-	Rc^-	Sc^-	Sc^-
W	W	W	W	W
Sc^+	Sc^+	Sc^+	Rc^+	c^+
Rc^+	Sc^+	Rc^+	c^+	Hc^+
c^+	Sc^+	Rc^+	Hc^+	Vc^+
Hc^+	Rc^+	c^+	Hc^+	Vc^+
Vc^+	c^+	Hc^+	Vc^+	Vc^+
1	1	1	1	1

3.3. Operations on Linguistic Truth Domains

In MVL/FLn/MFL [32,57,58], there are three prominent sets of connectives called Łukasiewicz, Gödel, and product logic ones. Each of the sets has a pair of a continuous t-norm and its residuum. Since our truth values are linguistic, we cannot use the product logic connectives.

On an LTD $\overline{X} = \{v_0, \ldots, v_n\}$ with $v_0 \le v_1 \le \cdots \le v_n$, Gödel t-norm, its residuum, and t-conorm can be, respectively, defined as follows [4,7]:

$$\begin{aligned} \mathcal{C}_G(v_i, v_j) &= \min(v_i, v_j), \\ \leftarrow^\bullet_G (v_j, v_i) &= \begin{cases} v_n & \text{if } i \le j \\ v_j & \text{otherwise,} \end{cases} \\ \vee^\bullet_G(v_i, v_j) &= \max(v_i, v_j). \end{aligned}$$

Łukasiewicz t-norm, its residuum, and t-conorm can be, respectively, defined by [4,7]:

$$\begin{aligned} \mathcal{C}_L(v_i, v_j) &= \begin{cases} v_{i+j-n} & \text{if } i+j-n > 0 \\ v_0 & \text{otherwise,} \end{cases} \\ \leftarrow^\bullet_L (v_j, v_i) &= \begin{cases} v_n & \text{if } i \le j \\ v_{n+j-i} & \text{otherwise,} \end{cases} \\ \vee^\bullet_L(v_i, v_j) &= \begin{cases} v_{i+j} & \text{if } i+j < n \\ v_n & \text{otherwise.} \end{cases} \end{aligned}$$

The residua are non-decreasing in the first argument, but non-increasing in the second. The other operations are non-decreasing in all arguments.

Every pair of a t-norm $\mathcal{C}$ and its residuum $\leftarrow^\bullet$ satisfy the following properties [32], for truth values b, r, and h (standing for *body*, *rule* and *head*, respectively):

$$\mathcal{C}(b, r) \le h \text{ iff } r \le \leftarrow^\bullet (h, b) \tag{9}$$

$$(\forall b)(\forall h)\ \mathcal{C}(b, \leftarrow^\bullet (h, b)) \le h \tag{10}$$

$$(\forall b)(\forall r)\ \leftarrow^\bullet (\mathcal{C}(b, r), b) \ge r \tag{11}$$

The *negation* of a truth value x, denoted by $-x$, is defined as follows: if $x = \sigma c$, where σ is a string of hedges (including the empty one) and $c \in \{c^+, c^-\}$, then $-x = \sigma c'$ where $\{c, c'\} = \{c^+, c^-\}$; and $-1 = 0$, $-0 = 1$, and $-W = W$. For instance, $-Vc^- = Vc^+$ and $-Vc^+ = Vc^-$. The negation is decreasing and satisfies $- - x = x$.

In [6], it is shown that the LOWA (Linguistic Ordered Weighted Averaging) operator [59], one of the most well-known linguistic aggregation operators that can directly compute with linguistic labels, can be used in body formulas of FLLP. Aggregation operators are very useful since they let us describe increased fulfillment of user requirements. A conjunction is one extreme case in which one desires that all the criteria be satisfied, and a disjunction is the other extreme in which the satisfaction of any of the criteria is all one needs. The LOWA operator is developed based on the ordered weighted averaging (OWA) operator [60] and the convex combination of linguistic labels [61].

Definition 1 (LOWA operator, [59]). *Let $S = \{s_1, \ldots, s_m\}$ be a set of linguistic terms to be aggregated, the LOWA operator $@^\bullet(s_1, \ldots, s_m) = C^m\{w_k, t_k, k = 1 \ldots m\}$ is defined inductively as follows.*

For $m = 2$,

$$C^2\{\{w_1, 1 - w_1\}, \{t_1, t_2\}\} = (w_1 \odot v_j) \oplus ((1 - w_1) \odot v_i) = v_k$$

where $t_1 = v_j, t_2 = v_i \in \overline{X}, j \geq i$, and $k = min\{n, i + round(w_1.(j - i))\}$, in which $n + 1$ is the cardinality of $\overline{X}$ and round*(.) is the usual round operation.*

For $m > 2$,

$$C^m\{w_k, t_k, k = 1 \ldots m\} = C^2\{\{w_1, 1 - w_1\}, \{t_1, C^{m-1}\{\eta_h, t_h, h = 2 \ldots m\}\}\}$$

where $W = [w_1, \ldots, w_m]$ is a weighting vector associated with S such that (i) $w_i \in [0, 1]$, and (ii) $\sum_{i=1}^m w_i = 1$; $T = [t_1, \ldots, t_m]$ is a vector such that t_i is the ith largest element in the collection $s_1, \ldots, s_m$; $\eta_h = w_h / \sum_2^m w_k, h = 2, \ldots, m$.

It can be seen that the LOWA operator is not well defined for the case $m > 2$ and $\sum_{k=k_0}^m w_k = 0$, for $k_0 \leq m - 1$.

A natural question arising is how to acquire the associated weighting vector. An interesting method to compute the weights of the OWA operator using linguistic quantifiers is proposed by Yager [60]. More concretely, if Q is a relative or proportional quantifier such as "*Most*", Q can be represented by a fuzzy subset of $[0, 1]$ such that for each $r \in [0, 1]$, $Q(r)$ states the degree to which r portion of objects satisfies the concept denoted by Q. Then, the weights can be computed by:

$$w_i = Q(i/n) - Q((i - 1)/n), i = 1, \ldots, n$$

The membership function of such a quantifier Q can be:

$$Q(r) = \begin{cases} 0 & \text{if } 0 \leq r < a \\ \frac{r-a}{b-a} & \text{if } a \leq r \leq b \\ 1 & \text{if } 1 \geq r > b \end{cases}$$

where $0 \leq a \leq b \leq 1$.

Due to the non-decreasing nature of Q, it follows that $w_i \geq 0$. Moreover, since $Q(1) = 1$ and $Q(0) = 0$, we have $\sum_{i=1}^n w_i = 1$. The use of those quantifiers to compute the weighting vector of the LOWA operator essentially implies that the more criteria are satisfied, the better the solution is.

The LOWA operator has the following properties [59]: (i) it is commutative, i.e., $@^\bullet(s_1, \ldots, s_m) = @^\bullet(\pi(s_1), \ldots, \pi(s_m))$, where π is a permutation over the set of arguments; (ii) it is non-decreasing in all arguments, i.e., given $S = [s_1, \ldots, s_m]$ and $T = [t_1, \ldots, t_m]$ being two vectors such that for all $i, s_i \geq t_i$, we have $@^\bullet(S) \geq @^\bullet(T)$; and (iii) it is an *or–and* operator, i.e., $min(s_i) \leq @^\bullet(s_1, \ldots, s_m) \leq max(s_i)$.

Since in all the proofs of the results in [4], we only require that truth functions of connectives in body formulas (other than negation) be non-decreasing in all arguments, allowing body formulas to be built using a LOWA operator does not affect any results.

Example 3. *If the quantifier "*Most*" with a=0.3 and b=0.8 is used to generate weighting vectors for the LOWA operator, the weighting vector of dimension 3 is* $[w_1 = 1/15, w_2 = 2/3, w_3 = 4/15]$. *We have the following:*

$$@^\bullet(Rc^+, Sc^+, Hc^+) \quad = \quad @^\bullet(v_8, v_7, v_{10}) = v_8 = Rc^+$$

3.4. Fuzzy Linguistic Logic Programming without Negation

The language is a many-sorted (or typed) predicate language without function symbols. Clauses are without negation. With no function symbols, Herbrand universes of all sorts of variables of a finite logic program are finite, and so is its Herbrand base. Together with a finite LTD, this allows us to obtain the least Herbrand model of logic programs by finitely iterating the immediate consequence operator from the least Herbrand interpretation [4]. Also, the least Herbrand model of a logic program can be obtained using finitely terminating tabulation proof procedures [7]. The underlying language of a program is assumed to be defined by constants and predicate symbols occurring in it (if no constants exist, some constant a is added to form ground terms).

Connectives are composed of the following: $\wedge_G, \vee_G, \leftarrow_G$ (Gödel conjunction, disjunction, and implication); $\wedge_L, \vee_L, \leftarrow_L$ (Łukasiewicz conjunction, disjunction, and implication); $@_i$ (aggregations whose truth functions are the LOWA operator with a specific weighting vector, and thus, we can have many distinct aggregation operators with different weighting vectors); and hedges as unary connectives. The truth function of a connective c is denoted by $c^\bullet$.

A *term* is either a constant or a variable. An *atom* is of the form $p(t_1, ..., t_n)$, where p is an n-ary predicate symbol, and $t_1, ..., t_n$ are terms. A *fact* is a graded atom $(A.a)$, where the atom A and the truth value a (different from 0) are the logical part and the truth value of the fact, respectively. A *body formula* is defined inductively as follows: (i) an atom is a body formula; (ii) if c is a connective with arity n other than the implications, and $B_i, i = \overline{1,n}$, are body formulas, so is $c(B_1, B_2, \ldots, B_n)$. A *rule* is a graded implication $(A \leftarrow B.r)$, where the atom A is called a *rule head*, the body formula B is called a *rule body*, r is the truth value (different from 0), and $(A \leftarrow B)$ is called the *logical part* of the rule. In every graded formula $(\varphi.t)$, the truth value t is understood as a lower bound to the exact truth value of φ.

A *fuzzy linguistic logic program* (also called a *positive program*) is a finite set of rules and facts, and there are no two rules (facts) with the same logical part but different truth values. Hence, a program P can be represented as a partial mapping:

$$P : Formulas \to \overline{X} \setminus \{0\},$$

where the domain of P, denoted $dom(P)$, is finite and consists only of logical parts of facts and rules. For any rule or fact $(\varphi.t) \in P$, $P(\varphi) = t$. A *fuzzy linguistic Herbrand interpretation* (interpretation, for short) f of P is a mapping from the Herbrand base B_P to $\overline{X}$, assigning to every ground atom a truth value.

The ordering $\leq$ in $\overline{X}$ is expanded to interpretations as follows: for all interpretations f_1 and f_2 of a program P, $f_1 \sqsubseteq f_2$ if for all $A \in B_P$, $f_1(A) \leq f_2(A)$. The meet (infimum, greatest lower bound) and join (supremum, least upper bound) operators of interpretations are denoted by $\otimes$ and $\oplus$, respectively; for all interpretations f_1, f_2 and atoms A, we have: (i) $(f_1 \otimes f_2)(A) = f_1(A) \otimes f_2(A)$, and (ii) $(f_1 \oplus f_2)(A) = f_1(A) \oplus f_2(A)$ [6].

An interpretation f is extended to all ground formulas, denoted by $\overline{f}$, as follows: (i) $\overline{f}(A) = f(A)$, if A is a ground atom; (ii) $\overline{f}(c(B_1, B_2, \ldots, B_n)) = c^\bullet(\overline{f}(B_1), \overline{f}(B_2), \ldots, \overline{f}(B_n))$, where $B_i, i = \overline{1,n}$, are ground formulas, and c is a connective with arity n. For non-ground formulas, since all variables in formulas are assumed to be universally quantified, $\overline{f}(\varphi) = \overline{f}(\forall\varphi) = \otimes\{\overline{f}(\varphi\vartheta) \mid \varphi\vartheta$ is a ground instance of $\varphi\}$, where $\forall\varphi$ denotes the *universal closure* of φ, which is obtained from φ by adding a universal quantifier for every variable with a free occurrence in φ.

Let P be a program. An interpretation f is a *Herbrand model* (model, for short) of a rule/fact $(\varphi.t) \in P$ if $\overline{f}(\varphi) \geq P(\varphi) = t$; and f is a model of P if f is a model of all formulas $(\varphi.t) \in P$.

Theorem 1 ([6]). *Let P be a positive program.*
(i) Let $\mathcal{F}_P$ be the set of all interpretations of P. Then $\mathcal{F}_P$ is non-empty, and $\langle \mathcal{F}_P, \otimes, \oplus \rangle$ is a complete lattice.
(ii) Let F be a non-empty set of models of P. Then $\otimes F$ is a model of P.
(iii) $M_P = \otimes\{f \mid f$ is a model of $P\}$ is the least model of P.

Definition 2 (Immediate consequence operator). *Let P be a positive program. The operator T_P, mapping from interpretations to interpretations, is defined as follows: for an interpretation f and a ground atom $A \in B_P$,*
$T_P(f)(A) = max\{\oplus\{\mathcal{C}_i(\overline{f}(B), r) \mid (A \leftarrow_i B.r)$ is a ground instance of a rule in $P\}$, $\oplus\{a \mid (A.a)$ is a ground instance of a fact in $P\}\}$.

Since the Herbrand base B_P is finite, for each $A \in B_P$, there are a finite number of ground instances of logical parts of facts and rule heads that match A. Hence, both $\oplus$ operators in the definition of T_P are, in fact, maxima.

Theorem 2 ([4]). *Let P be a positive program. Then T_P is non-decreasing and continuous.*

A model of a positive program P is a pre-fixpoint of T_P [62] and vice versa.

Theorem 3 ([4]). *An interpretation f is a model of a positive program P iff $T_P(f) \sqsubseteq f$.*

The bottom-up iteration of T_P is defined as:

$$T_P^k(\bot) = \begin{cases} \bot & \text{if } k = 0 \\ T_P(T_P^{k-1}(\bot)) & \text{if } k \text{ is a successor ordinal} \\ \oplus\{T_P^n(\bot) \mid n < k\} & \text{if } k \text{ is a limit ordinal,} \end{cases}$$

where $\bot$ denotes the least interpretation mapping each ground atom to 0. The bottom-up iteration of T_P is denoted by $T_P^\infty(\bot)$. The top-down iteration of T_P is defined similarly and denoted by $T_P^\infty(\top)$, where $\top$ is the greatest interpretation mapping each ground atom to 1.

The least model M_P is exactly the least fixpoint of T_P, denoted $lfp(T_P)$, and can be obtained by finitely bottom-up iterating T_P as follows.

Theorem 4 ([4]). *Let P be a positive program. Then there exists a finite number α such that $n \geq \alpha$ implies $T_P^n(\bot) = lfp(T_P) = M_P = T_P^\infty(\bot)$.*

Analogously, the top-down iteration of T_P, $T_P^\infty(\top)$, can also be finitely obtained.

Example 4. *Assume the following piece of knowledge:*
(a) (A car is good if it is rather *reliable and consumes* very *little fuel) is* very true;
(b) (A car is good if it is expensive) is rather true;
(c) (A car is safe if it is reliable) is very true;
(d) (A car is reliable if it is safe) is highly true;
(e) (A Toyota Corolla is safe) is rather true;
(f) (A Toyota Corolla is reliable) is true;
(g) (A Toyota Corolla consumes little fuel) is very true;
(h) (A Toyota Corolla is expensive) is slightly true.

The knowledge piece can be represented by the program below:

$$
\begin{aligned}
(gd(X) \leftarrow_G \wedge_G(R\, rl(X), V\, lf(X)).Vc^+) &\quad (r1)\\
(gd(X) \leftarrow_L ep(X).Rc^+) &\quad (r2)\\
(sf(X) \leftarrow_G rl(X).Vc^+) &\quad (r3)\\
(rl(X) \leftarrow_G sf(X).Hc^+) &\quad (r4)\\
(sf(cor).Rc^+) &\quad (f1)\\
(rl(cor).c^+) &\quad (f2)\\
(lf(cor).Vc^+) &\quad (f3)\\
(ep(cor).Sc^+) &\quad (f4)
\end{aligned}
$$

where gd, rl, lf, ep, sf, and cor stand for good, reliable, little fuel, expensive, safe, *and* Toyota Corolla, *respectively. Note that the program is recursive.*

The bottom-up iteration of T_P is as follows:

$$
\begin{aligned}
T_P(\bot) &= \{(gd(cor),0),(sf(cor),Rc^+),(rl(cor),c^+),(lf(cor),Vc^+),(ep(cor),Sc^+)\}\\
T_P^2(\bot) &= \{(gd(cor),c^+),(sf(cor),c^+),(rl(cor),c^+),(lf(cor),Vc^+),(ep(cor),Sc^+)\}\\
T_P^3(\bot) &= \{(gd(cor),c^+),(sf(cor),c^+),(rl(cor),c^+),(lf(cor),Vc^+),(ep(cor),Sc^+)\}
\end{aligned}
$$

Since $T_P^2(\bot)$ coincides with $T_P^3(\bot)$, it is also the least model of P.

4. Normal Fuzzy Linguistic Logic Programs and Their Semantics

FLLP is extended by permitting negative literals to appear in rule bodies, resulting in *normal fuzzy linguistic logic programs* (also called *normal programs*). An *extended positive program* is a positive program where elements of the truth domain can occur in the places of atoms in rule bodies, and the value of such an element in all interpretations is itself.

The notions of an interpretation, a model, and the immediate consequence operator T_P of an extended positive/normal program P are defined the same as those of a positive program.

4.1. The Stable Model Semantics of Normal Programs

Results in this subsection are presented in our preliminary paper [63]. We refer the reader to that paper for detailed proofs of the theorems in this subsection.

The stable model semantics of normal programs is generalized from that of classical normal logic programs based on the usual Gelfond–Lifschitz transformation [11]. Let P be a normal program, and P^* denote the program consisting of all ground instances of rules and facts in P. Given an interpretation J of P, the *reduct* P^J of P w.r.t. J is the extended positive program obtained by substituting in P^* all negative literals with their values w.r.t. J, computed as $\overline{J}(\neg A) = -J(A)$ for all ground atoms A.

Definition 3 (Stable model)**.** *Let P be a normal program and I an interpretation of P. Then I is a* stable model *of P if $I = I'$, where I' is the least model of the reduct P^I.*

The *stable model semantics* of a normal program is a whole family of its stable models.

Theorem 5 ([63])**.** *A stable model of a normal program is its model.*

A normal program may have more than one stable model.

Example 5. *Given the LTD in Example 1, consider a program P composed of the rules below:*

$$
\begin{aligned}
(good(X) &\leftarrow_G \neg bad(X).Vc^+)\\
(bad(X) &\leftarrow_G \neg good(X).Vc^+)
\end{aligned}
$$

We determine its stable models. Given an interpretation $I = \{(good(a), x), (bad(a), y)\}$ of P, in which the constant a is added to form ground terms, the reduct P^I is composed of the rules below:

$$\begin{array}{rcl} (good(a) & \leftarrow_G & -y.Vc^+) \\ (bad(a) & \leftarrow_G & -x.Vc^+) \end{array}$$

The least model of P^I is $M_{P^I} = \{(good(a), \mathcal{C}_G(-y, Vc^+)), (bad(a), \mathcal{C}_G(-x, Vc^+))\}$. Hence, I is a stable model of P if $x = \mathcal{C}_G(-y, Vc^+) = \min(-y, Vc^+)$ and $y = \mathcal{C}_G(-x, Vc^+) = \min(-x, Vc^+)$. It can easily be seen that all $Vc^- \leq x = -y \leq Vc^+$ (so $Vc^- \leq y = -x \leq Vc^+$) satisfy the equations. Thus, there are 11 stable models $\{(good(a), x), (bad(a), -x)\}$ where $v_1 = Vc^- \leq x \leq Vc^+ = v_{11}$. These stable models are minimal models, and the program does not have a least stable model.

Theorem 6 ([63])**.** *A stable model of a normal program is its minimal model.*

Theorem 3 also holds for normal programs.

Theorem 7 ([63])**.** *An interpretation f is a model of a normal program P iff $T_P(f) \sqsubseteq f$.*

Example 6. *Consider a normal program P composed of the rule:*

$$(good(X) \leftarrow_G \neg bad(X).1)$$

Given an interpretation $I = \{(good(a), x), (bad(a), y)\}$ with x and y being truth values, $T_P(I) = \{(good(a), -y), (bad(a), 0)\}$. Obviously, $T_P(I)$ is not non-decreasing.

Since for a normal program P, T_P may not be non-decreasing, T_P does not necessarily have a least fixpoint.

Theorem 8 ([63])**.** *A stable model of a normal program P is a minimal fixpoint of T_P.*

For a normal program P, T_P may have many minimal fixpoints, and a minimal fixpoint of T_P might not be a stable model of P.

Example 7. *Consider a normal program P composed of the rules below:*

$$\begin{array}{rcl} (good(X) & \leftarrow_G & good(X).1) \\ (bad(X) & \leftarrow_G & \neg good(X).1) \end{array}$$

First, we determine its stable models. Given an interpretation $I = \{(good(a), x), (bad(a), y)\}$ of P, the reduct P^I is the following:

$$\begin{array}{rcl} (good(a) & \leftarrow_G & good(a).1) \\ (bad(a) & \leftarrow_G & -x.1) \end{array}$$

The least model of P^I is $M_{P^I} = \{(good(a), 0), (bad(a), -x)\}$. Hence, I is a stable model of P if $I = M_{P^I}$, i.e., $I = \{(good(a), 0), (bad(a), 1)\}$.

Second, we determine fixpoints of T_P. Given an interpretation

$$I = \{(good(a), x), (bad(a), y)\},$$

we have $T_P(I) = \{(good(a), x), (bad(a), -x)\}$. Thus, I is a fixpoint of T_P if for any truth value x, $I = \{(good(a), x), (bad(a), -x)\}$. It can be observed that all these fixpoints are minimal, but only $\{(good(a), 0), (bad(a), 1)\}$ is a stable model.

Moreover, for a normal program P, the bottom–up iteration of T_P might not reach any of its stable models.

Example 8. *Consider the program P in Example 5:*

$$\begin{array}{rcl}(good(X) & \leftarrow_G & \neg bad(X).Vc^+) \\ (bad(X) & \leftarrow_G & \neg good(X).Vc^+)\end{array}$$

It has 11 stable models $I = \{(good(a), x), (bad(a), -x)\}$, where $v_1 = Vc^- \leq x \leq Vc^+ = v_{11}$, but the bottom-up iteration of T_P does not converge to a fixpoint as follows.

$$\begin{array}{rcl}T_P(\bot) & = & \{(good(a), Vc^+), (bad(a), Vc^+)\} \\ T_P^2(\bot) & = & \{(good(a), Vc^-), (bad(a), Vc^-)\} \\ T_P^3(\bot) & = & \{(good(a), Vc^+), (bad(a), Vc^+)\} \\ \dots & & \end{array}$$

Therefore, the operator T_P might not help compute any stable model.

4.2. The Well-Founded Semantics of Normal Programs

In this subsection, we define the well-founded semantics of a normal program based on the ideas in [16,36].

To ease the presentation below, we first define a *binary* immediate consequence operator $T_P(I, J)$ for a normal program P as an extension of the unary one, in which the interpretations I and J are utilized to assign truth values to positive literals and negative literals, respectively. Since the original $T_P(.)$ and the binary $T_P(.,.)$ are different in arity, by providing all the arguments, we can use the same notation T_P for both of them without confusion.

Definition 4 (Binary immediate consequence operator)**.** *Let P be a normal program and $\mathcal{F}_P$ the set of all interpretations of P. The operator $T_P(.,.)$ mapping from $\mathcal{F}_P \times \mathcal{F}_P$ to $\mathcal{F}_P$ is defined as follows: for every pair of interpretations (I, J) and each ground atom $A \in B_P$, $T_P(I, J)(A) = T_{P^J}(I)(A)$.*

Note that if both a positive literal A and the negative one $\neg A$ simultaneously occur in the program, I and J independently assign truth values to them.

Theorem 9. *The binary T_P is*
(i) non-decreasing in the first argument and
(ii) non-increasing in the second.

Proof. (i) If we fix an interpretation J for negative literals in P, then by definition, the reduct P^J is an extended positive program. By Theorem 2, $T_{P^J}(I)$ is non-decreasing, and so is $T_P(I, J)$.

(ii) Given an interpretation I for positive literals and two interpretations J_1 and J_2 for negative literals in P such that $J_1 \sqsubseteq J_2$, we will prove that $T_P(I, J_2) \sqsubseteq T_P(I, J_1)$.

Let $(A \leftarrow B[B_1, \dots, B_m, \neg B_{m+1}, \dots, \neg B_n].r)$ be any rule in P. For each of its ground instances $(A\vartheta \leftarrow B[B_1\vartheta, \dots, B_m\vartheta, \neg B_{m+1}\vartheta, \dots, \neg B_n\vartheta].r)$ in P^*, the counterpart rules in P^{J_1} and P^{J_2} are, respectively:

$$(A\vartheta \leftarrow B[B_1\vartheta, \dots, B_m\vartheta, -J_1(B_{m+1}\vartheta), \dots, -J_1(B_n\vartheta)].r)$$

and

$$(A\vartheta \leftarrow B[B_1\vartheta, \dots, B_m\vartheta, -J_2(B_{m+1}\vartheta), \dots, -J_2(B_n\vartheta)].r).$$

Since $J_1 \sqsubseteq J_2$, for all $m+1 \leq i \leq n$, $-J_1(B_i\vartheta) \geq -J_2(B_i\vartheta)$. Hence, we have:

$$\begin{aligned} & \mathcal{C}(\bar{I}(B[B_1\vartheta, \ldots, B_m\vartheta, -J_1(B_{m+1}\vartheta), \ldots, -J_1(B_n\vartheta)]), r) \\ = \; & \mathcal{C}(\mathcal{B}(I(B_1\vartheta), \ldots, B_m\vartheta, -J_1(B_{m+1}\vartheta), \ldots, -J_1(B_n\vartheta)), r) \\ \geq \; & \mathcal{C}(\mathcal{B}(I(B_1\vartheta), \ldots, B_m\vartheta, -J_2(B_{m+1}\vartheta), \ldots, -J_2(B_n\vartheta)), r) \\ = \; & \mathcal{C}(\bar{I}(B[B_1\vartheta, \ldots, B_m\vartheta, -J_2(B_{m+1}\vartheta), \ldots, -J_2(B_n\vartheta)]), r) \end{aligned}$$

By taking suprema for all the rules, we have:

$$\begin{aligned} & \oplus\{\mathcal{C}(\bar{I}(B[B_1\vartheta, \ldots, B_m\vartheta, -J_1(B_{m+1}\vartheta), \ldots, -J_1(B_n\vartheta)]), r)\} \\ \geq \; & \oplus\{\mathcal{C}(\bar{I}(B[B_1\vartheta, \ldots, B_m\vartheta, -J_2(B_{m+1}\vartheta), \ldots, -J_2(B_n\vartheta)]), r)\} \end{aligned}$$

Moreover, since the fact parts of P^{J_1} and P^{J_2} are the same, by definition, we have

$$\begin{aligned} & T_P(I, J_1)(A) \\ = \; & T_{P^{J_1}}(I)(A) \\ \geq \; & T_{P^{J_2}}(I)(A) \\ = \; & T_P(I, J_2)(A). \end{aligned}$$

Since the rule in P is arbitrary, $T_P(I, J_2) \sqsubseteq T_P(I, J_1)$. □

Definition 5. *Let P be a normal program. The operator S_P mapping from $\mathcal{F}_P$ to $\mathcal{F}_P$ is defined as $S_P(J) = T^{\infty}_{P^J}(\perp) = lfp(T_{P^J})$, i.e., $S_P(J)$ computes the least model of P^J.*

By definition, a stable model of P is a fixpoint of S_P. In addition to using the bottom–up iteration of T_{P^J}, $S_P(J)$ can be computed using tabulation proof procedures in [7], which are finitely terminating.

Theorem 10. *Let P be a normal program. The S_P operator is non-increasing.*

Proof. Recall that $S_P(J) = T^{\infty}_{P^J}(\perp)$. Given two interpretations J_1 and J_2 for negative literals in P such that $J_1 \sqsubseteq J_2$, we will prove by induction on $k \geq 0$ that $T^k_{P^{J_2}}(\perp) \sqsubseteq T^k_{P^{J_1}}(\perp)$. The base case $k = 0$ trivially holds. Suppose $T^k_{P^{J_2}}(\perp) \sqsubseteq T^k_{P^{J_1}}(\perp)$; we will show that $T^{k+1}_{P^{J_2}}(\perp) \sqsubseteq T^{k+1}_{P^{J_1}}(\perp)$. We have the following:

$$\begin{aligned} & T^{k+1}_{P^{J_2}}(\perp) \\ = \; & T_{P^{J_2}}(T^k_{P^{J_2}}(\perp)) \\ \sqsubseteq \; & T_{P^{J_2}}(T^k_{P^{J_1}}(\perp)), \text{ by Theorem 2} \\ = \; & T_P(T^k_{P^{J_1}}(\perp), J_2), \text{ by Definition 4} \\ \sqsubseteq \; & T_P(T^k_{P^{J_1}}(\perp), J_1), \text{ by Theorem 9} \\ = \; & T_{P^{J_1}}(T^k_{P^{J_1}}(\perp)), \text{ by Definition 4} \\ = \; & T^{k+1}_{P^{J_1}}(\perp) \end{aligned}$$

□

The following immediately follows.

Proposition 1. *$S_P \circ S_P$ (S_P composed with S_P) is non-decreasing.*

Clearly, Item (i) of Theorem 1 also holds for normal programs, as stated in the following proposition.

Proposition 2. *Let P be a normal program and $\mathcal{F}_P$ the set of all interpretations of P. Then $\mathcal{F}_P$ is non-empty and $\langle \mathcal{F}_P, \sqsubseteq \rangle$ is a complete lattice.*

The following results can be derived from the Knaster–Tarski theorem [64].

Proposition 3 ([19,65])**.** *Let f be a non-increasing function on a complete lattice with the ordering $\leq$. Then f has a unique pair of extreme oscillation points, a and b. More precisely, we have the following:*

(i) a and b are the least fixpoint and the greatest fixpoint of $f \circ f$, respectively, and $a \leq b$;
(ii) $f(a) = b$ and $f(b) = a$;
(iii) If $f(x) = y$ and $f(y) = x$ then both x and y are between a and b in the lattice ordering.

The following proposition follows immediately from Proposition 3.

Proposition 4. *Let P be a normal program. Then we have:*

(i) $(S_P \circ S_P)^\infty(\perp)$, denoted by $S_P^\perp$, is the least fixpoint of $(S_P \circ S_P)$; $(S_P \circ S_P)^\infty(\top)$, denoted by $S_P^\top$, is the greatest fixpoint of $(S_P \circ S_P)$. Thus, $S_P^\perp \sqsubseteq S_P^\top$.

(ii) $S_P^\perp$ and $S_P^\top$ are two extreme oscillation points of S_P, i.e., $S_P(S_P^\perp) = S_P^\top$ and $S_P(S_P^\top) = S_P^\perp$, and for all interpretations I, J such that $I = S_P(J)$ and $J = S_P(I)$, we have $S_P^\perp \sqsubseteq I, J \sqsubseteq S_P^\top$.

(iii) If an interpretation I is a stable model of P, i.e., $I = S_P(I)$, then $S_P^\perp \sqsubseteq I \sqsubseteq S_P^\top$. In particular, if $S_P^\perp = S_P^\top$, $S_P^\perp$ is the only stable model of P.

Therefore, $S_P^\perp$ and $S_P^\top$ can be seen as a lower-bound approximation and an upper-bound approximation of stable models of P, respectively.

Since given any interpretation J, $S_P(J)$ can be computed finitely, both $S_P^\perp$ and $S_P^\top$ can be computed finitely as well.

Example 9. *Given the program P in Example 5, composed of the rules below:*

$$\begin{array}{rcl} (good(X) & \leftarrow_G & \neg bad(X).Vc^+) \\ (bad(X) & \leftarrow_G & \neg good(X).Vc^+), \end{array}$$

and the interpretation $\perp = \{(good(a), 0), (bad(a), 0)\}$, the reduct $P^\perp$ is as follows:

$$\begin{array}{rcl} (good(a) & \leftarrow_G & 1.Vc^+) \\ (bad(a) & \leftarrow_G & 1.Vc^+) \end{array}$$

The least model of $P^\perp$ is $S_P(\perp) = \{(good(a), Vc^+), (bad(a), Vc^+)\}$. Then the reduct $P^{S_P(\perp)}$ is the following:

$$\begin{array}{rcl} (good(a) & \leftarrow_G & Vc^-.Vc^+) \\ (bad(a) & \leftarrow_G & Vc^-.Vc^+) \end{array}$$

The least model of $P^{S_P(\perp)}$ is

$$(S_P \circ S_P)(\perp) = S_P(S_P(\perp)) = \{(good(a), Vc^-), (bad(a), Vc^-)\}.$$

Then $(S_P \circ S_P)^2(\perp) = \{(good(a), Vc^-), (bad(a), Vc^-)\}$.
Hence, $S_P^\perp = (S_P \circ S_P)^\infty(\perp) = \{(good(a), Vc^-), (bad(a), Vc^-)\}$, denoted by μ.
Similarly, $S_P^\top = (S_P \circ S_P)^\infty(\top) = \{(good(a), Vc^+), (bad(a), Vc^+)\}$, denoted by ν.
It can be observed that $S_P(\mu) = \nu$ and $S_P(\nu) = \mu$.
As shown in Example 5, P has 11 stable models $I = \{(good(a), x), (bad(a), -x)\}$, where $v_1 = Vc^- \leq x \leq Vc^+ = v_{11}$. Obviously, all the stable models lie between μ and ν.

In order to define the well-founded semantics, we first define the notion of a *partial interpretation*.

Definition 6 (Partial interpretation)**.** *Let P be a normal program. A* partial interpretation *I of P is a partial mapping from B_P to the truth domain.*

Note that an interpretation is a partial interpretation. The ground atoms that are assigned a truth value by a partial interpretation I are said to be *I-defined*, and the other atoms in B_P are *I-undefined*. If an atom is I-defined, so is its negative literal. Also, if an atom is I-undefined, so is its negative literal. A formula (rule or fact) in P is said to be *I-defined* if all the literals appearing in it are I-defined. Otherwise, the formula is *I-undefined*.

Definition 7 (Partial model)**.** *Let P be a normal program and I a partial interpretation of P. We say that I is a* partial model *of a formula $(\phi.t) \in P^*$ if any of the following holds:*
(i) ϕ is I-defined and $\overline{I}(\phi) \geq t$;
(ii) ϕ is I-undefined.
Moreover, it is said that I is a partial model *of P as well as P^* if I is a partial model of all formulas in P^*.*

Note that a model is a partial model.

The notion of reduct program can be extended for the case of partial interpretations as follows. Given a partial interpretation J of a normal program P, the *partial reduct* P^J_p of P w.r.t. J is the extended program obtained by substituting in P^* all J-defined negative literals with their values w.r.t. J, and the other literals remain.

Given the partial reduct program P^J_p of a normal program P w.r.t. a partial interpretation J, we denote by P^J_I, where I is another partial interpretation of P, the program consisting of all the I-defined rules and facts in P^J_p.

Proposition 5. *Let P be a normal program and I a partial interpretation of P. Then I is a partial model of P if I is a model of P^I_I.*

Proof. First, it can be seen that I is a (total) interpretation of P^I_I, and thus, P^I_I is an extended positive program. Then, the result follows from the fact that all I-defined formulas $(\phi.t) \in P^*$ have a corresponding formula $(\phi'.t) \in P^I_I$, where ϕ' is obtained from ϕ by replacing all negative literals with their values under I, and vice versa, and obviously, $\overline{I}(\phi) = \overline{I}(\phi')$. □

As in the *alternating fixpoint approach* [16,36], $S^{\perp}_P$ and $S^{\top}_P$ in Proposition 4 are an under-estimation and an over-estimation of the semantics of P, respectively. If we consider a (partial) interpretation as a set in which each element consists of a ground atom and its truth value, the well-founded semantics of normal programs can be defined as follows.

Definition 8 (Well-founded semantics)**.** *Let P be a normal program. The set $S^{\perp}_P \cap S^{\top}_P$ is denoted by W_P. Then W_P is called the* well-founded (partial) model *of P.*

Clearly, all normal programs have a unique well-founded model. Since $S^{\perp}_P$ and $S^{\top}_P$ can be computed finitely, the well-founded model W_P can also be computed finitely.

Theorem 11. *Let P be a normal program. The well-founded model W_P is a partial model of P.*

Proof. We will show that W_P is a model of the extended positive program $P^{W_P}_{W_P}$. Then by Proposition 5, we obtain the result.

By Definition 5, $S_P(S^{\perp}_P)$ computes the least model of $P^{S^{\perp}_P}$. On the other hand, by Item (ii) of Proposition 4, we have $S_P(S^{\perp}_P) = S^{\top}_P$. Therefore, $S^{\top}_P$ is the least model of $P^{S^{\perp}_P}$.

It can be seen that since W_P is a part of the (total) interpretation $S_P^{\perp}$, the program $P_{W_P}^{W_P}$ is a part of the program $P^{S_P^{\perp}}$, which is obtained from $P^{S_P^{\perp}}$ by removing all W_P-undefined rules and facts. Since $S_P^{\top}$ is the least model of $P^{S_P^{\perp}}$, $S_P^{\top}$ is also a model of $P_{W_P}^{W_P}$. Since W_P is the projection of $S_P^{\top}$ on the Herbrand base of $P_{W_P}^{W_P}$, W_P is a model of $P_{W_P}^{W_P}$. □

4.3. The Relation between the Stable Semantics and the Well-Founded Semantics

We denote the domain of W_P by $dom(W_P)$, which is the Herbrand base of $P_{W_P}^{W_P}$.

Theorem 12. *Every stable model of a normal program P contains its well-founded model W_P.*

Proof. Let I be a stable model of P. By Item (iii) of Proposition 4, we have $S_P^{\perp} \sqsubseteq I \sqsubseteq S_P^{\top}$. Thus, for every ground atom $A \in dom(W_P)$, we have $S_P^{\perp}(A) \leq I(A) \leq S_P^{\top}(A)$. On the other hand, for all $A \in dom(W_P)$, we have $W_P(A) = S_P^{\perp}(A) = S_P^{\top}(A)$, so $I(A) = W_P(A)$. □

Corollary 1. *If the well-founded model of a normal program is total, then it is its unique stable model.*

Proof. Given a normal program P, since W_P is a total model, we have $S_P^{\perp} = S_P^{\top}$. By Proposition 4, W_P is the only stable model of P. □

Example 10. *Given the LTD in Example 1, consider the normal program P composed of the rules and fact below:*

$$\begin{array}{ccc} (good(X) & \leftarrow_G & good(X).W) \\ (bad(X) & \leftarrow_G & \neg good(X).W) \\ & (good(a).W) & \end{array}$$

First, we determine stable models of P. Given an interpretation $I = \{(good(a), x), (bad(a), y)\}$, the reduct P^I is the following:

$$\begin{array}{ccc} (good(a) & \leftarrow_G & good(a).W) \\ (bad(a) & \leftarrow_G & -x.W) \\ & (good(a).W) & \end{array}$$

The least model of P^I is $M_{P^I} = \{(good(a), W), (bad(a), \min(-x, W))\}$. Hence, I is a stable model of P if $I = M_{P^I}$, i.e., $I = \{(good(a), W), (bad(a), W)\}$ (note that $-W = W$).

Given $\perp = \{(good(a), 0), (bad(a), 0)\}$, $P^{\perp}$ is:

$$\begin{array}{ccc} (good(a) & \leftarrow_G & good(a).W) \\ (bad(a) & \leftarrow_G & 1.W) \\ & (good(a).W) & \end{array}$$

It is easy to see that for $n \geq 1$, $S_P^n(\perp) = \{(good(a), W), (bad(a), W)\}$. That is,

$$S_P^{\perp} = \{(good(a), W), (bad(a), W)\}.$$

Similarly, given $\top = \{(good(a), 1), (bad(a), 1)\}$, $P^{\top}$ is the following:

$$\begin{array}{ccc} (good(a) & \leftarrow_G & good(a).W) \\ (bad(a) & \leftarrow_G & 0.W) \\ & (good(a).W) & \end{array}$$

and $S_P(\top) = \{(good(a), W), (bad(a), 0)\}$.

For $n \geq 2$, $S_P^n(\top) = \{(good(a), W), (bad(a), W)\}$. That is,

$$S_P^\top = \{(good(a), W), (bad(a), W)\}.$$

Therefore, $S_P^\perp = S_P^\top$ and $W_P = \{(good(a), W), (bad(a), W)\}$ is the only stable model of P.

However, the converse is not true.

Example 11. *Consider a program P having only the rule:*

$$(good(a) \leftarrow_G \neg good(a).1)$$

Given an interpretation $I = \{(good(a), x)\}$, the reduct P^I is

$$(good(a) \leftarrow_G -x.1),$$

which has the least model $M_{P^I} = \{(good(a), -x)\}$. Thus, I is a stable model if $I = M_{P^I}$, i.e., $x = -x = W$. Therefore, the program has a unique stable model $\{(good(a), W)\}$. On the other hand, it is not difficult to see that $S_P^\perp = \{(good(a), 0)\}$ and $S_P^\top = \{(good(a), 1)\}$, so the well-founded model is $W_P = \emptyset$, and it does not coincide with the unique stable model.

In LP and ASP, there is a well-known method to compute stable models/answer sets of normal programs, called "*guess-and-check*" [9,48,49]. It consists of two steps: (1) guessing a candidate for a stable model/answer set of a normal logic program, and (2) checking whether the candidate is, in fact, a stable model/answer set of the program. Given a normal program P, since $S_P^\perp$ and $S_P^\top$ are, respectively, a lower bound and an upper bound of stable models of P, they can be used to reduce the number of candidates for Step (1). Note that for FLLP, both $S_P^\perp$ and $S_P^\top$ can be computed finitely.

Example 12. *Given the LTD in Example 1, consider a program P composed of the rules below:*

$$\begin{array}{rcl} (good(X) & \leftarrow_G & \neg bad(X).Sc^+) \\ (bad(X) & \leftarrow_G & \neg good(X).Sc^+) \end{array}$$

Similar to the computation in Example 9, we have $S_P^\perp = \{(good(a), Sc^-), (bad(a), Sc^-)\}$ and $S_P^\top = \{(good(a), Sc^+), (bad(a), Sc^+)\}$. Thus, in order to compute stable models of the program, we only need to check six candidates $\{(good(a), x), (bad(a), y)\}$, where $v_5 = Sc^- \leq x, y \leq Sc^+ = v_7$, instead of checking all 26 possible candidates $\{(good(a), x), (bad(a), y)\}$, where $v_0 = 0 \leq x, y \leq v_{12} = 1$. In fact, the program has three stable models $\{(good(a), x), (bad(a), -x)\}$, where $v_5 = Sc^- \leq x \leq Sc^+ = v_7$.

5. Conclusions and Future Work

In this paper, we extend FLLP by allowing negative literals to appear in rule bodies, resulting in normal logic programs. We study both the stable model semantics and the well-founded semantics of such programs and their relation in terms of fixpoints of several operators. More concretely, we first adapt the stable model semantics of classical LP to FLLP based on the usual Gelfond–Lifschitz transformation. In fact, according to the literature, this seems to be the only way to study the stable model semantics of normal logic programs. Since our hedge connectives have no counterpart in any FLP framework, our FLLP [4] is not a special case of any of the FLP frameworks based on the well-known Vojtáš's FLP, especially MALP, which appears to be the most expressive one among those. Therefore, our stable model semantics is not a special case of that of any of the frameworks. Furthermore, we are the first ones studying the well-founded semantics in an FLP framework among those based on Vojtáš's FLP. The well-founded semantics is characterized by two operators $S_P^\perp$ and $S_P^\top$, which can be computed finitely. It can be seen that our well-founded semantics can be adapted for other FLP frameworks. We then study the relation between the two kinds of semantics. In fact, in the literature, this relation is usually studied using a bilattice setting of the truth domain. However, our truth domains do not possess a complete knowledge-ordering lattice and, thus, do not have a bilattice structure. Interestingly, in FLLP, the two kinds of semantics possess properties similar to those of the classical case. More precisely, every stable model contains the well-founded (partial) model, and the well-founded total model coincides with the unique stable model, but not vice versa. Moreover, in LP and ASP, there is a well-known method to compute stable models/answer sets of normal logic programs, called "*guess-and-check*" [9,48,49]. It is composed of two steps: (1) guessing a candidate for a stable model/answer set of a normal logic program, and (2) checking whether the candidate is, in fact, a stable model/answer set of the program. Since for a normal program P, $S_P^{\perp}$ and $S_P^{\top}$ are, respectively, a lower bound and an upper bound of stable models, they can be used to reduce the number of candidates for Step (1). That is, the well-founded semantics can make the computation of stable models more efficiently.

For future work, we will study other well-known kinds of semantics of normal logic programs such as *perfect model semantics*, *stationary semantics* [9], and their relations with the two kinds of semantics studied in this work as well as with each other.

Funding: This research is funded by the Vietnam Ministry of Education and Training under grant number CT.2019.01.02.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Chen, S.J.J.; Hwang, C.L. *Fuzzy Multiple Attribute Decision Making: Methods and Applications*; Springer Inc.: Secaucus, NJ, USA, 1992.
2. Levrat, L.; Voisin, A.; Bombardier, S.; Bremont, J. Subjective evaluation of car seat comfort with fuzzy set techniques. *Int. J. Intell. Syst.* **1997**, *12*, 891–913.
3. Cao, Z.; Kandel, A. Applicability of some fuzzy implication operators. *Fuzzy Sets Syst.* **1989**, *31*, 151–186.
4. Le, V.H.; Liu, F.; Tran, D.K. Fuzzy linguistic logic programming and its applications. *Theory Pract. Log. Program.* **2009**, *9*, 309–341.
5. Zadeh, L.A. Fuzzy Logic. *Computer* **1988**, *21*, 83–93.
6. Le, V.H.; Tran, D.K. Further Results on Fuzzy Linguistic Logic Programming. *J. Comput. Sci. Cybern.* **2014**, *30*, 139–147.
7. Le, V.H.; Liu, F. Tabulation proof procedures for fuzzy linguistic logic programming. *Int. J. Approx. Reason.* **2015**, *63*, 62–88.
8. Le, V.H. Efficient Query Answering for Fuzzy Linguistic Logic Programming. In Proceedings of the 9th International Conference on Computing & Communication Technologies, Research, Innovation, and Vision for the Future, RIVF 2012, Ho Chi Minh city, Vietnam, February 27–March 1, 2012; pp. 113–116.
9. Apt, K.R.; Bol, R.N. Logic programming and negation: A survey. *J. Log. Program.* **1994**, *19-20*, 9–71.
10. van Gelder, A.; Ross, K.A.; Schlipf, J.S. The well-founded semantics for general logic programs. *J. ACM* **1991**, *38*, 619–649.
11. Gelfond, M.; Lifschitz, V. The stable model semantics for logic programming. In Proceedings of the 5th International Conference on Logic Programming, Seattle, WA, USA, 15–19 August 1988; pp. 1070–1080.
12. Schlipf, J.S. The Expressive Powers of the Logic Programming Semantics. *J. Comput. Syst. Sci.* **1995**, *51*, 64–86.
13. Truszczynski, M. An introduction to the stable and well-founded semantics of logic programs. In *Declarative Logic Programming: Theory, Systems, and Applications*; Kifer, M.; Liu, Y.A., Eds.; ACM/Morgan & Claypool: San Rafael, CA, USA, 2018.
14. Cornejo, M.E.; Lobo, D.; Medina, J. Syntax and semantics of multi-adjoint normal logic programming. *Fuzzy Sets Syst.* **2018**, *345*, 41–62.
15. Madrid, N.; Ojeda-Aciego, M. On the existence and unicity of stable models in normal residuated logic programs. *Int. J. Comput. Math.* **2012**, *89*, 310–324.
16. Loyer, Y.; Straccia, U. The Well-Founded Semantics in Normal Logic Programs with Uncertainty. In Proceedings of the 6th International Symposium on Functional and Logic Programming, Aizu, Japan, 15–17 September 2002; Springer: London, UK, 2002; pp. 152–166.
17. Ginsberg, M.L. Multi-valued logics: a uniform approach to reasoning in Artificial Intelligence. *Comput. Intell.* **1988**, *4*, 265–316.
18. Fitting, M. Bilattices and the semantics of logic programming. *J. Log. Program.* **1991**, *11*, 91–116.
19. Fitting, M. Fixpoint semantics for logic programming: A survey. *Theor. Comput. Sci.* **2002**, *278*, 25–51.
20. Fitting, M. The Family of Stable Models. *J. Log. Program.* **1993**, *17*, 197–225.

21. Loyer, Y.; Straccia, U. Approximate Well-Founded Semantics, Query Answering and Generalized Normal Logic Programs over Lattices. *Ann. Math. Artif. Intell.* **2009**, *55*, 389–417.
22. Loyer, Y.; Straccia, U. The Approximate Well-Founded Semantics for Logic Programs with Uncertainty. In Proceedings of the 28th International Symposium on Mathematical Foundations of Computer Science, Prague, Czech Republic, 24–28 August 2020; Rovan, B., Vojtáš, P., Eds.; Lecture Notes in Computer Science; Springer: New York, USA, 2003, Volume 2747, pp. 541–550.
23. Vojtáš, P. Fuzzy logic programming. *Fuzzy Sets Syst.* **2001**, *124*, 361–370.
24. Straccia, U. Managing Uncertainty and Vagueness in Description Logics, Logic Programs and Description Logic Programs. In Proceedings of the 4th International Summer School on Reasoning Web, Venice, Italy, 7–11 September 2008; Lecture Notes in Computer Science, Springer: New York, NY, USA, Volume 5224, pp. 54–103.
25. Lakshmanan, L.V.S.; Shiri, N. A Parametric Approach to Deductive Databases with Uncertainty. *IEEE Trans. Knowl. Data Eng.* **2001**, *13*, 554–570.
26. Krajči, S.; Lencses, R.; Vojtáš, P. A comparison of fuzzy and annotated logic programming. *Fuzzy Sets Syst.* **2004**, *144*, 173–192.
27. Damásio, C.V.; Pereira, L.M. Antitonic Logic Programs. In Proceedings of 6th International Conference on Logic Programming and Nonmonotonic Reasoning, LPNMR 2001, Vienna, Austria, September 17-19, 2001; Lecture Notes in Computer Science, Volume 2173, pp. 379–392.
28. Gallier, J.H. *Logic for Computer Science: Foundations of Automatic Theorem Proving*; Harper & Row Publishers, Inc.: New York, NY, USA, 1985.
29. Medina, J.; Ojeda-Aciego, M.; Vojtás, P. Multi-adjoint Logic Programming with Continuous Semantics. In Proceedings of the 6th International Conference on Logic Programming and Nonmonotonic Reasoning, LPNMR 2001, Vienna, Austria, 17–19 September 2001; Eiter, T., Faber, W., Truszczynski, M., Eds.; Lecture Notes in Computer Science; Springer: New York, USA, 2001; Volume 2173, pp. 351–364.
30. Medina, J.; Ojeda-Aciego, M.; Vojtáš, P. Similarity-based unification: a multi-adjoint approach. *Fuzzy Sets Syst.* **2004**, *146*, 43–62.
31. Le, V.H.; Nguyen, C.H.; Liu, F. Semantics and Aggregation of Linguistic Information Based on Hedge Algebras. In Proceedings of the 3rd International Conference on Knowledge, Information and Creativity Support Systems, KICSS 2008, Hanoi, Vietnam, 22-23 December 2028, pp. 128–135.
32. Hájek, P. *Metamathematics of Fuzzy Logic*; Kluwer: Dordrecht, The Netherlands, 1998.
33. Straccia, U.; Madrid, N. A Top-k Query Answering Procedure for Fuzzy Logic Programming. *Fuzzy Sets Syst.* **2012**, *205*, 1–29.
34. Pan, J.Z.; Stoilos, G.; Stamou, G.B.; Tzouvaras, V.; Horrocks, I. f-SWRL: A Fuzzy Extension of SWRL. *J. Data Semant.* **2006**, *6*, 28–46.
35. van Emden, M.H. Quantitative Deduction and Its Fixpoint Theory. *J. Log. Program.* **1986**, *3*, 37–53.
36. van Gelder, A. The Alternating Fixpoint of Logic Programs with Negation. *J. Comput. Syst. Sci.* **1993**, *47*, 185–221.
37. Belnap, N.D., Jr. A Useful Four-Valued Logic. In *Modern Uses of Multiple-Valued Logic*; Dunn, J.M., Epstein, G., Eds.; D. Reidel Publishing Co.: Dordrecht, The Netherland, 1977; pp. 5–37.
38. Marek, V.W.; Truszczynski, M. Stable Models and an Alternative Logic Programming Paradigm. In *The Logic Programming Paradigm—A 25-Year Perspective*; Apt, K.R., Marek, V.W., Truszczynski, M., Warren, D.S., Eds.; Artif. Intell.; Springer: New York, NY, USA, 1999; pp. 375–398.
39. Niemelä, I. Logic Programs with Stable Model Semantics as a Constraint Programming Paradigm. *Ann. Math. Artif. Intell.* **1999**, *25*, 241–273.
40. Brewka, G.; Eiter, T.; Truszczynski, M. Answer set programming at a glance. *Commun. ACM* **2011**, *54*, 92–103.
41. Gebser, M.; Kaminski, R.; Kaufmann, B.; Schaub, T. *Answer Set Solving in Practice*; Synthesis Lectures on Artificial Intelligence and Machine Learning; Morgan & Claypool Publishers: San Rafael, CA, USA 2012.
42. Blondeel, M.; Schockaert, S.; Vermeir, D.; Cock, M.D. Fuzzy Answer Set Programming: An Introduction. In *Soft Computing: State of the Art Theory and Novel Applications*; Yager, R.R., Abbasov, A.M., Reformat, M.Z., Shahbazova, S.N., Eds.; Studies in Fuzziness and Soft Computing; Springer: New York, NY, USA, 2013; Volume 291, pp. 209–222.
43. Nieuwenborgh, D.V.; Cock, M.D.; Vermeir, D. An introduction to fuzzy answer set programming. *Ann. Math. Artif. Intell.* **2007**, *50*, 363–388.
44. Janssen, J.; Vermeir, D.; Schockaert, S.; Cock, M.D. Reducing fuzzy answer set programming to model finding in fuzzy logics. *TPLP* **2012**, *12*, 811–842.
45. Janssen, J.; Schockaert, S.; Vermeir, D.; Cock, M.D. Aggregated Fuzzy Answer Set Programming. *Ann. Math. Artif. Intell.* **2011**, *63*, 103–147.
46. Cornejo, M.E.; Lobo, D.; Medina, J. Relating Multi-Adjoint Normal Logic Programs to Core Fuzzy Answer Set Programs from a Semantical Approach. *Mathematics* **2020**, *8*, 881.
47. Cornejo, M.E.; Lobo, D.; Medina, J. Extended multi-adjoint logic programming. *Fuzzy Sets Syst.* **2020**, *388*, 124–145.
48. Eiter, T.; Polleres, A. Towards automated integration of guess and check programs in answer set programming: A meta-interpreter and applications. *THeory Pract. Log. Program.* **2006**, *6*, 23–60.
49. Vienna University of Technology. DLVHEX System. Available online: http://www.kr.tuwien.ac.at/research/systems/dlvhex/ (accessed on 18 August 2022).
50. Nguyen, C.H.; Wechler, W. Hedge algebras: An algebraic approach to structure of sets of linguistic truth values. *Fuzzy Sets Syst.* **1990**, *35*, 281–293.

51. Nguyen, C.H.; Wechler, W. Extended hedge algebras and their application to fuzzy logic. *Fuzzy Sets Syst.* **1992**, *52*, 259–281.
52. Le, V.H.; Tran, D.K. Extending fuzzy logics with many hedges. *Fuzzy Sets Syst.* **2018**, *345*, 126–138.
53. Le, V.H.; Liu, F.; Tran, D.K. Mathematical Fuzzy Logic with Many Dual Hedges. In Proceedings of the 5th Symposium on Information and Communication Technology, SoICT 2014, Hanoi, Vietnam, December 4-5, 2014; pp. 7–13.
54. Zadeh, L.A. A Theory of Approximate Reasoning. In *Machine Intelligence*; Hayes, J.E., Michie, D., Mikulich, L.I., Eds.; Halsted Press: Ultimo, Australia, 1979; Volume 9, pp. 149–194.
55. Bellman, R.E.; Zadeh, L.A. Local and fuzzy logics. In *Fuzzy Sets, Fuzzy Logic, and Fuzzy Systems: Selected Papers by Lotfi A. Zadeh*; World Scientific Publishing Co., Inc.: River Edge, NJ, USA, 1996; pp. 283–335.
56. Nguyen, C.H.; Tran, D.K.; Huynh, V.N.; Nguyen, H.C. Hedge algebras, linguistic-value logic and their application to fuzzy reasoning. *Int. J. Uncertain. Fuzziness Knowl. Based Syst.* **1999**, *7*, 347–361.
57. Novák, V.; Perfilieva, I.; Mockor, J. *Mathematical Principles of Fuzzy Logic*; Kluwer: Dordrecht, The Netherland, 2000.
58. Cintula, P.; Hájek, P.; Noguera, C. (Eds.). *Handbook of Mathematical Fuzzy Logic*; Studies in Logic, Mathematical Logic and Foundations; College Publications: London, UK 2011.
59. Herrera, F.; Verdegay, J.L. Linguistic assessments in group decision. In Proceedings of the 1st European Congress on Fuzzy and Intelligent Technologies, Aachen, Germany, 7–10 September 1993; pp. 941–948.
60. Yager, R. On Ordered Weighted Averaging Aggregation Operators in Multicriteria Decisionmaking. *IEEE Trans. on Syst. Man Cyber.* **1988**, *18*, 183–190.
61. Delgado, M.; Verdegay, J.; Vila, M. On aggregation operations of linguistic labels. *Int. J. Intell. Syst.* **1993**, *8*, 351–370.
62. Davey, B.A.; Priestley, H.A. *Introduction to Lattices and Order*; Cambridge University Press: Cambridge, UK, 2002.
63. Le, V.H. The Stable Model Semantics of Normal Fuzzy Linguistic Logic Programs. In Proceedings of the 11th International Conference on Computational Collective Intelligence, ICCCI 2019, Hendaye, France, 4–6 September 2019; Part I; Nguyen, N.T., Chbeir, R., Exposito, E., Aniorté, P., Trawinski, B., Eds.; Lecture Notes in Computer Science; Springer: New York, NY, USA, 2019; Volume 11683, pp. 53–65.
64. Tarski, A. A lattice-theoretical fixpoint theorem and its applications. *Pac. J. Math.* **1955**, *5*, 285–309.
65. Fitting, M. Bilattices Are Nice Things. In *Self-Reference*; Bolander, T., Hendricks, V., Pedersen, S.A., Eds.; CSLI Publications: Stanford, CA, USA, 2006; pp. 53–77.