IS THE SUCCESS OF FUZZY LOGIC REALLY PARADOXICAL? OR: TOWARDS THE ACTUAL LOGIC BEHIND EXPERT SYSTEMS

Hung T. Nguyen[*], Olga M. Kosheleva[**,***], Vladik Kreinovich[**]

[*]*Department of Mathematical Sciences, New Mexico State University*
Las Cruces, NM 88003
[**]*Computer Science Department, University of Texas at El Paso*
El Paso, TX 79968, USA, email vladik@cs.utep.edu
[***]*Department of Electrical Engineering, University of Texas at El Paso*
El Paso, TX 79968, USA

Abstract. The formal concept of logical equivalence in fuzzy logic[8,25], while theoretically sound, seems impractical. The misinterpretation of this concept has led to some pessimistic conclusions[4]. Motivated by practical interpretation of truth values for fuzzy propositions, we take the class (lattice) of all *sub-intervals* of the unit interval [0,1] as the truth value space for fuzzy logic, subsuming the traditional class of *numerical* truth values from [0,1]. The associated concept of logical equivalence is stronger than the traditional one. Technically, we are dealing with much smaller set of pairs of equivalent formulas, so that we are able to check equivalence algorithmically. The checking is done by showing that our strong equivalence notion coincides with the equivalence in *logic programming.*

1. INTRODUCTION

Since fuzzy logic is a formalization of commonsense reasoning, statements that are "equivalent" according to commonsense reasoning must have the same truth values in this logic. This informal argument was used by several researchers in fuzzy logic[6,13,25]. How can one formalize it? Definitely, this "commonsense equivalence" is not the same as equivalence in 2-valued logic; this difference was noticed in the first papers on fuzzy logic, and it was recently mentioned again by Elkan[4]. Is there any reasonable formalization of this "equivalence"? In this paper, we solve this problem by providing a formal definition of such "equivalence". We also prove that this definition relates exactly the same pairs of formulas as a definition that stems from *logic programming.* The very fact that two radically different approaches (fuzzy logic and logic programming) lead to exactly the same notion of equivalence between propositional formulas, is, to our viewpoint, an additional indication that both approaches lead to adequate descriptions of commonsense reasoning (at least as far as equivalence goes). In addition to that, we provide an algorithm that enables us to tell whether two given propositional formulas are equivalent in this sense or not.

In his 1993 paper[4], Elkan describes the difficulties that he has encountered when he tried to describe fuzzy logic formally. Elkan tried two different formalizations of this "equivalence" based on two most well-known logical formalisms:

- two-valued propositional logic, and
- intuitionistic logic.

In both cases, he got a contradiction. This contradiction is easy to explain: in both logics (two-valued and intuitionistic), $v_1 \vee (v_2 \& \neg v_2)$ is equivalent to v_1. However, in fuzzy logic, for truth values $t(v_1) = 0.3$ and $t(v_2) = 0.4$, we get $t(\neg v_2) = 1 - t(v_2) = 0.4$ and therefore, $t(v_2 \& \neg v_2) = \min(t(v_2), t(\neg v_2)) = \min(0.6, 0.4) = 0.4 \neq 0.3 = t(v_1)$.

Since both his attempts of formalization (based on reasonable choices of logic) has failed, Elkan concludes that fuzzy logic does not have consistent foundations and that therefore, successes of fuzzy logic are paradoxical.

We do not agree with this pessimistic conclusion, for three reasons:

- First, researchers in mathematical foundations of fuzzy sets and fuzzy control do not use the fact that if F and G are "equivalent" then their truth values coincide ($t(F) = t(G)$) as an independent axiom (see, e.g.,[16,17,9,14,18,23,26,20,21]).
- Second, as far as we know, neither this "postulate" itself, nor any conclusions based on this postulate have ever been used in any real-life fuzzy control applications[15,1,11,21,27].
- Third, there *exists* a consistent formalization of this postulate, and this formalization has been proposed by Goguen[8] and Schwartz[25] (we will describe it in just a moment).

(other arguments against Elkan's pessimism can be found in[5]).

In spite of this confusion, the notion of commonsense equivalence is useful and therefore, worth formalizing. Definitely, this "commonsense equivalence" is not the same as equivalence in 2-valued logic; this difference was noticed in the first papers on fuzzy logic[8] (see also[25], and it was recently mentioned again by Elkan (see the above example). The same example works if we identify this "commonsense equivalence" with equivalence in intuitionistic logic (such examples were also originally given in[8,25]).

Is there any reasonable formalization of this "equivalence"? A consistent formalization of commonsense equivalence has been proposed by Goguen in[8] and later by Schwartz in[25]: according to Goguen and Schwartz, two propositional formulas F and G with variables v_i are *equivalent* in fuzzy logic if for all possible assignments of truth values $t(v_i)$ to their variables v_i, the resulting truth values of these two formulas coincide: $t(F) = t(G)$ (actually, instead of using the interval [0,1], Goguen uses arbitrary *lattices* of truth values).

This formalization is not completely satisfactory for three reasons:

- First, it is not algorithmic. It follows closely the definition of equivalence in two-valued logic, in which formulas are equivalent iff their truth values coincide for all truth values $t(v_i) \in \{T, F\}$. However, from the computational viewpoint, there is a big difference between these two cases:
  - In two-valued logic, each variable v_i has only two possible truth values. Therefore, we can actually enumerate all 2^n combinations of truth values and thus check whether the formulas F and G are equivalent or not.
  - In fuzzy case, each variable v_i has infinitely many possible truth values (the set of truth values coincides with the interval [0,1]). Therefore, it is impossible to enumerate and test all possible combinations of truth values.
- Second, this definition is very formal. It would be nice to have some extra support for the assumption that this definition reflects commonsense equivalence of two propositional formulas.
- Finally, it is not exactly clear what values of $t(v_i)$ we should use:

- If we use only real numbers from the interval [0,1], then we neglect the fact that every estimation of the truth values is in itself approximate.
- If we use arbitrary lattices, then we are using mathematical model that may have no relation whatsoever to commonsense reasoning.

The problem of describing a formalization of "commonsense equivalence" that is free from these three drawbacks was, in effect, formulated in[4] in the following terms: *It is an open question how to choose a notion of logical equivalence that simultaneously (i) remains philosophically justifiable, (ii) allows useful inferences in practice, and (iii) removes the opportunity to prove results similar to Theorem 1* (i.e., inconsistency).

In this paper, we are going to address these three problems. Namely:

- First, we will choose the proper representation of the truth values as *sub-intervals* of [0,1] (this idea also goes back to Goguen and Schwartz).
- Second, we will show that the resulting definition of equivalence of propositional formulas is the same as the definition of equivalence stemming from an absolutely different area of Artificial Intelligence: namely, from logic programming. The fact that two radically different approaches (fuzzy logic and logic programming) lead to exactly the same notion of equivalence between propositional formulas is, to our viewpoint, an indication that both approaches lead to adequate descriptions of commonsense reasoning (at least as far as equivalence is concerned).
- Third, this equivalence enables us to design an *algorithm* that given two propositional formulas F and G, decides whether these formulas are equivalent or not.

2. PRELIMINARY DEFINITIONS: PROPOSITIONAL FORMULAS AND EQUIVALENCE IN 2-VALUED LOGIC

To make the paper easier to read, let us start with reviewing the notion of equivalence in a classical (2–valued) logical setting.

Definition 2.1. *Assume that a finite set $V = \{v_1, ..., v_n\}$ is given. Its elements v_i will be called propositional* (or *Boolean*) *variables. A propositional formula with n variables v_i is then defined as follows:*

- *every variable v_i is a formula;*
- *if F is a formula, then $\neg F$ (called not F) is also a formula;*
- *if F and G are formulas, then the following are formulas:*
    - *$F \& G$ (called F and G);*
    - *$F \vee G$ (called F or G);*
- *nothing else is a formula.*

Formulas that have been used in the construction of a formula F are called its subformulas. Formulas different from variables are called composite formulas.

Examples of propositional formulas:

- $v_1 \vee (\neg v_2)$ is a propositional formula that has the following subformulas:
  - v_1, v_2, and v_3;
  - $\neg v_2$;
  - $v_1 \vee (\neg v_3)$.
- $v_1 \& (v_2 \vee \neg v_1)$ is a propositional formula that has the following subformulas:
  - v_1, v_2, and v_3;
  - $\neg v_1$;
  - $v_2 \vee \neg v_1$;
  - $v_1 \& (v_2 \vee \neg v_1)$.

Definition 2.2. *By an $n-$dimensional Boolean vector, or a Boolean array t, we mean a function from the set V into the set of truth values $\{T, F\}$ (where $T = 1$ stands for "true", and $F = 0$ stands for "false").*

Definition 2.3. *Let t be a Boolean vector. Then, for every formula F, we can define its truth value $t(F)$ for this Boolean vector t as follows:*

- *if $F = v_i$, then $t(F) = t(v_i)$;*
- *if $F = \neg G$, then $t(F) = \neg t(G)$;*
- *if $F = G \& H$, then $t(F) = t(G) \& t(H)$;*
- *if $F = G \vee H$, then $t(F) = t(G) \vee t(H)$,*

(where $\neg$, $\&$, and $\vee$ in the right hand sides mean standard operations with Boolean values $\{T, F\}$).

Example. If $t(v_1) = T$, and $t(v_2) = F$, then for $F = v_1 \vee (\neg v_2)$, the truth value $t(F)$ is defined as follows:

- $t(\neg v_2) = \neg F = T$;
- $t(F) = t(v_1 \vee (\neg v_2)) = t(v_1) \vee t(\neg v_2) = T \vee T = T$.

Definition 2.4. *We say that formulas F and G are equivalent in 2-valued logic if for every Boolean vector t, $t(F) = t(G)$.*

3. FUZZY LOGIC: WHAT TRUTH VALUES SHOULD WE CHOOSE?

3.1. Traditional fuzzy logic

One of the main reasons for fuzzy logic is that we want to design computer systems that would incorporate the knowledge of human experts. When applied for decision making, such systems are called *expert systems*. When apply for control, they are called *intelligent control systems*. Usually, the knowledge of an expert contains some statements that this expert is not 100% sure about, and these statements form an essential part of the knowledge base. If we want to use this knowledge base, it is important to know the degree of belief that an expert has in each statement, so that, e.g., in case of conflicting conclusions we will be able to choose the conclusion that is supported by the more convincing statements from the knowledge base. So, an adequate computer representation of this knowledge must include not only the representation of the experts' *statements* themselves, but it must also contain the computer representation of the experts' *degrees* with which they believe in these statements. Traditionally, these degrees of belief are represented by numbers from the interval [0,1] (0 corresponds to "false", and 1 to "true").

There are several possible ways to assign a degree of belief to a statement (see, e.g.,[2,3]). Let us just name three of them:

- We can ask an expert to estimate his degree of belief by a number on a scale from, say, 0 to 5. If he marks 3, then we say that his degree of belief is $3/5 = 0.6$.
- We can ask several experts whether they believe in a given statement, and as a degree of belief, take a fraction of those who answered "yes".
- Another way to get $t(S)$ is to ask an expert to compare an alternative "\$100 if S is true else 0" with the lotteries in which this same amount of money (\$100) is guaranteed with a certain probability p. If there is a probability p for which the above alternative and a lottery are equivalent, then we take p as $t(S)$.

In classical (2-valued) logic, it was sufficient to assign truth values to all basic statements v_i; then we can compute the truth value of any composite statement (that is formed from v_i by applying &, $\vee$, and $\neg$). In case of degrees of belief, it is no longer so. In general, the degree of belief $t(F\&G)$ in a statement $F\&G$ is *not* uniquely determined by the degrees of belief $t(F)$ and $t(G)$. Let us give an extreme example: assume that we know nothing about a statement F. Then, it is natural to assign both to F, and to its negation $\neg F$, the degree of belief 0.5. Let us now consider two cases:

- If we take $G = F$, then we have $t(F) = t(G) = 0.5$. Here, $F\&G = F\&F$ means the same as F, so $t(F\&G) = 0.5$.
- It we take $G = \neg F$, then $t(F) = t(G) = 0.5$, but $F\&G = F\&\neg F$ is simply false, so $t(F\&G) = 0$.

In both cases, $t(F) = t(G) = 0.5$, but the values of $t(F\&G)$ are different.

In view of that, it would be nice to know not only the experts' degrees of belief in elementary statements $v_1, ..., v_n$, but also their degrees of beliefs in composite statements. This is, alas, not feasible: even for pairs $v_i\&v_j$, we need thus to acquire from experts $\approx n^2$ numbers, which for large n is practically impossible. So, in practice, we only know the degrees of belief $t(v_i)$ in the original statements v_i. What should we do if the decision that we are about to make must be based on the degree of belief in $v_i\&v_j$?

A natural approach to this problem is as follows. We must be able, given two numbers $t_1 = t(F)$ and $t_2 = t(G)$, to generate an estimate for $t(F\&G)$. As we have mentioned, there can be different situations with these values t_i that lead to different values of $t(F\&G)$. So, to be on the safe side, it is natural to take the *average* of these values as the desired estimate for $t(F\&G)$. There has been several experimental papers in which the corresponding averages have been found[10,22,29]. It turns out that in most situations, the average for & is best described by either the product $t_1 \cdot t_2$, or by the minimum $\min(t_1, t_2)$. Similarly, the best estimate for $\vee$ is best described either by $t_1 + t_2 - t_1 \cdot t_2$, or by maximum $\max(t_1, t_2)$. All four operations have been originally proposed by L. Zadeh in his pioneer paper on fuzzy set theory[28].

In other words, if we know the degrees of belief in F and in G, then, as an *estimate* $t(F\&G)$ for the actual (unknown) degree of belief of $F\&G$, we take either $t(F) \cdot t(G)$, or

$\min(t(F), t(G))$. In the present paper, we consider only the simplest case of min and max. So, we arrive at the following definitions.

Definition 3.1. *By an initial fuzzy truth value assignment, we mean a function t from the set V into the set $[0, 1]$.*

Definition 3.2. *Let t be an initial truth value assignment. Then, for every formula F, we can define its truth value $t(F)$ as follows:*

- *if $F = v_i$, then $t(F) = t(v_i)$;*
- *if $F = \neg G$, then $t(F) = 1 - t(G)$;*
- *if $F = G \& H$, then $t(F) = \min(t(G), t(H))$;*
- *if $F = G \vee H$, then $t(F) = \max(t(G), t(H))$.*

Example. If $t(v_1) = 0.3$, and $t(v_2) = 0.6$, then for $F = v_1 \vee (\neg v_2)$, the truth value $t(F)$ is defined as follows:

- $t(\neg v_2) = 1 - 0.6 = 0.4$;
- $t(F) = t(v_1 \vee (\neg v_2)) = t(v_1) \vee t(\neg v_2)) = \max(0.3, 0.4) = 0.4$.

3.2. Intervals are a more realistic representation of degrees of belief than real numbers

The above definitions is based on the assumption that we can describe the degree of belief in a statement by a precise number. However, this is an idealization of the real situation. For example, suppose that we are getting the value $t(S)$ by asking an expert to compare an alternative "\$100 if S is true else 0" with the lotteries in which this same amount of money (\$100) is guaranteed with a certain probability p. If there is a probability p for which the above alternative and a lottery are equivalent, then we take p as $t(S)$. In some cases, it is possible. In many other cases, it is difficult for a person to compare. In such cases, instead of a single probability p, we get a whole interval of probabilities $[p^-, p^+]$ for which the above alternative is more or less indistinguishable from a lottery.

A similar problem occurs for other approaches. As a result, instead of a *single* value $t(F)$, we get an *interval* $\mathbf{t}(F) = [t^-(F), t^+(F)]$ of possible values of degree of belief. So, it is reasonable to describe the degree of belief in a statement by an interval.

What if we know the intervals $\mathbf{t}(F)$ and $\mathbf{t}(G)$ of possible values of degree of belief in statements F and G, and we are interested in the degree of belief in $F \& G$? The fact that we know $\mathbf{t}(F)$ means that any number t_1 from this interval can be the degree of belief for F. Similarly, any number t_2 from the interval $\mathbf{t}(G)$ can be the degree of belief in G. If we use min for &, then possible values $\min(t_1, t_2)$ form an interval of possible values of degrees of belief for $F \& G$.

To be able to compute this interval, we must thus be able, given two intervals $\mathbf{t}$ and $\mathbf{s}$, to compute the interval of possible values of $\min(t, s)$, where $t \in \mathbf{t}$ and $s \in \mathbf{s}$. For $\vee$ and $\neg$, we must be able to compute similar sets for max and for $1 - t$. Let us describe how to do it.

Definition 3.3. *Let $f : R^m \to R$ be an arbitrary function of n real variables, and let $\mathbf{x}_1, ..., \mathbf{x}_m$ be m intervals. Then, we denote*

$$f(\mathbf{x}_1, ..., \mathbf{x}_m) = \{f(x_1, ..., x_n) \,|\, x_1 \in \mathbf{x}_1, ..., x_m \in \mathbf{x}_m\}.$$

Comment. We will use this definition only for three functions $1-t$, min, and max that correspond to three logical connectives $\neg$, &, and $\vee$.

PROPOSITION 1.

- *For $f(t) = 1-t$, we have* $1-[t^-, t^+] = [1-t^+, 1-t^-]$.
- *For $f = \min$, we have* $\min([t^-, t^+], [s^-, s^+]) = [\min(t^-, s^-), \min(t^+, s^+)]$.
- *For $f = \max$, we have* $\max([t^-, t^+], [s^-, s^+]) = [\max(t^-, s^-), \max(t^+, s^+)]$.

Comments. This Proposition is easy to prove if we take into consideration that all three functions $1-t$, min, and max are monotonic, and therefore, we can easily find the values form the corresponding intervals for which this function attains its biggest or its smallest value.

3.3. Practical (interval) fuzzy logic and equivalence in practical fuzzy logic

With these definitions, we can now formulate the more realistic definition of the truth value of a propositional formula:

Definition 3.4. *By an initial practical fuzzy truth value assignment, we mean a function $\mathbf{t}$ that assigns to every element v_i from the set of variables V an interval $\mathbf{t}(v_i) \subseteq [0,1]$.*

Definition 3.5. *Let $\mathbf{t}$ be an initial practical truth value assignment. Then, for every formula F, we can define its truth value $\mathbf{t}(F)$ as follows:*

- *if $F = v_i$, then $\mathbf{t}(F) = \mathbf{t}(v_i)$;*
- *if $F = \neg G$, then $\mathbf{t}(F) = 1 - \mathbf{t}(G)$;*
- *if $F = G \& H$, then $\mathbf{t}(F) = \min(\mathbf{t}(G), \mathbf{t}(H))$;*
- *if $F = G \vee H$, then $\mathbf{t}(F) = \max(\mathbf{t}(G), \mathbf{t}(H))$.*

Comment. These definitions were originally proposed by Schwartz[24].

Example. If $\mathbf{t}(v_1) = [0.2, 0.4]$, and $\mathbf{t}(v_2) = [0.6, 0.7]$, then for $F = v_1 \vee (\neg v_2)$, the truth value $\mathbf{t}(F)$ is defined as follows:

- $\mathbf{t}(\neg v_2) = 1 - [0.6, 0.7] = [1-0.7, 1-0.6] = [0.3, 0.4]$;
- $\mathbf{t}(F) = \mathbf{t}(v_1 \vee (\neg v_2)) = \mathbf{t}(v_1) \vee \mathbf{t}(\neg v_2)) = \max([0.2, 0.4], [0.3, 0.4]) =$ $[\max(0.2, 0.3), \max(0.4, 0.4)] = [0.3, 0.4]$.

Definition 3.6. *We say that formulas F and G are equivalent in practical fuzzy logic if for every initial practical fuzzy truth value assignment $\mathbf{t}$, we have $\mathbf{t}(F) = \mathbf{t}(G)$.*

Comment. In principle, there is another method to define the interval truth value of a composite formula; this method, and its difference from Definition 3.5, is given in a special Appendix,

4. THE SAME CONCEPT OF EQUIVALENCE OF PROPOSITIONAL FORMULAS STEMS FROM LOGIC PROGRAMMING

Another approach for representing commonsense reasoning (that is more traditional for Artificial Intelligence) is based on logic programming. Several researchers argue that logic programs, that were originally viewed as a representation of first order formulas, turned out to be *closer* to commonsense reasoning than the original first order logic. Traditional logic programming does not allow all propositional connectives (&, $\vee$, and $\neg$) to be used, but recently, an extension of traditional logic programming has been proposed in which all these connectives are well defined (see, e.g.,[7]). In application to propositional formulas, this extension is defined as follows:

Definition 4.1. *By a literal, we will mean either a variable v_i or its negation $\neg v_i$. Let a set W of literals be given. By induction over F, we can define when a propositional formula F is true or false for W:*

- *A literal is true for W if it belongs to W, and it is false for W if its negation belongs to W.*
- *$\neg F$ is true for W iff F is false for W, and $\neg F$ is false for W iff F is true for W.*
- *$F \vee G$ is true for W iff either F is true for W, or G is true for W; $F \vee G$ is false for W if both F and G are false for W.*
- *$F\&G$ is true for W iff both F and G are true for W; $F\&G$ is false for W iff either F is false for W, or G is false for W.*

We say that a set W is an answer set for a formula F if the following two conditions are met:

- *first, F is true for W, and*
- *second, F is not true for any proper subset of W.*

Comments.

1. One of the differences between logic programming and 2–valued logic is as follows:
   - Logic describes the objective state of the world. Therefore, in logic, for the given Boolean vector t, every formula F is either true, or false.
   - Logic programming describes our *knowledge* about the world. In this interpretation:
     - "F is true for W" means that, if W is what we know, then we know that F is true.
     - Similarly, "F is false for W" means that for the knowledge W, F is known to be false.

   Since the knowledge need not be complete, for a given set of literals W and a given formula F, in addition to two previous possibilities:
     - that F is true for W and
     - that F is false for W,

   we have two more options:
     - that F is neither true, nor false for W (meaning that if W is all we know, then whether F is true or not is unknown).
   - that F is both true and false in W, meaning that the knowledge about F that is contained in W is inconsistent.

For example, if $F = v_1 \vee \neg v_2$, and all we know is $W = \{v_3\}$ (i.e., that v_3 is true), then we have no information about v_1 and v_2 at all, and therefore, it is reasonable to conclude that F is unknown. This is exactly what we will get from Definition 4.1: that the given F is neither true, nor false for the given W. For this same F, if we take an inconsistent knowledge $W = \{v_1, \neg v_1\}$, then F is both true and false in W.

2. A formula can have several answer sets (see the example below).

Example. Let us consider the formula $F = v_1 \vee (\neg v_2)$. Then, the following two sets of literals are answer sets for this formula: $W_1 = \{v_1\}$ and $W_2 = \{\neg v_2\}$. Let us check the two conditions for W_2:

- For W_2, $\neg v_2$ is true and therefore, $v_1 \vee (\neg v_2)$ is also true for W_2.
- The only proper subset of W_2 is an empty set W. For empty set, none of the formulas v_1 and $\neg v_2$ is true; therefore, $F = v_1 \vee (\neg v_2)$ is also not true.

A set $W_3 = \{v_1, v_2\}$ is not an answer set because although F is true for W_3, but F is also true for a proper subset $W_1 \subset W_3$.

Definition 4.2. *We say that propositional formulas F and G are equivalent in logic programming if these two formulas have the same class of answer sets (i.e., if every answer set of F is an answer set of G, and vice versa).*

Now, we are ready to formulate and prove our main result.

THEOREM 1. *Let F and G be any propositional formulas. Then, F is equivalent to G in practical fuzzy logic if and only if F is equivalent to G in logic programming.*

The proof of this theorem is lengthy; so, to make reading easier, we put all the proofs in the last section.

Comments.
1. So, the desired definition of a logical equivalence that Elkan was looking for can be taken right from logic programming.

2. This Theorem was announced in[12] and[19].

3. What will happen if instead of the initial *interval* truth value assignments $\mathbf{t}$, we only consider the *numerical* truth value assignment (i.e., if all intervals $\mathbf{t}(v_i)$ contain just one number $t(v_i)$? This situation is described by the following proposition:

Definition 4.3. *We say that propositional formulas F and G are equivalent in traditional fuzzy logic if for every initial truth value assignment t, we have $t(F) = t(G)$.*

PROPOSITION 2.

- *If the formulas F and G are equivalent in practical fuzzy logic, then they are equivalent in traditional fuzzy logic.*
- *There exist formulas F and G that are equivalent in traditional fuzzy logic but not equivalent in practical fuzzy logic.*

Comments.
1. In other words, there are some formulas that are equivalent as long as we only allow numbers as truth values, but that stop being equivalent if we allow intervals as well. Therefore, equivalence classes in practical fuzzy logic are, generally speaking, much smaller than in the traditional fuzzy logic case.

2. In the next section, we will describe the algorithm that checks when the two given formulas F and G are equivalent in *practical* fuzzy logic. Suppose now that we are actually interested in whether this they are equivalent in *traditional* fuzzy logic. To do that, we will apply the algorithm for checking their equivalence in practical logic. Depending on the result of this algorithm, we will have two possibilities:

- If F and G turned out to be equivalent in practical fuzzy logic, then, due to Proposition 2, they are equivalent in traditional fuzzy logic as well.
- If, however, F and G are *not* equivalent in practical fuzzy logic, then we cannot make any conclusion about whether they are equivalent in traditional fuzzy logic or not.

3. This Proposition states that the notions of equivalence in traditional and practical fuzzy logics are different. Which of them is closer to commonsense reasoning? To our viewpoint, equivalence in practical fuzzy logic is closer. Indeed, let us consider the following example of formulas F and G that are equivalent in traditional fuzzy logic, but not in the practical one: $F = v_1 \& \neg v_1$ and $G = v_1 \& \neg v_1 \& \neg (v_2 \& \neg v_2)$.

- Let us first show that these formulas are equivalent in traditional fuzzy logic. Indeed, in traditional fuzzy logic, we have $t(F) = \min(t(v_1), 1 - t(v_1))$, and
  $t(G) = \min(t(v_1), 1 - t(v_1), 1 - \min(t(v_2), 1 - t(v_2)) = \min(t(F), 1 - \min(t(v_2), 1 - t(v_2))$.
  Let us consider two cases:
  - If $t(v_1) \le 0.5$, then $1 - t(v_1) \ge 0.5 \ge t(v_1)$, and therefore, $\min(t(v_1), 1 - t(v_1)) = t(v_1) \le 0.5$.
  - If $t(v_1) \ge 0.5$, then $1 - t(v_1) \le 0.5 \le t(v_1)$, and therefore, $\min(t(v_1), 1 - t(v_1)) = 1 - t(v_1) \le 0.5$.

  In both cases, $t(F) = \min(t(v_1), 1 - t(v_1)) \le 0.5$. Similarly, we have $\min(t(v_2), 1 - t(v_2)) \ge 0.5$, hence, $1 - \min(t(v_2), 1 - t(v_2)) \ge 0.5 \ge t(F)$, and therefore,
  $$t(G) = \min(t(F), 1 - \min(t(v_2), 1 - t(v_2)) = t(F).$$

- In interval fuzzy logic, these formulas are not equivalent: e.g., if we take $\mathbf{t}(v_1) = [0.5, 0.5]$, and $\mathbf{t}(v_2) = [0, 1]$, we get $\mathbf{t}(F) = [0.5, 0.5]$; but $\min(\mathbf{t}(v_2), 1 - \mathbf{t}(v_2)) = \min([0, 1], [0, 1]) = [0, 1]$ and therefore, $\mathbf{G} = \min([0.5, 0.5], [0, 1]) = [0, 0.5] \ne \mathbf{t}(F)$.

- We will now explain why from commonsense point of view these two formulas F and G are not exactly equivalent. To illustrate this point, we will give a commonsense example of these two logical statements. For that, we need an interpretation of the variables v_i. We interpret v_1 as "young", v_2 as "smart". In this interpretation, F would means "young and not young." Informally, any person who is to some extent still young, but at the same time, to some extent not young (i.e., who is in the intermediate age category), satisfies the property F. Similarly, a person who is to some extent smart (but not the smartest possible), but who is also to some extent not smart (i.e., who is in the intermediate smartness category) satisfies the property

$v_2\&\neg v_2$. In these terms, the property G means that a person is in the intermediate age category, but that he is *not* in the intermediate smartness category. Now, if a person belong to both the intermediate age category, and to the intermediate intelligence category, then he satisfies the property F but not the property G. So, from the commonsense viewpoint, these properties are indeed not equivalent.

5. AN ALGORITHM THAT CHECKS EQUIVALENCE IN PRACTICAL FUZZY LOGIC

5.1. Why checking equivalence is a problem

Equivalence of propositional formulas F and G in practical fuzzy logic have been defined as $\mathbf{t}(F) = \mathbf{t}(G)$ for every initial practical fuzzy truth value assignment $\mathbf{t}$. This definition is *similar* to the definition of equivalence in a 2–valued logic, where equivalence is defined as $t(F) = t(G)$ for every Boolean vector (i.e., for every initial 2–valued truth assignment) t. However, if you want to use this definition for checking whether two given formulas F and G are equivalent or not, there is a big difference between these two definitions:

- In 2–valued case, there are *finitely many* possible Boolean vectors, and therefore, we can (potentially) check them all. This gives us the algorithmic way to check whether F is equivalent to G or not.
- In practical fuzzy case, there are *infinitely many* possible initial fuzzy truth value assignments, and it is impossible to check them all directly. So, to check whether F and G are equivalent in practical fuzzy logic, we must find some *indirect* method.

5.2. Relationship with equivalence in logic programming leads to an algorithm

In Section 4, we have shown that two formulas F and G are equivalent in practical fuzzy logic iff they are equivalent in logic programming. This result provides us with a hope that equivalence can be algorithmically checked. Indeed, the whole idea of *logic programming* was to develop a *programming* (hence, *algorithmic*) language that would be able to process logical statements.

Equivalence in logic programming is indeed algorithmic. Indeed, we can check this equivalence as follows:

- For each of the formulas F and G, and for every set of literals W, we can use Definition 4.1 to check whether a formula is true for W, false for W, or neither true nor false for this W.
  There are totally $2n$ literals:
  - n variables $v_1, ..., v_n$, and
  - n negations of literals $\neg v_1, ..., \neg v_n$.

  Therefore, there are 2^{2n} possible sets of literals. We can (in principle) check them all. As a result of this checking, we will have for each of the formulas F and G, the list of all sets of literals W on which this formula is true.
- Next, for each of the two formulas F and G, we go through the list of all sets of literals W for which this formula is true, and eliminate all sets that have proper subsets in the same list. After this elimination, we have for each of the two formulas, the list of all its answer sets.

- If these lists coincide, then these formulas are equivalent. If they differ, then the formulas F and G are not equivalent.

Example 1. Let us check whether the formulas $F = v_1 \& \neg v_2$ and $G = \neg(\neg v_1 \vee v_2)$ are equivalent. Here, $n = 2$, so we have to check $2^{2n} = 2^4 = 16$ sets of literals. Let us list them all. First, we will list the sets with no elements; then, the sets with one element, etc.:

- set with no element: ϕ;
- sets with one element: $\{v_1\}$, $\{v_2\}$, $\{\neg v_1\}$, $\{\neg v_2\}$;
- sets with two elements: $\{v_1, v_2\}$, $\{v_1, \neg v_1\}$, $\{v_1, \neg v_2\}$, $\{v_2, \neg v_1\}$, $\{v_2, \neg v_2\}$, $\{\neg v_1, \neg v_2\}$;
- sets with three elements: $\{v_1, v_2, \neg v_1\}$, $\{v_1, v_2, \neg v_2\}$, $\{v_1, \neg v_1, \neg v_2\}$, $\{v_2, \neg v_1, \neg v_2\}$;
- a set with four elements: $\{v_1, v_2, \neg v_1, \neg v_2\}$.

For each set W, in order to check whether G is true for W, we must apply Definition 4.1 to all subformulas of G. Let us trace this procedure for $W = \{v_1, v_2\}$:

- v_1 is true for W, because $v_1 \in W$;
- v_2 is also true for W;
- since v_1 is true for W, $\neg v_1$ is false for W;
- since v_2 is true for W, $\neg v_1 \vee v_2$ is true for W;
- since $\neg v_1 \vee v_2$ is true for W, the formula $G = \neg(\neg v_1 \vee v_2)$ is false for W.

After we check all these sets, we will have only one set of variables on which each formula is true: $\{v_1, \neg v_2\}$. So, this very set will be the only answer set for both formulas F and G. So, for F and G, the classes of answer sets coincide therefore, F and G are equivalent.

Example 2. In the above example, F and G are equivalent in 2–valued logic, and they turned out to be equivalent for our case as well. Let us give another example in which formulas are equivalent in 2–valued logic but not in practical fuzzy logic: $F = v_1 \vee v_2$, and $G = v_1 \vee v_2 \vee (v_1 \& \neg v_1)$. In this case:

- F is true for the following sets of literals: $\{v_1\}$, $\{v_2\}$, $\{v_1, v_2\}$, $\{v_1, \neg v_2\}$, and $\{\neg v_1, v_2\}$. If we delete the proper supersets from this list, we will be left with only two answer sets: $\{v_1\}$ and $\{v_2\}$.
- G is true for the following sets of literals: $\{v_1\}$, $\{v_2\}$, $\{v_1, v_2\}$, $\{v_1, \neg v_2\}$, $\{\neg v_1, v_2\}$, and $\{v_1, \neg v_1\}$. If we delete the proper supersets from this list, we will be left with three answer sets: $\{v_1\}$, $\{v_2\}$, and $\{v_1, \neg v_1\}$.

The resulting classes of answer sets differ, so, F and G are not equivalent in logic programming (hence, not equivalent in practical fuzzy logic).

5.3. The need for a faster algorithm

The algorithm that we have just described is very slow, because it requires checking all 2^{2n} possible sets of literals. For $n = 4$, we must check $2^8 = 256$ cases; for $n = 10$, about a million cases. This is too long. Therefore, it would be nice to present a faster algorithm for checking whether two given formulas are equivalent. Let us present such an algorithm.

5.4. Second (faster) algorithm

This algorithm, like the one before, is related to our Theorem. The difference is that the previous algorithm has been based on the *formulation* of the theorem, and this new one is based on the *proof* of the theorem.

The main idea of this algorithm is as follows:

- first, we transform each of the formulas F and G to some simplified form, and
- then, we compare the results of these simplifications.

If the simplified formulas coincide, then F and G are equivalent in practical fuzzy logic; else, they are not.

The simplification of a formula A consists of two steps:

- First, if somewhere in the formula A, negation is applied to a composite subformula, i.e., if A has a subformula of one of the following types
  - $\neg(B\&C)$,
  - $\neg(A \vee C)$, or
  - $\neg(\neg B)$

  for some B and C), then we replace such a subformula by (correspondingly)
  - $\neg B \vee \neg C$,
  - $\neg B\&\neg C$, and
  - B.

  If after this procedure, we still have negations applied to composite formulas, we repeat this procedure again and again until we reach a formula in which negation are only applied to variables. After this, we move to the second step.
- On the second step, if the resulting formula has & applied to a composite formula that is more complicated than a literal, i.e., if it contains a subformula $A\&(B \vee C)$ for some A, B, and C, then we replace this subformula by $(A \vee B)\&(A \vee C)$. If after this replacement, we still have & applied to a non-literal, we repeat this procedure again and again. At the end, we get the formula in which & is applied only to literals, i.e., formula of the type $C_1 \vee ... \vee C_m$, where each C_j is an expression of the type $a\&...\&b$ with literals $a, ..., b$.
- Finally, we compare the resulting subformulas C_j. If the set of literals of one of them is a superset of the set of literals in another, then we delete the one with more literals in it. As a result, we get a *reduced* formula $C_1 \vee ... \vee C_m$.

In the Proof of Theorem 1, we have shown that this simplification replacements will eventually stop, and that to check whether F and G are equivalent, it is sufficient to check whether their resulting simplifications coincide (to be more precise, coincide modulo possible transpositions of literals and C_j). So, after we simplify F and G, we simply compare the simplifications.

Let us trace this algorithm on the above two examples.

Example 1. The formula $F = v_1\&\neg v_2$ is already in the simplified form, with $m = 1$. The formula $G = \neg(\neg v_1 \vee v_2)$ has negation applied to a composite formula. After an appropriate replacement, we get $\neg(\neg v_1)\&\neg v_2$. Here, again, we have a negation applied to a composite formula $\neg v_1$. After replacing $\neg(\neg v_1)$ by v_1, we finally get the simplified formula $v_1\&\neg v_2$. Further simplification is not needed. Since we got the same simplified form for both formulas, this means that F and G are indeed equivalent.

Example 2. The formulas $F = v_1 \vee v_2$ and $G = v_1 \vee v_2 \vee (v_1\&\neg v_1)$ are already in the simplified form, and they are different. So, F and G are not equivalent in practical fuzzy logic.

Comment. We can see from these examples that our second algorithm is indeed faster.

6. ANOTHER REASONABLE QUESTION: WHEN IS A FORMULA F A CONSEQUENCE OF A FORMULA G?

If formulas F and G are not equivalent, then a natural question to ask is: are these two formulas independent from each other, or one of them is a consequence of another one? In order to answer this question, we must first define what a consequence means. This question is the easiest to answer for 2-valued logic:

Definition 6.1. *In 2–valued logic, we say that F is a consequence of G if for every Boolean vector t for which $t(G)$ is true, $t(F)$ is also true.*

Comment. Since we identified 1 with "true", and 0 with "false", we can reformulate this definition as follows:

Definition 6.1′. *In 2–valued logic, we say that F is a consequence of G is for every Boolean vector t, $t(F) \geq t(G)$.*

Comments.
1. Indeed, if $t(G) = 0$, i.e., if G is false in t, then this inequality is always true. If, however, $t(G) = 1$, then this inequality implies that $t(F) = 1$. So, this modified definition is equivalent to the original one.

2. In traditional fuzzy logic, in addition to the cases when $t(G) = 0$ and $t(G) = 1$, we can have cases when $0 < t(G) < 1$. Intuitively, if we have some reasons to believe in G, and F is a consequence of G, then these same reasons can be also viewed as reasons to believe in F. There may be also other reasons to believe in F, that are unrelated to G. Therefore, if F is a consequence of G, this means that in all cases, we have either *the same* or even *more* reasons to believe in F than in G. In other words, the *degree of belief* in F must be always not smaller than the degree of belief in G: $t(F) \geq t(G)$. We can use this inequality as a definition of the notion of consequence in traditional (non-interval) fuzzy logic:

Definition 6.2. *In traditional fuzzy logic, we say that F is a consequence of G if for every initial truth value assignment t, we have $t(F) \geq t(G)$.*

Comment. How to define "consequence" for practical (interval) fuzzy logic? If the degree of belief in G is described by an interval $[t^-(G), t^+(G)]$, this means that the actual degree of belief can take any value from this interval. For each of these degrees of belief, the corresponding degree of belief in F is greater than the degree of belief in G. In other words, every possible degree of belief in F (i.e., every number from the interval $\mathbf{t}(F)$) must be not smaller than some possible degree of belief in G (i.e., some number from the interval $\mathbf{t}(F)$). It is natural to make this condition a definition of a consequence in the practical fuzzy case. We will describe this definition in two steps:

- First, we will describe the corresponding relation between the intervals.
- Second, we will use this relation to describe what a consequence means.

Definition 6.3. *We say that an interval* $\mathbf{a} = [a^-, a^+]$ *is not smaller than the interval* $\mathbf{b} = [b^-, b^+]$ *(and denote it by* $\mathbf{a} \geq \mathbf{b}$*) if the following two conditions are satisfied:*

- *for every value* $a \in \mathbf{a}$*, there exists a* $b \in \mathbf{b}$ *such that* $a \geq b$*; and*
- *for every* $b \in \mathbf{b}$*, there exists a* $a \in \mathbf{a}$ *such that* $a \geq b$*.*

PROPOSITION 3. $\mathbf{a} \geq \mathbf{b}$ *iff* $a^+ \geq b^+$ *and* $a^- \geq b^-$*.*

Comments.
1. This statement is easy to prove.

2. This partial order among intervals coincides with the order imposed by the lattice structure $(\min, \max)$ on the set of all intervals, and is, therefore, in good accordance with the original ideas of[8,25].

Definition 6.4. *In practical fuzzy logic, we say that* F *is a consequence of* G *if for every initial practical fuzzy truth assignment* $\mathbf{t}$*, we have* $\mathbf{t}(F) \geq \mathbf{t}(G)$*.*

Comment. Similarly to Theorem 1, we will show that this definition is equivalent to a similar definition from logic programming.

Definition 6.5. *In logic programming, we say that* F *is a consequence of* G *if* F *is true for every answer set of* G*.*

Comment. In Section 4, we have already described how to determine all answer sets of a given formula, and how to check whether a given formula F is true for a given set of literals. These two algorithms enable us to check whether F is a consequence of G in logic programming.

THEOREM 2. *Let* F *and* G *be any propositional formulas. Then,* F *is a consequence of* G *in practical fuzzy logic if and only if* F *is a consequence of* G *in logic programming.*

Comment. Because of this theorem, we can apply the above-described method of checking whether F is a consequence of G in logic programming to check whether F is a consequence of G in the sense of practical fuzzy logic.

7. PROOFS

7.1. Proof of Theorem 1

7.1.1. The structure of the proof

Our theorem is about arbitrary propositional formulas. To simplify its proof, we will first simplify the formulas. To be more precise, we will do two things:

- First, we will describe three transformations T of propositional formulas for which we will prove that an arbitrary formula A is equivalent to the result $T(A)$ of applying T to A (equivalent in both senses).
- Second, we will show that consequent application of such transformation will indeed simplify the original propositional formulas.

As a result, we will reduce the problem of checking equivalence for arbitrary propositional formulas to the problem of checking equivalence for simplified ones.

7.1.2. First transformation

Let us first consider transformations that corresponds to De Morgan's laws, and prove that they keep the formula equivalent in both senses. In logic programming, we will also prove that these transformations keep formulas equivalent in a somewhat stronger sense:

Definition 7.1. *We say that propositional formulas A and B are strongly equivalent in logic programming if for every set of literals W, the following two conditions are true:*

- *A is true in W iff B is true in W, and*
- *A is false in W iff B is false in W.*

The following Lemma shows that this definition is indeed stronger than equivalence in the sense of Definition 4.2:

LEMMA 1. *In logic programming, if formulas A and B are strongly equivalent, then they are equivalent.*

Proof of Lemma 1. According to Definition 4.2, formulas A and B are equivalent if they have exactly the same answer set. But if A and B are strongly equivalent, then, in particular, A and B are true or not true for exactly the same sets of literals W. Therefore, their answer sets, i.e., the *smallest* sets of literals W for which the formula is true, are exactly the same for A and for B. Q.E.D.

Let us now prove that De Morgan's transformation do lead to an equivalent (and in the logic programming case, even to a strongly equivalent) formula.

LEMMA 2_f. *In practical fuzzy logic, the following three equivalences hold:*

i) *$\neg(A \vee B)$ is equivalent to $(\neg A)\&(\neg B)$ for every two formulas A and B.*
ii) *$\neg(A\&B)$ is equivalent to $(\neg A) \vee (\neg B)$ for every two formulas A and B.*
iii) *$\neg(\neg A)$ is equivalent to A for every formula A.*

LEMMA 2_l. *In logic programming, the following three strong equivalences hold:*

i) *$\neg(A \vee B)$ is strongly equivalent to $(\neg A)\&(\neg B)$ for every two formulas A and B.*
ii) *$\neg(A\&B)$ is strongly equivalent to $(\neg A) \vee (\neg B)$ for every two formulas A and B.*
iii) *$\neg(\neg A)$ is strongly equivalent to A for every formula A.*

Proof of Lemma 2_f. We will proof part i) of this lemma. The other two statements are proved similarly. The proof of part i) follows from the following sequence of steps:

- According to Definition 3.6, to prove this equivalence, we need to show that the interval truth values $\mathbf{t}(\neg(A \vee B))$ and $\mathbf{t}((\neg A)\&(\neg B))$ assigned to these two formulas are the same for every initial practical fuzzy truth value assignment $\mathbf{t}$.
- According to Definition 3.5:
  - $\mathbf{t}(\neg(A \vee B)) = 1 - \min(\mathbf{t}(A), \mathbf{t}(B))$, and
  - $\mathbf{t}((\neg A)\&(\neg B)) = \max(1 - \mathbf{t}(A), 1 - \max(1 - \mathbf{t}(B))$.
- We will now use Proposition 1 to express operations with intervals in terms of operations with their bounds. To do that, we will need denotations for the bounds of $\mathbf{t}(A)$ and $\mathbf{t}(B)$:
  - We will denote the bounds of the interval $\mathbf{t}(A)$ by $[a^-, a^+]$, and

- We will denote the bounds of the interval $\mathbf{t}(B)$ by $[b^-,b^+]$.

In these terms, we have the following formulas:

- $\mathbf{t}(\neg(A\vee B))=[1-\max(a^+,b^+),1-\max(a^-,b^-)]$.
- $\mathbf{t}((\neg A)\&(\neg B))=[\min(1-a^+,1-b^+),\min(1-a^-,1-b^-)]$.

- The function $1-x$ is monotonically decreasing. Therefore, the value of $1-x$ is the smallest iff the value of x is the largest. So, $\min(1-a^+,1-b^+)=1-\max(a^+,b^+)$. Hence, the lower bounds of the intervals that we want to prove to be equal really coincide. Similarly, their upper bound coincide, so, the intervals are equal. Q.E.D.

Proof of Lemma 2_l. We will only prove part i) of this lemma; the other two cases can be proved similarly.

According to Definition 7.1, the desired equivalence means that the formulas $\neg(A\vee B)$ and $(\neg A)\&(\neg B)$ are true or false on exactly the same sets of literals.

The fact that A is true for W iff B is true for W, follows from the following sequence of equivalences (extracted from Definition 4.1):

- According to Definition 4.1 (case of $\neg$), $\neg(A\vee B)$ is true in W if and only if $A\vee B$ is false in W.
- This, in its turn, is (according to Definition of "true" for $\vee$) equivalent to both A and B being false in W.

On the other hand, for $(\neg A)\&(\neg B)$, Definition 4.1 leads to the following sequence of equivalences:

- By Definition 4.1 (part &), $(\neg A)\&(\neg B)$ is true for W iff both $\neg A$ and $\neg B$ are true for W.
- This, in its turn, is (according to the $\neg$ part of Definition 4.1) equivalent to both A and B being false in W.

So, the first of our two composite formulas $\neg(A\vee B)$ is true for W iff both A and B are false for W. Similarly, the second of our composite formulas is true for W iff both A and B are false in W. Therefore, the first composite formula is true for W iff the second composite formula is true for W.

A similar proof works for falseness: namely, each of the composite formulas is false for W iff either A if true for W, or B is true for W. Therefore, the first composite formula is false for W iff the second composite formula is false for W. Q.E.D.

Comment. We want to use these transformations to transform the original propositional formula A to some form that is simpler but still equivalent to the original formula A (equivalent both in the sense of practical fuzzy logic and in the sense of logic programming). It turns out that we can apply transformations not only to the formula itself, but to its subformulas as well.

LEMMA 3_f. *In practical fuzzy logic,*

- *If A is equivalent to A', then $\neg A$ is equivalent to $\neg A'$.*
- *If A is equivalent to A', and B is equivalent to B', then $A\vee B$ is equivalent to $A'\vee B'$.*
- *If A is equivalent to A', and B is equivalent to B', then $A\&B$ is equivalent to $A'\&B'$.*

LEMMA 3_l. *In logic programming,*

- *If A is strongly equivalent to A', then $\neg A$ is strongly equivalent to $\neg A'$.*
- *If A is strongly equivalent to A', and B is strongly equivalent to B', then $A \vee B$ is strongly equivalent to $A' \vee B'$.*
- *If A is strongly equivalent to A', and B is strongly equivalent to B', then $A \& B$ is strongly equivalent to $A' \& B'$.*

Proof of Lemma 3_f. The proof of this lemma directly follows from the definition of equivalence (Definition 3.6), and the definition of a truth value of a formula. For example, for $\vee$, since A is equivalent to A', we can conclude that $\mathbf{t}(A) = \mathbf{t}(A')$ for all initial truth value assignments $\mathbf{t}$. Similarly, $\mathbf{t}(B) = \mathbf{t}(B')$ for all initial truth value assignments $\mathbf{t}$. Therefore, $\mathbf{t}(A \vee B) = \max(\mathbf{t}(A), \mathbf{t}(B)) = \max(\mathbf{t}(A'), \mathbf{t}(B')) = \mathbf{t}(A' \vee B')$. Q.E.D.

Proof of Lemma 3_l. The proof of this lemma directly follows from the definition of strong equivalence (Definition 7.1). For example, let us consider the case of $\vee$. According to Definition 7.1, we must prove that $A \vee B$ and $A' \vee B'$ are true or false for exactly the same sets of literals. Let's prove it for "true". Indeed, according to Definition 4.1, $A \vee B$ is true for W iff either A, or B is true for W. But since A is strongly equivalent to A', A is true for W iff A' is true for W. Similarly, B is true for W iff B' is true for W. Therefore, $A \vee B$ is true for W iff either A' or B' is true for W. But this exactly when $A' \vee B'$ is true. So, $A \vee B$ is true for W iff $A' \vee B'$ is true for W. Similarly, we can prove that $A \vee B$ is false for W iff $A' \vee B'$ is false for W. So, $A \vee B$ and $A' \vee B'$ are strongly equivalent.

A similar straightforward proof can be proposed for two other cases. Q.E.D.

LEMMA 4_f. *Assume that we have a formula A, its subformula B, and a formula B' that is equivalent to B in the sense of practical fuzzy logic. Then, if we substitute B' into A instead of B, we will get a new formula A' that is equivalent to A.*

LEMMA 4_l. *Assume that we have a formula A, its subformula B, and a formula B' that is strongly equivalent to B in the sense of logic programming. Then, if we substitute B' into A instead of B, we will get a new formula A' that is strongly equivalent to A.*

Proof of Lemmas 4_f and 4_l. These results follow directly from Lemmas 3_f and 3_l, if we use induction over the length of the formula to reduce the problem to shorter A's and A''s. Q.E.D.

As a result of Lemmas 2–4, we can simplify each of the given propositional formulas F and G as follows.

LEMMA 5. *For every propositional formula A, there exists a propositional formula $T_1(A)$ with the following three properties:*

- *in $T_1(A)$, negation is applied only to variables:*
- *$T_1(A)$ is equivalent to A in practical fuzzy logic; and*
- *$T_1(A)$ is strongly equivalent to A in logic programming.*

Proof of Lemma 5. First, let us describe the desired simplification procedure. If a formula A contains negation applied to a composite formula, then let us take the longest composite formula to which negation is applied, and move this negation "inside" the formula using one of the following transformations:

- If negation is applied to a formula $A \vee B$ for some A and B, then we replace $\neg(A \vee B)$ by $(\neg A)\&(\neg B)$.
- If negation is applied to a formula $A\&B$ for some A and B, then we replace $\neg(A\&B)$ by $(\neg A) \vee (\neg B)$.
- If negation is applied to a formula $\neg A$ for some A, then we replace $\neg(\neg A)$ by A.

If there are several longest formulas, then we choose one of them. In each case, according to Lemmas 2–3, the transformed subformula has the following two properties:

- it is equivalent to the result of its transformation in practical fuzzy logic, and
- it is strongly equivalent to the result of its transformation in logic programming.

Therefore, due to Lemmas 4, the formula in which we made this replacement is equivalent to the original formula in the same two senses.

If the resulting transformed formula still contains negation applied to a composite formula, then we repeat this transformation once again, etc. Let us show that these iterations will stop. Indeed, let us denote by L, the largest length of the subformula to which negation is applied. Suppose that initially, there were $n(L)$ negated formulas of this maximal length L, and that we applied this transformation to an expression $\neg A$ for some subformula A of length L. After such a transformation, negation is applied to one or two formulas of smaller length. So, in the resulting formula, the maximal length of the negated subformula is still $\leq L$, and the total number of negated subformulas of length L has decreased by one. Therefore, after $\leq n(L)$ steps, we will eliminate all negated formulas of length L. After that, if there still is a negated composite formula, its length is $< L$. Therefore, the largest of the lengths of negated composite formulas is $\leq L - 1$.

If we repeat this iterative procedure again, we can transform our formula into a new one in which the negated composite subformulas have lengths $\leq L - 2$, then $\leq L - 3$, etc. When we bring this length to 1, this means that we have an equivalent formula in which negation is *not* applied to any composite formula, and is, therefore, applied only to the variables. This is the desired $T_1(A)$. Q.E.D.

Example. Suppose that the formula A is $\neg(v_1\&\neg(v_2\&\neg v_3))$. Let us describe our transformation from A to $T_1(A)$ step by step:

- In the original formula A, there are two negated composite subformulas: $v_2\&\neg v_3$ and $v_1\&\neg(v_2\&\neg v_3)$. The second negated subformula is longer, so we start our transformation with this longer subformula. As a result, we get the formula $\neg v_1 \vee \neg(\neg(v_2\&\neg v_3))$.
- In this new formula, the longest negated subformula is $\neg(v_2\&\neg v_3)$. So:
  - We apply the appropriate transformation to the negation $\neg(\neg(v_2\&\neg v_3))$ of this subformula, and get the equivalent subformula $v_2\&\neg v_3$.
  - Hence, if we substitute $v_2\&\neg v_3$ instead of $\neg(\neg(v_2\&\neg v_3))$ into the formula $\neg v_1 \vee \neg(\neg(v_2\&\neg v_3))$ that we have obtained after the first transformation step, then we get (due to Lemmas 4) an equivalent formula $\neg v_1 \vee (v_2\&\neg v_3)$.

In this resulting formula $\neg v_1 \vee (v_2\&\neg v_3)$, negation is applied only to variables, so this formula is the desired $T_1(A)$.

7.1.3. Second transformation step

Next, we will consider transformations that correspond to distributivity law.

LEMMA 6_f. *In practical fuzzy logic, $A\&(B \vee C)$ is equivalent to $(A\&B) \vee (A\&C)$ for arbitrary three formulas A, B, and C.*

LEMMA 6_l. *In logic programming, $A\&(B \vee C)$ is strongly equivalent to $(A\&B) \vee (A\&C)$ for arbitrary three formulas A, B, and C.*

Proof of Lemma 6_f. This proof follows from the following sequence of steps.

- According to Definition 3.6, to prove this equivalence, we need to show that the interval truth values $\mathbf{t}(A\&(B \vee C))$ and $\mathbf{t}((A\&B) \vee (A\&C))$ assigned to these two formulas are the same for every initial practical fuzzy truth value assignment $\mathbf{t}$.
- Let us use Definition 3.5 and Proposition 1 to describe the intervals assigned to these formulas. To do that, we will need denotations for the bounds of the intervals $\mathbf{t}(A)$, $\mathbf{t}(B)$, and $\mathbf{t}(C)$:
  - We will denote the bounds of the interval $\mathbf{t}(A)$ by $[a^-, a^+]$.
  - We will denote the bounds of the interval $\mathbf{t}(B)$ by $[b^-, b^+]$.
  - We will denote the bounds of the interval $\mathbf{t}(C)$ by $[c^-, c^+]$.

  In these terms, we have the following formulas:
  - $\mathbf{t}(B \vee C)) = \max(\mathbf{t}(B), \mathbf{t}(C)) = [\max(b^-, c^-), \max(b^+, c^+)]$.
  - $\mathbf{t}(A\&(B \vee C)) = \min(\mathbf{t}(A), \mathbf{t}(B \vee C)) =$ $[\min(a^-, \max(b^-, c^-)), \min(a^+, \max(b^+, c^+))]$.
  - $\mathbf{t}(A\&B)) = \min(\mathbf{t}(A), \mathbf{t}(B)) = [\min(a^-, b^-), \min(a^+, b^+)]$.
  - $\mathbf{t}(A\&C)) = \min(\mathbf{t}(A), \mathbf{t}(C)) = [\min(a^-, c^-), \min(a^+, c^+)]$.
  - $\mathbf{t}((A\&B) \vee (A\&C)) = \max(\mathbf{t}(A\&B), \mathbf{t}(A\&C)) =$ $[\max(\min(a^-, b^-), \min(a^-, c^-)), \max(\min(a^+, b^+), \min(a^+, c^+))]$.
- So, to prove that these intervals coincide, it is sufficient to prove the equality $\min(a, \max(b, c)) = \max(\min(a, b), \min(a, c))$ for all a, b, and c. Then:
  - applying this equality to a^-, b^-, and c^-, we will prove that the lower bounds coincide, and
  - applying it for a^+, b^+, and c^+, we will conclude that their upper bounds coincide.

  So, let us prove the desired equality. We will prove it by considering all possible orderings of a, b, and c:
  - If $a \le b$ and $a \le c$, then $a \le \max(b, c)$, and therefore, $\min(a, \max(b, c)) = a$. On the other hand, $\min(a, b) = a$, $\min(a, c) = a$, and thus, $\max(\min(a, b), \min(a, c)) = \max(a, a) = a$. So, in this case, the desired equality is true.
  - If $b \le a \le c$, then $\max(b, c) = c \ge a$, therefore, $\min(a, \max(b, c)) = a$. On the other hand, $\min(a, b) = b$, $\min(a, c) = c$, and so $\max(\min(a, b), \min(a, c)) = \max(a, b) = a$. In this case, the desired equality is also true.
  - If $c \le a \le b$, then this inequality is proved similarly.
  - If $b \le c < a$, then $\max(b, c) = c$ and $\min(a, \max(b, c)) = c$. On the other hand, $\min(a, b) = b$, $\min(a, c) = c$, and $\max(\min(a, b), \min(a, c)) = \max(b, c) = c$. The equality is thus true.

- The only remaining case $c < b < a$ is treated similarly.

The equality is proved, so the intervals that correspond to the two composite formulas do coincide. Q.E.D.

Proof of Lemma 6_l. Applying Definition 4.1 twice, we can easily conclude that $A\&(B \vee C)$ is true for W iff the following two statements is true:

- First, A is true for W.
- Second, either B is true for W, or C is true for W.

Similarly, we can conclude that $(A\&B) \vee (A\&C)$ is true for W iff either A and B are both true for W, or A and C are both true for W.

One can see that these conditions are equivalent. Therefore, $A\&(B \vee C)$ is true for W iff $(A\&B) \vee (A\&C)$ is true for W.

Similarly, we can prove that $A\&(B \vee C)$ is false for W iff $(A\&B) \vee (A\&C)$ is false for W. The strong equivalence is proved. Q.E.D.

Comment. Lemmas 6 lead to a possibility of a new transformation: $A\&(B \vee C)$ into $(A\&B) \vee (A\&C)$. Let us show how this transformation can simplify the original propositional formula.

Definition 7.2.

- *By a conjunction C, we mean a propositional formula of the type $a\&...\&b$, where $a, ..., b$ are literals (i.e., variables or their negations).*
- *By a formula in a disjunctive normal form, or in DNF, for short, we mean a propositional formula of the type $C_1 \vee ... \vee C_m$, where C_j are conjunctions.*

LEMMA 7. *For every propositional formula A, there exists a propositional formula $T_2(A)$ with the following three properties:*

- *$T_2(A)$ is a formula in DNF;*
- *$T_2(A)$ is equivalent to A in practical fuzzy logic; and*
- *$T_2(A)$ is strongly equivalent to A in logic programming.*

Proof of Lemma 7. To get $T_2(A)$, we do the following:

- First, we apply a procedure from Lemma 5 to the formula A and get an equivalent formula $T_1(A)$ in which negations are only applied to variables.
- Then, we process $T_1(A)$ in a manner similar to Lemma 5:
    - Namely, if in $T_1(A)$, & is applied to a subformula that is more complicated than a literal, i.e., if $T_1(A)$ contains a subformula of the type $A\&(B \vee C)$, then we replace this subformula by an equivalent formula $(A\&B) \vee (A\&C)$. Due to Lemmas 4, the result of this replacement is equivalent to $T_1(A)$.
    - If in this result, we still have & applied to a composite formula, we repeat this procedure again, etc.

Similarly to the proof of Lemma 5, we can proof that these iterations will eventually stop. In the resulting formula $T_2(A)$, & can be only applied to literals. Therefore, $T_2(A)$ is in DNF. Q.E.D.

7.1.4. Third transformation

A formula in DNF can still be simplified:

- first, we will show that if a conjunction contains two identical literals, then we can delete one of them (i.e., reduce $v_1\&v_1$ to v_1);
- second, we will show that if one conjunction is a, so to say, "superset" of another one, then this larger conjunction can also be deleted; e.g., a formula $v_1 \vee (v_1\&\neg v_2)$ can be reduced to a formula v_1.

Let us describe the corresponding transformations in detail.

LEMMA 8_f. *In practical fuzzy logic, $A\&A$ is equivalent to A for an arbitrary formula A.*

LEMMA 8_l. *In logic programming, $A\&A$ is strongly equivalent to A for an arbitrary formula A.*

Proof of Lemma 8_f. If we denote $\mathbf{t}(A)$ by $[a^-, a^+]$, then $\mathbf{t}(A\&A) = \min(\mathbf{t}(A), \mathbf{t}(A)) = [\min(a^-, a^-), \min(a^+, a^+)] = [a^-, a^+] = \mathbf{t}(A)$. Q.E.D.

Proof of Lemma 8_l. According to Definition 4.1, $A\&A$ is true for W iff A is true for W *and* A is true for W. In other words, $A\&A$ is true for W iff A is true for W. Similarly, for false. Q.E.D.

LEMMA 9_f. *In practical fuzzy logic, $A \vee (A\&B)$ is equivalent to A for arbitrary formulas A and B.*

LEMMA 9_l. *In logic programming, $A \vee (A\&B)$ is strongly equivalent to A for arbitrary formulas A and B.*

Proof of Lemma 9_f. This proof will be similar to the proofs of Lemmas 2_f and 6_f. If we denote $\mathbf{t}(A)$ by $[a^-, a^+]$, and $\mathbf{t}(B)$ by $[b^-, b^+]$,
then we get $\mathbf{t}(A\&B) = [\min(a^-, b^-), \min(a^+, b^+)]$, and

$$\mathbf{t}(A \vee (A\&B)) = \max(a^-, \min(a^-, b^-)), \max(a^+, \min(a^+, b^+)).$$

But for every a and b, $a \geq \min(a, b)$, and therefore, $\min(a, \max(a, b)) = a$. In particular, $\max(a^-, \min(a^-, b^-)) = a^-$ and $\max(a^+, \min(a^+, b^+)) = a^+$, hence, $\mathbf{t}(A \vee (A\&B)) = \mathbf{t}(A)$. This means that these formulas are indeed equivalent. Q.E.D.

Proof of Lemma 9_l. Let us first prove that $A \vee (A\&B)$ is true for W iff A is true for W. Indeed, according to Definition 4.1, $A \vee (A\&B))$ is true for W if one of the following two conditions is true:

- A is true for W;
- A is true for W and B is true for W.

If A is true for W, then the first condition is satisfied. If any of these conditions is satisfied, then A is true for W. Therefore, $A \vee (A\&B))$ is true for W iff A is true for A.

Similarly, it can be proved that $A \vee (A\&B))$ is false for W iff A is false for A. So, these two formulas are strongly equivalent. Q.E.D.

Comment. The resulting transformation enables us to do a further reduction of the original formula.

Definition 7.3.

- *We say that a conjunction $C = a\&...\&b$ is irreducible if all its literals are different.*
- *We say that two irreducible conjunctions C and C' coincide, and denote it by $C = C'$, if they contain exactly the same literals (in other words, if C can be obtained from C' by a transposition of literals).*
- *We say that an irreducible conjunction C is a superset of an irreducible conjunction C' if the set of literals of C is a (proper) superset of the of literals of C'.*
- *We say that a formula in DNF is irreducible if the following two conditions are satisfied:*
  - *First, each of its conjunctions is irreducible.*
  - *Second, none of its conjunctions is a superset of another one.*
- *We say that two irreducible DNF formulas A and A' coincide (and denote it $A = A'$) if they contain the same number of conjunctions, and these conjunctions can be divided into coinciding pairs (in other words, if A can be obtained from A' after an appropriate transposition of conjunctions and transpositions of literals inside each conjunction).*

LEMMA 10. *For every propositional formula A, there exists an irreducible DNF formula $T_3(A)$ that is equivalent to A in practical fuzzy logic, and that is strongly equivalent to A in logic programming.*

Proof of Lemma 10. First, we apply Lemma 7 to reduce A to its DNF form $T_2(A)$. Then, we consequently apply two reduction procedures:

- First, we make each conjunction irreducible. If one of the conjunctions contains a repeating literal a, then we reduce $a\&a$ to a. On each step, we reduce the total length of the formula. Therefore, this procedure will eventually stop. When we stop, this means that all our conjunctions are irreducible.
- Next, if one conjunction is a superset of another one, then we (based on Lemmas 9) eliminate the superset one. Again, each such elimination reduces the total length of the formula, so sooner or later, we will stop.

At the end of this procedure, we have the desired irreducible DNF that is equivalent to the original formula A. Q.E.D.

7.1.5. Final proof

Now, we are ready to prove the main theorem. Suppose that have two formulas F and G, and we know that in practical fuzzy logic, F is equivalent to G. According to Lemma 10, in practical fuzzy logic, F is equivalent to $T_3(F)$, and G is equivalent to $T_3(G)$. Therefore, $T_3(F)$ is equivalent to $T_3(G)$ in practical fuzzy logic. If we will able to prove that $T_3(F)$ is also equivalent to $T_3(G)$ in logic programming, then, due to the fact that in logic programming, F is equivalent to $T_3(F)$, and G is equivalent to $T_3(G)$, we will be able to conclude that $F \sim T_3(F) \sim T_3(G) \sim G$, and $F \sim G$.

Similar idea can work if we start with the assumption that F and G are equivalent in logic programming.

So, to complete our proof, it is sufficient to show that irreducible DNF formulas are equivalent in practical fuzzy logic iff they are equivalent in logic programming. Let us prove it.

LEMMA 11. *Let A and B be irreducible formulas. Then, the following three conditions are equivalent to each other:*
i) *$A = B$;*
ii) *A is equivalent to B is practical fuzzy logic;*
iii) *A is equivalent to B in logic programming.*

Proof of Lemma 11. This proof will consist of four parts:
- First, we will prove that i)$\to$ii);
- Second, we will prove that ii)$\to$i);
- Third, we will prove that i)$\to$iii);
- Finally, we will prove that iii)$\to$i).

- *Let us first prove that i)$\to$ii).* If $A = B$, then B can be obtained from A simply by transpositions. Transpositions do not change the truth values assigned to a formula. Hence, $\mathbf{t}(A) = \mathbf{t}(B)$, and so, formulas A and B are indeed equivalent in practical fuzzy logic.

- *Let us now prove that ii)$\to$i).* Suppose that irreducible formulas A and B are equivalent to each other in practical fuzzy logic. Let us prove by reduction to a contradiction that $A = B$.

  Indeed, assume that $A \neq B$. This means that there exists a conjunction that belongs to one of the formulas A, B, but that does not belong to another formula. If there are several such conjunctions, let us take the shortest of them, and denote it by C. There are two possibilities:
  - C belongs to A but does not belong to B;
  - C belongs to B and does not belong to A.

  Without losing any generality, let us consider the first case (the second case will lead to a similar contradiction). We assumed that A and B are equivalent in practical fuzzy logic. This means that for every initial truth value assignment $\mathbf{t}$, we have $\mathbf{t}(A) = \mathbf{t}(B)$. In particular, the upper bounds $t^+(A)$ and $t^+(B)$ of these interval must coincide. Let us take the following initial truth assignment:
  - If a variable v_i is one of the literals from C, and its negation $\neg v_i$ does not belong to C, then we take $\mathbf{t}(v_i) = [1, 1]$.
  - If a literal $\neg v_i$ belongs to C, an v_i does not belong to C, we take $\mathbf{t}(v_i) = [0, 0]$.
  - If both the variable v_i and its negation $\neg v_i$ belong to C, we take $\mathbf{t}(v_i) = [0, 1]$.
  - If neither the variable v_i, nor its negation $\neg v_i$ belong to C, we take $\mathbf{t}(v_i) = [0.5, 0.5]$.

  The idea behind this assignment is that we want to make C true, and all other conjunctions unknown (with degree of belief 0.5). Let us show how this arrangement works. First, let us compute the upper bound $t^+(A)$ of the interval $\mathbf{t}(A)$ assigned to the formula A.

- Because of our definition, for every literal a from C, we have $[t^-(a), t^+(a)] = \mathbf{t}(a)$ equal either to $[1,1]$, or to $[0,1]$. In both cases, the upper bound $t^+(a)$ is equal to 1.
- According to Proposition 1, the upper bound $t^+(C)$ of the interval $\mathbf{t}(C) = \mathbf{t}(a\&...\&b)$ is equal to $\min(t^+(a), ..., t^+(b))$ for all literals $a, ..., b$ that form the conjunction C. Since for all these literals, $t^+(a) = ... = t^+(b) = 1$, we have $t^+(C) = 1$.
- Now, according to Proposition 1, the upper bound $t^+(A)$ of the interval $\mathbf{t}(A) = \mathbf{t}(C_1 \vee ... \vee C \vee ... \vee C_m)$ assigned to the formula A is equal to $t^+(A) = \max(t^+(C_1), ..., t^+(C), ..., t^+(C_m))$. Since all these upper bounds are ≤ 1, and one of the them (namely, $t^+(C)$) has been proven to be equal to 1, we can conclude that their maximum is equal to 1, i.e., that $t^+(A) = 1$.

Let us now arrive at a contradiction. This will be done in several steps:

- Since we assumed that A and B are equivalent in practical fuzzy logic, we conclude that $t^+(B) = 1$.
- The formula B is in DNF: $B = C_1 \vee ... \vee C_m$ for some conjunctions C_j. So, according to Proposition 1, we have $t^+(B) = \max(t^+(C_1), ..., t^+(C_m))$. Hence, the fact that $t^+(B) = 1$ means that the largest of the values $t^+(C_j)$ is equal to 1. In other words, for one of the conjunctions C_j from the formula B, we have $t^+(C_j) = 1$.
- Let us denote by $a, ..., b$ literals that form the conjunction C_j. Then, C_j is $a\&...\&b$ and therefore (according to Proposition 1), $t^+(C_j) = \min(t^+(a), ..., t^+(b))$. From the fact that $t^+(C_j) = 1$, we can thus conclude that $t^+(a) = ... = t^+(b) = 1$ for all literals from C_j.
- According to our choice of the initial truth assignment, $t^+(a) = 1$ iff a belongs to C (else, $t^+(a) = 0$ or $t^+(a) = 0.5$). So, the fact that $t^+(a) = 1$ for all a from C_j means that all literals from C_j are at the same time literals from C. In other words, either C coincides with C_j, or C is a superset of C_j. Let us how that in both cases, we get a contradiction.
- If C_j coincides with C, then C is a conjunction in *both* formulas A and B, and we have chosen C as a conjunction that belongs to A but not to B. So, this case is impossible.
- If C is a superset of C_j, then C_j is shorter than C. Since we have chosen C to be the shortest of all conjunctions that belongs to only one of the formulas A and B, and C_j is even shorter, this means that C_j cannot be from this class. In other words, the conjunction C_j must belong to both formulas A and B. So, the formula A contains both the conjunction C_j and its superset C. But we have assumed that A is irreducible and therefore, it cannot contain two conjunctions, one of which is a superset of another. So, in the second case, we also get a contradiction.

So, in both cases, our assumption that $A \neq B$ leads to a contradiction. Therefore, $A = B$.

- *Let us now prove that i)→iii).* If $A = B$, then evidently A is true for W iff B is true for W, and similarly, for false. So, A and B are strongly equivalent in logic programming and therefore (due to Lemma 1) they are equivalent in the sense of logic programming.
- *Finally, let us prove that iii)→i).* Suppose that A and B are equivalent in logic programming. This means that they have the same class of answer sets. To prove that $A = B$, let us show that an irreducible formula $A = C_1 \vee ... \vee C_m$ has exactly m answer sets that correspond to C_j: namely, for every $C = a\&..\&b$, the corresponding answer set is $\{a, ..., b\}$. To find out all answer sets, let us describe step by step when A is true for a given set of literals W:
  - According to Definition 3.6, A is true for W iff one of the formulas C_j is true for W.
  - A formula $C_j = a\&...\&b$, in its turn, is true for W iff all its literals a, ..., b are true for W.
  - Finally, a literal is true for W iff it belongs to W.

  Combining these definitions, we conclude that A is true for W iff W contains all the literals from one of the conjunctions C_j of the formula A. We can make two conclusions from here:
  - First, that for every j from 1 to m, A is true for the set W_j of all literals from the $j-$th conjunction C_j.
  - Second, that if A is true for W, then W contains W_j for some j.

  An answer set is defined as a set W such that A is true for W but not for any proper subset of W. So, if W is an answer set, then A must be true for W and therefore, W must be a superset of one of the sets W_j. W cannot be a proper superset, because else for W, there would exist a proper subset (namely, W_j) for which A is also true. So, only sets W_j can be answer sets. Let us show that they all *are* answer sets. Indeed, the only possibility for W_j *not* to be an answer set for A is if A would have a proper subset W for which A is also true. But for W to be such a subset, it must contain W_k for some k. So, $W_j \supset W \supseteq W_k$, hence $W_j \supset W_k$. This inclusion contradicts irreducibility of A.

  So, the class of answer sets of A coincides with $W_1, ..., W_m$. Therefore, if we know the answer sets, we can easily reconstruct C_j and thus, the formula A. Therefore, if two formulas A and B has the same class of answer sets, this means that they consist of exactly the same conjunctions, i.e., that $A = B$.

Comment. As we have already mentioned, proving Lemma 11 completes the proof of our Theorem. Q.E.D.

7.2. Proof of Proposition 2

The first part of this proposition immediately follows from Definition 3.6, if we take degenerate intervals $\mathbf{t}(v_i) = [t(v_i), t(v_i)]$. The desired example in the second part is provided by the following two formulas: $F = v_1 \vee \neg v_1$, and $G = v_1 \vee \neg v_1 \vee (v_2 \& \neg v_2)$. In practical fuzzy logic, $t(F) = \max(t(v_1), 1 - t(v_1))$ and

$$t(G) = \max[t(F), \min(t(v_2), 1 - t(v_2))].$$

If $t(v_1) \geq 0.5$, then $t(F) \geq t(v_1) \geq 0.5$. If $t(v_1) < 0.5$, then $1 - t(v_1) > 0.5$ and therefore, $t(F) \geq 1 - t(v_1) > 0.5$. So, in both possible cases, $t(F) \geq 0.5$.

Similarly, one can easily prove that in all possible cases, $\min(t(v_2), 1 - t(v_2)) \leq 0.5$. Therefore, $\min(t(t(v_2), 1 - t(v_2)) \leq 0.5 \leq t(F)$, and

$$t(G) = \max[t(F), \min(t(v_2), 1 - t(v_2))] = t(F).$$

So, $t(F) = t(G)$ for all t.

On the other hand, the formulas F and G are already in the irreducible DNF form, so from the fact that F and G are different, we can (using Lemma 11 from the Proof of Theorem 1) conclude that F is not equivalent to G in practical fuzzy logic. Q.E.D.

7.3. Proof of Theorem 2

Similarly to the proof of Theorem 1, since every formulas can be reduced to an irreducible one by equivalent transformations, it is sufficient to prove the desired result for irreducible formulas F and G.

So, let us assume that F and G are both irreducible. Then, $F = C_1 \vee \ldots \vee C_n$, and G is $C'_1 \vee \ldots \vee C'_m$ for some m and n, where C_i and C'_j are conjunctions. We have already proved in the proof of Theorem 1 (to be more precise, in the proof of its Lemma 11), that an irreducible formula G has exactly m answer sets $W'_1, \ldots, W'_m$, where W'_j is a set of all literals from the conjunction C'_j. We have also proved that F is true in W iff at least one of the conjunctions from F consists of literals from W. So, for irreducible formulas F and G, the fact that F is a consequence of G in logic programming, i.e., that F is true for all answer sets of G, can be reformulated as follows:

For every conjunction C'_i from G, there exists a conjunction C_j from F such that C'_i is a superset of C_i.

Let us show that this condition is indeed equivalent to F being a consequence of G in the sense of practical fuzzy logic.

- First, let us prove that if the logic programming condition is satisfied, then F is a consequence of G is the sense of logic programming, i.e., that $\mathbf{t}(F) \geq \mathbf{t}(G)$. Let us prove that $t^+(F) \geq t^+(G)$ (for t^-, the proof is similar). Indeed, since $G = C'_1 \vee \ldots \vee C'_m$, we have $t^+(G) = \max(t^+(C'_1), \ldots, t^+(C'_m))$. For every i from 1 to m, there exists a conjunction C_j from F that is a subset of C'_i. Therefore, we have one of the two cases:
  - either $C'_i = C_j$, or
  - $C'_i = C_j \& a \& \ldots \& b$ for some literals a, ..., b.

  In both cases, we can get some bounds on $t^+(C'_i)$:
  - in the first case, $t^+(C'_i) = t^+(C_j)$;
  - in the second case, $t^+(C'_i) = \min(t^+(C_j), t^+(a), \ldots) \leq t^+(C_j)$.

  So, in both cases, $t^+(C'_i) \leq t^+(C_j)$. Since $F = C_1 \vee \ldots \vee C_n$, we have $t^+(F) = \max(t^+(C_1), \ldots, t^+(C_n))$, and therefore, $t^+(C_j) \leq t^+(F)$. So, for every i from 1 to m, $t^+(C'_i) \leq t^+(C_j) \leq t^+(F)$. Hence, $t^+(G) = \max(t^+(C'_1), \ldots, t^+(C'_m)) \leq t^+(F)$.

- Let us now prove that if $\mathbf{t}(F) \geq \mathbf{t}(G)$ for all initial practical fuzzy truth value assignments $\mathbf{t}$, then the logic programming condition holds, i.e., for every conjunction C'_i from G, there exists a conjunction C_i from F for which C'_i is a superset of C_j. To prove it, let us fix i, and for that conjunction C'_i, let us choose the same truth assignment that we have used in the proof of Lemma 11 (from the proof of Theorem 1):
    - if a variable v_k belongs to C'_i, and its negation $\neg v_k$ does not belong to C'_i, then we take $\mathbf{t}(v_k) = [1,1]$.
    - if a literal $\neg v_k$ belongs to C'_i, and its negation v_k does not belong to C'_i, then we take $\mathbf{t}(v_k) = [0,0]$.
    - if both a variable v_k and its negation $\neg v_k$ belong to C'_i, then we take $\mathbf{t}(v_k) = [0,1]$.
    - if neither a variable v_k, nor its negation $\neg v_k$ belong to C'_i, then we take $\mathbf{t}(v_k) = [0.5, 0.5]$.

  With this choice, as in the proof of Lemma 11, $t^+(C'_i) = 1$, and therefore, $t^+(G) = 1$. Since F is a consequence of G in practical fuzzy logic, we can conclude that $t^+(F) \geq t^+(F) = 1$. Since t^+ is a number from the interval $[0,1]$, we can thus conclude that $t^+(F) = 1$. Since $t^+(F)$ is the maximum of n numbers $t^+(C_j)$, one of these n numbers must be equal to 1. Let us assume that $t^(C_j) = 1$. Since $C_j = a\&...\&b$, we have $t^+(C_j) = \min(t^+(a), ..., t^+(b))$. The minimum of several numbers $t^+(a)$, ..., $t^+(b)$ is equal to 1. Therefore, each of these numbers is equal to 1. So, $t^+(a) = 1$ for all literals from the conjunction C_j. But according to our choice of $\mathbf{t}$, $t^+(a) = 1$ iff a belongs to C'_i (else, $t^+(a) = 0$ or $= 0.5$). Hence, we can conclude that every literal from C_j belongs to C'_i, i.e., that C'_i is indeed a superset of C_j. Q.E.D.

Acknowledgments. This work was partially supported by NSF grants No. CDA-9015006 and No. EEC-9322370, and by a NASA grant No. NAG 9-757. The authors are thankful to Michael Gelfond, Slava Nesterov, Daniel G. Schwartz, Ronald Yager, and to the anonymous referees for their important attention and help.

REFERENCES

1. H. R. Berenji, "Fuzzy logic controllers", In: *An Introduction to Fuzzy Logic Applications in Intelligent Systems* (R. R. Yager, L. A. Zadeh. eds.), Kluwer Academic Publ., 1991.

2. D. Dubois and H. Prade, *Fuzzy sets and systems: theory and applications*, Academic Press, N.Y., London, 1980.

3. D. Dubois and H. Prade, *Possibility theory. An approach to computerized processing of uncertainty*, Plenum Press, N.Y. and London, 1988.

4. C. Elkan, "The paradoxical success of fuzzy logic", Proceedings of AAAI-93, American Association for Artificial Intelligence 1993 Conference, 698–703 (reprinted in *IEEE Expert. Intelligent Systems and Their Applications*, August 1994, 3–8).

5. C. Elkan et al, "Fuzzy Logic Symposium", *IEEE Expert*, August 1994.

6. B. R. Gaines, "Precise past, fuzzy future", *Intl. J. of Man-Machine Studies*, **19**, 117–134 (1983).

7. M. Gelfond and H. Przymusinska, "Definitions in epistemic specifications", In: A. Nerode, Wiktor Marek, V. S. Subrahmanian, *Logic Programming and Non-Monotonic Reasoning*, Proc. of the 1st Intl. Workshop, 1991, pp. 245–259.

8. J. A. Goguen, "The logic of inexact reasoning", *Synthese*, **19**, 325–373 (1969); reprinted in D. Dubois, H. Prade, and R. Yager (eds.), *Reading in Fuzzy Sets for Intelligent Systems*, Morgan Kaufmann, San Mateo, CA, 1994, 417–441.

9. I. R. Goodman and H. T. Nguyen, *Uncertainty models for knowledge-based systems*, North Holland, Amsterdam, 1985.

10. H. M. Hersch and A. Caramazza, "A fuzzy-set approach to modifiers and vagueness in natural languages", *J. Exp. Psychol.: General*, **105**, 254–276 (1976).

11. A. Kandel and G. Langholtz (Eds.), *Fuzzy Control Systems*, CRC Press, Boca Raton, FL, 1994.

12. O. Kosheleva and V. Kreinovich, "One more potential application of symbolic computations: interval logic", Abstracts of the International Conference on Interval and Computer-Algebraic Methods in Science and Engineering (Interval'94), St. Petersburg, Russia, March 7–10, 1994, 143–146.

13. B. Kosko, "Fuzziness vs. probability", *Intl. J. of General Systems*, **17**, No. 2–3, 211–240 (1990).

14. V. Kreinovich et al, "What non-linearity to choose? Mathematical foundations of fuzzy control,", Proceedings of the 1992 International Conference on Fuzzy Systems and Intelligent Control, Louisville, KY, 1992, 349–412.

15. C.-C. Lee, "Fuzzy logic in control systems: fuzzy logic controller", *IEEE Transactions on Systems, Man and Cybernetics*, **20** (2), 404–435 (1990).

16. H. T. Nguyen, "A note on the extension principle for fuzzy sets", *J. Math. Anal. and Appl.*, **64**, 359–380 (1978).

17. H. T. Nguyen, "Some mathematical tools for linguistic probabilities", *Fuzzy Sets and Systems*, **2**, 53–65 (1979).

18. H. T. Nguyen, *Lecture Notes on Fuzzy Logic*, LIFE, Tokyo Institute of Technology, 1993.

19. H. T. Nguyen and V. Kreinovich, "On Logical Equivalence in Fuzzy Logic", Sixth International Fuzzy Systems Association World Congress, San Paulo, Brazil, July 22–28, 1995 (to appear).

20. H. T. Nguyen, V. Kreinovich, and D. Tolbert, "A measure of average sensitivity for fuzzy logics", *International Journal on Uncertainty, Fuzziness, and Knowledge-Based Systems*, **2**, No. 4 361–375 (1994).

21. H. T. Nguyen, M. Sugeno, R. Tong, and R. Yager (eds.), *Theoretical aspects of fuzzy control*, J. Wiley, N.Y., 1995.

22. G. C. Oden, "Integration of fuzzy logical information", *Journal of Experimental Psychology: Human Perception Perform.*, **3** (4), 565–575 (1977).

23. A. Ramer and V. Kreinovich, "Information complexity and fuzzy control", Chapter 4 in: A. Kandel and G. Langholtz (Eds.), *Fuzzy Control Systems*, CRC Press, Boca Raton, FL, 1994, 75–97.

24. D. G. Schwartz, "The case for an interval-based representation of linguistic truth", *Fuzzy Sets and Systems*, **17**, 153–165 (1985).

25. D. G. Schwartz, "Axioms for a theory of semantic equivalence", *Fuzzy Sets and Systems*, **21**, 319–349 (1987).

26. M. H. Smith and V. Kreinovich, "Optimal strategy of switching reasoning methods in fuzzy control", Chapter 6 in H. T. Nguyen, M. Sugeno, R. Tong, and R. Yager (eds.), *Theoretical aspects of fuzzy control*, J. Wiley, N.Y., 1995, 117–146.

27. M. Sugeno, (editor), *Industrial applications of fuzzy control*, North Holland, Amsterdam, 1985.

28. L. Zadeh, "Fuzzy sets", *Information and control*, **8**, 338–353 (1965).

29. H. J. Zimmerman, "Results of empirical studies in fuzzy set theory", In: G. J. Klir (ed.), *Applied General System Research*, Plenum, N. Y., 1987, 303–312.

APPENDIX
TWO POSSIBLE DEFINITIONS OF INTERVAL-VALUED FUZZY LOGIC

In justifying Definition 3.5, we used the same idea as in Definitions 2.3 or 3.2: if we want to estimate the degree of belief in $F\&G$, then the only information that we have available for that estimation is:

- the degree of belief in F, and
- the degree of belief in G.

These formulas can be actually related (e.g., G can coincide with $\neg F$), but we are *not* using any information about this relation.

In Section 3.1, we have shown that if we know the relation between F and G then we can get an estimate for the expert's degree of belief in $t(F\&G)$ that is better than the result of applying an operation min. Indeed, this operation min describes an "average" meaning of &, and in specific cases, another operation may be a better fit. Similarly, we can show that if we know the relationship between F and G, then we can get a better estimate for the interval of possible degrees of belief than by applying Definition 3.5. Let us describe this idea in some detail.

If we know the intervals $\mathbf{t}(v_1), ..., \mathbf{t}(v_n)$, then, for an arbitrary formula F, we can define an interval $\tilde{\mathbf{t}}(F)$ as follows (we use different notations to avoid confusion with $\mathbf{t}(F)$). Namely, for each set of values $t(v_1) \in \mathbf{t}(v_1), ..., t(v_n) \in \mathbf{t}(v_n)$, we could apply Definition

3.2, and get the value $t(F)$. Thus, we define a *function* with n inputs $t(v_1), ..., t(v_n)$, and an output $t(F)$. Let us denote this function by $f_F(t(v_1), ..., t(v_n))$.

For example, if the formula F is $v_1 \vee \neg v_1$, then $t(F) = \max(t(v_1), 1 - t(v_1))$, and therefore, for this formula F, $f_F(t(v_1)) = \max(t(v_1), 1 - t(v_1))$.

We can then take the set of all possible values $t(F)$ that result from this procedure as $\tilde{\mathbf{t}}(F)$. In other words, we can define $\tilde{\mathbf{t}}(F)$ as an *image* of the intervals $\mathbf{t}(v_i)$ under the mapping f_F: $\tilde{\mathbf{t}}(F) = \{f_F(t(v_1), ..., t(v_n)) \,|\, t(v_1) \in \mathbf{t}(v_1), ..., t(v_n) \in \mathbf{t}(v_n)\}$.

The point that we want to make is that thus defined interval function $\tilde{\mathbf{t}}$ is *different* from the above-defined function $\mathbf{t}(F)$. As an example, we can take the same formula $F = v_1 \vee \neg v_1$ that we used to illustrate this new notion, with $\mathbf{t}(v_1) = [0, 1]$. In this case:

- Definition 3.5 leads to:
  - $\mathbf{t}(\neg v_1) = 1 - \mathbf{t}(v_1) = 1 - [0, 1] = [1 - 1, 1 - 0] = [0, 1]$.
  - $\mathbf{t}(F) = \max(\mathbf{t}(v_1), \mathbf{t}(\neg v_1)) = \max([0, 1], [0, 1]) = [0, 1]$.
- On the other hand, the alternative definition described in this comment will lead to defining $\tilde{\mathbf{t}}(F)$ as the set of all values of the function $f_F(t(v_1)) = \max(t(v_1), 1 - t(v_1))$ for all $t(v_1) \in [0, 1]$. To describe this set of values, let us consider two cases: $t(v_1) \geq 0.5$ and $t(v_1) < 0.5$.
  - If $t(v_1) \geq 0.5$, then $1 - t(v_1) \leq 0.5$ and therefore, $t(v_1) \geq 1 - t(v_1)$ and $f_F(t(v_1)) = \max(t(v_1), 1 - t(v_1)) = t_1$. Therefore, when $t(v_1) \in [0.5, 1]$, then the function $f_F(t(v_1))$ takes the values from the interval $[0.5, 1]$.
  - If $t(v_1) < 0.5$, then $1 - t(v_1) > 0.5$ and therefore, $t(v_1) < 1 - t(v_1)$ and $f_F(t(v_1)) = \max(t(v_1), 1 - t(v_1)) = 1 - t(v_1)$. Therefore, when $t(v_1) \in [0, 0.5)$, then the function $f_F(t(v_1))$ takes the values from the interval $(0.5, 1]$.

  The complete image is equal to the union of these two sets, i.e., to $f_F([0, 1]) = f_F([0, 0.5)) \cup f_F([0.5, 1]) = (0.5, 1] \cup [0.5, 1] = [0.5, 1]$. This image is exactly the set $\tilde{\mathbf{t}}(F) = f_F(\mathbf{t}(v_1) = f_F([0, 1])$. So, we see that this set is *different* from the interval $\mathbf{t}(F) = [0, 1]$.

The reason for this difference, as we have already mentioned, is as follows. Our formula F is the result of applying $\vee$ to formulas $G = v_1$ and $H = \neg v_1$. For each of these component formulas G and H, both definitions lead to the same result: $\mathbf{t}(G) = \tilde{\mathbf{t}}(G) = [0, 1]$, and $\mathbf{t}(H) = \tilde{\mathbf{t}}(H) = [0, 1]$. However:

- To compute $\mathbf{t}$, we can only use the intervals $\mathbf{t}(G)$ and $\mathbf{t}(H)$, and we cannot use the fact that G and H are actually related (H is a negation of G). Since we cannot use this additional information, and the only "information" that we can use is that both $t(G) \in \mathbf{t}(G)$ and $t(H) \in \mathbf{t}(H)$ can be *arbitrary* numbers from the interval [0,1]. No wonder that as a result, we also only get a conclusion that $t(F)$ can be an arbitrary number from [0,1]. In other words, we use here the formula

$$\mathbf{t}(F) = \{\max(t_1, t_2) \,|\, t_1 \in \mathbf{t}(G), t_2 \in \mathbf{t}(H)\}.$$

- When we compute $\tilde{\mathbf{t}}(F)$, we actually take this additional information into consideration, because we only consider the set of all the values $\max(t_1, t_2)$ for which both t_1 and t_2 can be simultaneously represented as $t(G)$ and $t(H)$ for some t. In our case, this can only happen when $t_2 = 1 - t_1$. For example, we do not include the value 0.4 into the set $\tilde{\mathbf{t}}(F)$ because, although in principle, 0.4 can be represented as $\max(t_1, t_2)$ for some $t_1 \in [0,1] = \mathbf{t}(G)$ and $t_2 \in \mathbf{t}(H)$, but it can never be represented in this manner for any $t_2 = 1 - t_1$.