Fuzzy Logic Explains the Usual Choice of Logical Operations in 2-Valued Logic

Julio Urenda[1,2], Olga Kosheleva[3], and Vladik Kreinovich[2]
[1]Department of Mathematical Sciences
[2]Department of Computer Science
[3]Department of Teacher Education
University of Texas at El Paso
500 W. University
El Paso, TX 79968, USA
jcurenda@utep.edu, olgak@utep.edu, vladik@utep.edu

Abstract

In the usual 2-valued logic, from the purely mathematical viewpoint, there are many possible binary operations. However, in commonsense reasoning, we only use a few of them: why? In this paper, we show that fuzzy logic can explain the usual choice of logical operations in 2-valued logic.

1 Formulation of the Problem

In 2-valued logic, there are many possible logical operations: reminder. In the usual 2-valued logic, in which each variable can have two possible truth values – 0 (false) or 1 (true), for each n, there are many possible $n$logical operations, i.e., functions $f : \{0,1\}^n \to \{0,1\}$. To describe each such function, we need to describe, for each of 2^n boolean vectors $(a_1, \ldots, a_n)$, whether the resulting value $f(a_1, \ldots, a_n)$ is 0 or 1.

Case of unary operations. For unary operations, i.e., operations corresponding to $n = 1$, we need to describe two values: $f(0)$ and $f(1)$. For each of these two values, there are 2 possible options, so overall, we have $2 \cdot 2 = 2^2 = 4$ possible unary operations:

- the case when $f(0) = f(1) = 0$ corresponds to a constant $f(a) \equiv 0$;
- the case when $f(0) = 0$ and $f(1) = 1$ corresponds to the identity function
$$f(a) = a;$$
- the case when $f(0) = 1$ and $f(1) = 0$ corresponds to negation $f(a) = \neg a$; and

- the case when $f(0) = f(1) = 1$ corresponds to a constant function

$$f(a) = 1.$$

The only non-trivial case is negation, and it is indeed actively used in our logical reasoning.

Case of binary operations. For binary operations, i.e., operations corresponding to $n = 2$, we need to describe four values $f(0,0)$, $f(0,1)$, $f(1,0)$, and $f(1,1)$. For each of these four values, there are 2 possible options, so overall, we have $2^4 = 16$ possible binary operations:

- the case when $f(0,0) = f(0,1) = f(1,0) = f(1,1) = 0$ corresponds to a constant function $f(a,b) \equiv 0$;
- the case when $f(0,0) = f(0,1) = f(1,0) = 0$ and $f(1,1) = 1$ corresponds to "and" $f(a,b) = a \,\&\, b$;
- the case when $f(0,0) = f(0,1) = 0$, $f(1,0) = 1$, and $f(1,1) = 0$, corresponds to $f(a,b) = a \,\&\, \neg b$;
- the case when $f(0,0) = f(0,1) = 0$ and $f(1,0) = f(1,1) = 1$, corresponds to $f(a,b) = a$;
- the case when $f(0,0) = 0$, $f(0,1) = 1$, and $f(1,0) = f(1,1) = 0$, corresponds to $f(a,b) = \neg a \,\&\, b$;
- the case when $f(0,0) = 0$, $f(0,1) = 1$, $f(1,0) = 0$, and $f(1,1) = 1$, corresponds to $f(a,b) = b$;
- the case when $f(0,0) = 0$, $f(0,1) = 1$, $f(1,0) = 1$, and $f(1,1) = 0$, corresponds to exclusive "or" (= addition modulo 2) $f(a,b) = a \oplus b$;
- the case when $f(0,0) = 0$ and $f(0,1) = f(1,0) = f(1,1) = 1$, corresponds to "or" $f(a,b) = a \vee b$;
- the case when $f(0,0) = 1$ and $f(0,1) = f(1,0) = f(1,1) = 0$, corresponds to $f(a,b) = \neg a \,\&\, \neg b$;
- the case when $f(0,0) = 1$, $f(0,1) = f(1,0) = 0$, and $f(1,1) = 1$, corresponds to equivalence (equality) $f(a,b) = a \equiv b$;
- the case when $f(0,0) = 1$, $f(0,1) = 0$, $f(1,0) = 1$, and $f(1,1) = 0$, corresponds to $f(a,b) = \neg b$;
- the case when $f(0,0) = 1$, $f(0,1) = 0$, and $f(1,0) = f(1,1) = 1$, corresponds to $f(a,b) = a \vee \neg b$ or, equivalently, to the implication

$$f(a,b) = b \to a;$$

- the case when $f(0,0) = f(0,1) = 1$, and $f(1,0) = f(1,1) = 0$, corresponds to $f(a,b) = \neg a$;
- the case when $f(0,0) = f(0,1) = 1$, $f(1,0) = 0$, and $f(1,1) = 1$, corresponds to $f(a,b) = \neg a \vee b$, or, equivalently, to the implication

$$f(a,b) = a \to b;$$

- the case when $f(0,0) = f(0,1) = f(1,0) = 1$, and $f(1,1) = 0$, corresponds to $f(a,b) = \neg a \vee \neg b$;
- the case when $f(0,0) = f(0,1) = f(1,0) = f(1,1) = 1$, corresponds to a constant function $f(a,b) \equiv 1$.

In commonsense reasoning, we only some of the binary operations. Out of the above 16 operations,

- two are constants: $f(a,b) = 0$ and $f(a,b) = 1$, and
- four are actually unary: $f(a,b) = a$, $f(a,b)\neg a$, $f(a,b) = b$, and

$$f(a,b) = \neg b.$$

In addition to these $2 + 4 = 6$ operations, there are also 10 non-constant and non-unary binary logical operations:

- six named operations "and", "or", exclusive "or", equivalence, and two implications ($a \to b$ and $b \to a$), and
- four usually un-named logical operations

$$\neg a \,\&\, b, \quad a \,\&\, \neg b, \quad \neg a \,\&\, \neg b, \text{ and } \neg a \vee \neg b.$$

In commonsense reasoning, however, we only use the named operations. Why?

Maybe it is the question of efficiency? Maybe it is the question of efficiency? To check on this, we can use the experience of computer design, where the constituent binary gates are selected so as to make computations more efficient.

Unfortunately, this leads to a completely different set of binary operations: e.g., computers typically use "nand" gates, that implement the function $f(a,b) = \neg(a \,\&\, b) = \neg a \vee \neg b$, but they never use gates corresponding to implication. So, the usual selection of binary logical operations remains a mystery.

What we do in this paper. In this paper, we show that the usual choice of logical operations in the 2-valued logic can be explained by ... fuzzy logic.

2 Our Explanation

Why fuzzy logic. We are interested in operations in 2-valued logics, so why should we take fuzzy logic into account? The reason is straightforward: in commonsense reasoning, we deal not only with precisely defined statements, but also with imprecise ("fuzzy") ones. For example:

- We can say that the age of a person is 18 or above *and* and this person is a US citizen, so he or she is eligible to vote.
- We can also say that a student is good academically and enthusiastic about research, so this student can be recommended for the graduate school.

We researchers may immediately see the difference between these two uses of "and": precisely defined ("crisp") in the first case, fuzzy in the second case. However, to many people, these two examples are very similar.

So, to understand why some binary operations are used in commonsense reasoning and some are not, it is desirable to consider the use of each operation not only in the 2-valued logic, but also in the more general fuzzy case; see, e.g., [1, 3, 4, 6, 7, 9].

Which fuzzy generalizations of binary operations should we consider? In fuzzy logic, in addition to value 1 (true) and 0 (false), we also consider intermediate values corresponding to uncertainty. To describe such intermediate degrees, it is reasonable to consider real numbers intermediate between 0 or 1 – this was exactly the original Zadeh's idea which is still actively used in applications of fuzzy logic.

So, to compare different binary operations, we need to extend these operations from the original set $\{0, 1\}$ to the whole interval $[0, 1]$. There are many possible extension of this type; which one should we select?

A natural idea is to select the most robust operations. For each fuzzy statement, its degree of confidence has to be elicited from the person making this statement. These statements are fuzzy, so naturally, it is not reasonable to expect that the same expert will always produce the exact same number: for the same statement, the expert can one day produce one number, another day a slightly different number, etc. When we plot these numbers, we will get something like a bell-shaped histogram – similar to what we get when we repeatedly measure the same quantity by the same measuring instrument. It is therefore reasonable to say that, in effect, the value a marked by an expert can differ from the corresponding mean value $\overline{a}$ by some small random value Δa, with zero mean and small standard deviation σ.

Since this small difference does not affect the user's perception, it should not affect the result of commonsense reasoning – in particular, the result of applying a binary operation to the corresponding imprecise numbers should not change much if we use slightly different estimates of the same expert. For example, 0.9 and 0.91 probably represent the same degree of expert's confidence. So, it is

not reasonable to expect that we should get drastically different values of $a \,\&\, b$ if we use $a = 0.9$ or $a = 0.91$.

To be more precise, if we fix the value of one of the variables in a binary operation, then the effect of changing the second value on the result should be as small as possible. Due to the probabilistic character, we can only talk about being small "on average", i.e., about the smallest possible mean square difference. This idea was, in effect, presented in [5, 6, 8] for the case of "and"- and "or"-operations; let us show how it can be extended to all possible binary logical operations.

For a function $F(a)$ of one variable, if we replace a with $a + \Delta a$, then the value $F(a)$ changes to $F(a + \Delta a)$. Since the difference Δa is small, we can expand the above expression in Taylor series and ignore terms which are quadratic (or higher order) in terms of Δa. Thus, we keep only linear terms in this expansion: $F(a + \Delta a) = F(a) + F'(a) \cdot \Delta a$, where $F'(a)$, as usual, indicates the derivative. The resulting difference in the value of $F(a)$ is equal to $\Delta F \stackrel{\text{def}}{=} F(a+\Delta a) - F(a) = F'(a) \cdot \Delta a$. Here, the mean squared value (variance) of Δa is equal to σ^2; thus, the mean squared value of $F'(a) \cdot \Delta a$ is equal to

$$(F'(a))^2 \cdot \sigma^2.$$

We are interested in the mean value of this difference. Here, a can take any value from the interval $[0, 1]$, so the resulting mean value takes the form $\int_0^1 (F'(a))^2 \cdot \sigma^2 \, da$. Since σ is a constant, minimizing this difference is equivalent to minimizing the integral $\int_0^1 (F'(a))^2 \, da$.

Our goal is to extend a binary operation from the 2-valued set $\{0, 1\}$. So, we usually know the values of the function $F(a)$ for $a = 0$ and $a = 1$. In general, a function $f(x)$ that minimizes a functional $\int L(f, f') \, dx$ is described by the Euler-Lagrange equation

$$\frac{\partial L}{\partial f} - \frac{d}{dx} \frac{\partial L}{\partial f'} = 0;$$

see, e.g., [2]. In our case, $L = (F'(a))^2$, so the Euler-Lagrange equation has the form

$$\frac{d}{da}(2F'(a)) = 2F''(a) = 0.$$

Thus, $F''(a) = 0$, meaning that the function $F(a)$ is linear.

Thus, we conclude that in our extensions of binary (and other) operations, the corresponding function should be linear in each of the variables, i.e., that this function should be bilinear (or, in general, multi-linear).

Such an extension is unique: a proof. Let us show that this requirement of bilinearity uniquely determines the corresponding extension. Indeed, suppose that two bilinear expressions $f(a, b)$ and $g(a, b)$ have the same values when $a \in \{0, 1\}$ and $b \in \{0, 1\}$. In this case, the difference $d(a, b) = f(a, b) - g(a, b)$ is also a bilinear expression whose value for all four pairs (a, b) for which $a \in \{0, 1\}$ and $b \in \{0, 1\}$ is equal to 0. Since $d(0, 0) = d(1, 0) = 0$, by linearity, we conclude that $d(a, 0) = 0$ for all a. Similarly, since $d(0, 1) = d(1, 1) = 0$, by linearity, we

conclude that $d(a,1)=0$ for all a. Now, since $d(a,0)=d(a,1)=0$, by linearity, we conclude that $d(a,b)=0$ for all a and b – thus, indeed, $f(a,b)=g(a,b)$ and the extension is indeed defined uniquely.

How can we find the corresponding bilinear extension. For negation, linear interpolation leads to the usual formula $f(a)=1-a$.

Let us see what happens for binary operations. Once we know the values $f(0,0)$ and $f(1,0)$, we can use linear interpolation to find the values $f(a,0)$ for all a:

$$f(a,0)=f(0,0)+a\cdot(f(1,0)-f(0,0)).$$

Similarly, once we know the values $f(0,1)$ and $f(1,1)$, we can use linear interpolation to find the values $f(a,1)$ for all a:

$$f(a,1)=f(0,1)+a\cdot(f(1,1)-f(0,1)).$$

Now, since we know, for each a, the values $f(a,0)$ and $f(a,1)$, we can use linear interpolation to find the value $f(a,b)$ for each b as

$$f(a,b)=f(a,0)+b\cdot(f(a,1)-f(a,0)).$$

Let us use this idea to find the bilinear extensions of all ten non-trivial binary operations.

Let us list bilinear extensions of all non-trivial binary operations.

- for $a\,\&\,b$, we have $f(a,b)=a\cdot b$;
- for $a\,\&\,\neg b$, we have $f(a,b)=a\cdot(1-b)=a-a\cdot b$;
- for $\neg a\,\&\,b$, we have $f(a,b)=(1-a)\cdot b=b-a\cdot b$;
- for $a\oplus b$, we have $f(a,b)=a+b-2a\cdot b$;
- for $a\vee b$, we have $f(a,b)=a+b-a\cdot b$;
- for $\neg a\,\&\,\neg b$, we have $f(a,b)=(1-a)\cdot(1-b)$;
- for $a\equiv b$, we have $f(a,b)=1-a-b+2a\cdot b$;
- for $\neg a\vee b=a\to b$, we have $f(a,b)=1-a+a\cdot b$;
- for $\neg a\vee b=b\to a$, we have $f(a,b)=1-b+a\cdot b$;
- finally, for $\neg a\vee\neg b=\neg(a\,\&\,b)$, we have $f(a,b)=1-a\cdot b$.

Which operations should we select as basic. As the basic operations, we should select the ones which are the easiest to compute. Of course, we should have *negation* $f(a)=1-a$, since it is the easiest to compute: it requires only one subtraction.

Out of all the above ten binary operations, the simplest to compute is $f(a,b)=a\cdot b$ (corresponding to "*and*") which requires only one arithmetic

operation – multiplication. All other operations need at least one more arithmetic operation. This explains why "and" is one of the basic operations in commonsense reasoning.

Several other operations can be described in terms of the selected "and"- and "not"-operations $f_{\&}(a,b) = a \cdot b$ and $f_{\neg}(a) = 1 - a$:

- the operation $f(a,b) = a - a \cdot b$ corresponding to $a \,\&\, \neg b$ can be represented as $f_{\&}(a, f_{\neg}(b))$;
- the operation $f(a,b) = b - a \cdot b$ corresponding to $\neg a \,\&\, b$ can be represented as $f_{\&}(f_{\neg}(a), b)$;
- the operation $f(a,b) = a + b - a \cdot b$ corresponding to $a \vee b$ can be represented as $f_{\neg}(f_{\&}(f_{\neg}(a), f_{\neg}(b)))$;
- the operation $f(a,b) = (1-a) \cdot (1-b)$ corresponding to $\neg a \,\&\, \neg b$ can be represented as $f_{\&}(f_{\neg}(a), f_{\neg}(b))$;
- the operation $f(a,b) = 1 - a + a \cdot b$ corresponding to $\neg a \vee b = a \to b$ can be represented as $f_{\neg}(f_{\&}(a, f_{\neg}(b)))$;
- the operation $f(a,b) = 1 - b + a \cdot b$ corresponding to $a \vee \neg b = b \to a$ can be represented as $f_{\neg}(f_{\&}(f_{\neg}(a), b))$; and
- the operation $f(a,b) = 1 - a \cdot b$ corresponding to $\neg a \vee \neg b = \neg(a \,\&\, b)$ can be represented as $f_{\neg}(f_{\&}(a,b))$.

There are two binary operations which *cannot* be represented as compositions of the selected "and" and "or"-operations: namely:

- the operation $f(a,b) = a + b - 2a \cdot b$ corresponding to exclusive "or" $a \oplus b$, and
- the operation $f(a,b) = 1 - a - b + 2a \cdot b$ corresponding to equivalence $a \equiv b$.

Thus, we need to add at least one of these operations to our list of basic operations. Out of these two operations, the simplest to compute – i.e., requiring the smallest number of arithmetic operations – is the function corresponding to *exclusive "or"*. This explains why we use exclusive "or" in commonsense reasoning.

What about operations that *can* be described in terms of "and" and "not"? For some of these operations, e.g., for the function

$$f(a,b) = a - a \cdot b = a \cdot (1-b)$$

corresponding to $a \,\&\, \neg b$, direct computation requires exactly as many arithmetic operations as computing the corresponding representation in terms of "and" and "or". However, there is one major exception: for the function $f(a,b) = a + b - a \cdot b$ corresponding to "or":

- its straightforward computation requires two additions/subtractions and one multiplication, while
- its computation as $f_\neg(f_\&(f_\neg(a), f_\neg(b))$ requires one multiplication (when applying $f_\&$) but *three* subtractions (corresponding to three negation operations).

Thus, for the purpose of efficiency, it makes sense to consider *"or"* as a separate operation. This explains why we use "or" in commonsense reasoning.

So far, we have explained all basic operations except for implication. Explaining implication requires a slightly more subtle analysis.

Why implications. In the above analysis, we considered addition and subtraction to be equally complex. This is indeed the case for computer-based computations, but for us humans, subtraction is slightly more complex than addition. This does not change our conclusion about operations like $f(a,b) = a - a \cdot b$: whether we compute them directly or as $f(a,b) = f_\&(a, f_\neg(b))$, in both cases, we use the same number of multiplications, the same number of additions, and the same number of subtractions.

There is, however, a difference for implication operations such as

$$f(a,b) = 1 - a + a \cdot b = f_\neg(f_\&(a, f_\neg(b))) :$$

- its direct computation requires one multiplication, one addition, and one subtraction, while
- its computation in terms of "and"- and "not"-operations requires one multiplication and two subtractions.

In this sense, the direct computation of implication is more efficient – which explains why we also use implication in commonsense reasoning.

Conclusion. By using fuzzy logic, we have explained why negation, "and", "or", implication, and exclusive "or" are used in commonsense reasoning while other binary 2-valued logical operations are not.

3 Auxiliary Result: Why the Usual Quantifiers?

Formulation of the problem. In the previous sections, we consider binary logical operations. In our reasoning, we also use quantifiers such as "for all" and "there exists" which are, in effect, n-ary logical operations, where n is the number of possible objects.

Why these quantifiers? Why not use additional quantifiers like "there exists at least two"? Let us analyze this question from the same fuzzy-based viewpoint from which we analyzed binary operations. It turns out that this way, we get a (partial) explanation for the usual choice of quantifiers.

Why universal quantifier. Let us consider the case of n objects $1, \ldots, n$. We have some property $p(i)$ which, for each object i, can be true or false. We would

like to combine these n truth values into a single one, i.e., we need an n-ary operation $f(a_1, \ldots, a_n)$ that would transform n truth values $p(1)$, $\ldots$, $p(n)$ into a single combined value $f(p(1), \ldots, p(n))$.

Similarly to the previous chapter, let us consider a fuzzy version. Just like all non-degenerate binary operations – i.e., operations that are not constants or unary operations – must contain a product of two numbers, similarly, all non-degenerate n-ary operations must contain the product of n numbers.

Thus, the simplest possible case – with the fastest computations – is when the operation is simply the product of the given n numbers, i.e., the operation $f(a_1, \ldots, a_n) = a_1 \cdot \ldots \cdot a_n$. Indeed, every other operation requires also addition of subtraction. This operation transforms the values $p(1)$, $\ldots$, $p(n)$ into their product $p(1) \cdot \ldots \cdot p(n)$, that corresponds exactly to the formula $p(1)\,\&\,\ldots\,\&\,p(n)$, i.e., to the formula $\forall i\, p(i)$. This explains the ubiquity of the universal quantifiers.

Why existential quantifier. The universal quantifier has the property that it does not change if we permute the objects: $\prod_{i=1}^{n} p(i) = \prod_{i=1}^{n} p(\pi(i))$ for every permutation $\pi : \{1, \ldots, n\} \to \{1, \ldots, n\}$. This condition of permutation invariance holds for all quantifiers and it is natural to be required. We did not explicitly impose this condition in our derivation of universal quantifier for only one reason – that we were able to derive this quantifier only from the requirement of computational simplicity, without a need to also explicitly require permutation invariance.

However, now that we go from the justification of the simplest possible quantifier to a justification of other quantifiers, we need to explicitly require permutation invariance – otherwise, the next simplest operations are operations like

$$\neg a_1\,\&\,a_2\,\&\,a_2\,\ldots\,\&\,a_n.$$

It turns out that among permutation-invariant n-ary logical operations, the simplest are:

- the operation $\neg\forall i\, p(i)$ for which the corresponding formula $1 - \prod_{i=1}^{n} p_i$ requires $n-1$ multiplications and one subtraction;
- the operation $\forall i\, \neg p(i)$ for which the corresponding formula $\prod_{i=1}^{n} (1 - p_i)$ requires $n-1$ multiplications and n subtractions; and
- the operation $\exists i\, p(i)$, i.e., equivalently, $\neg\forall i\, \neg p(i)$, for which the corresponding formula $1 - \prod_{i=1}^{n} (1 - p_i)$ requires $n-1$ multiplications and $n+1$ subtractions.

This explains the ubiquity of existential quantifiers.

Acknowledgments

This work was supported in part by the US National Science Foundation grants 1623190 (A Model of Change for Preparing a New Generation for Professional Practice in Computer Science) and HRD-1242122 (Cyber-ShARE Center of Excellence).

References

[1] R. Belohlavek, J. W. Dauben, and G. J. Klir, *Fuzzy Logic and Mathematics: A Historical Perspective*, Oxford University Press, New York, 2017.

[2] I. M. Gelfand and S V. Fomin, *Calculus of Variations*, Dover Publ., New York, 2000.

[3] G. Klir and B. Yuan, *Fuzzy Sets and Fuzzy Logic*, Prentice Hall, Upper Saddle River, New Jersey, 1995.

[4] J. M. Mendel, *Uncertain Rule-Based Fuzzy Systems: Introduction and New Directions*, Springer, Cham, Switzerland, 2017.

[5] H. T. Nguyen, V. Kreinovich, and D. Tolbert, "On robustness of fuzzy logics", *Proceedings of the 1993 IEEE International Conference on Fuzzy Systems FUZZ-IEEE'93*, San Francisco, California, March 1993, Vol. 1, pp. 543–547.

[6] H. T. Nguyen, C. L. Walker, and E. A. Walker, *A First Course in Fuzzy Logic*, Chapman and Hall/CRC, Boca Raton, Florida, 2019.

[7] V. Novák, I. Perfilieva, and J. Močkoř, *Mathematical Principles of Fuzzy Logic*, Kluwer, Boston, Dordrecht, 1999.

[8] D. Tolbert, *Finding "and" and "or" operations that are least sensitive to change in intelligent control*, University of Texas at El Paso, Department of Computer Science, Master's Thesis, 1994.

[9] L. A. Zadeh, "Fuzzy sets", *Information and Control*, 1965, Vol. 8, pp. 338–353.