# Fuzzy and Fuzzing: Contrary to the Usual Viewpoint, Fuzzy Techniques Naturally Lead to Fuzzing

Stephanie Reyes[1], Saied Tizpaz-Niari[2], and Vladik Kreinovich[1]
[1]Department of Computer Science, University of Texas at El Paso
500 W. University, El Paso, TX 79968, USA
sreyes31@miners.utep.edu, vladik@utep.edu
[2]Department of Computer Science, University of Illinois at Chicago
851 S. Morgan St., Chicago, IL 60607, USA, saeid@uic.edu

**Abstract**

At present, one of the most effective and widely used methods of software testing is a method known as *fuzzing*. The name of this method come from the same word *fuzzy* as fuzzy logic and related fuzzy techniques. However, the authors of the fuzzing technique have always emphasized that there is no relation between the two uses of this word. It is indeed true that the invention of fuzzing was not related to fuzzy logic. However, we show, in this paper, that a straightforward use of simple fuzzy ideas naturally leads to fuzzing – in this sense, these two uses of the word *fuzzy* are indeed related.

## 1 Introduction

**Traditional approach to software testing.** The modern world runs on software. A minor mistake in a program can cause a big disaster. It is therefore very important to test each program – and to find all its faults – before the program is released for use.

In the past, software tests used three main ideas. One of these ideas is to test the program on randomly generated inputs. The more random inputs we test, the larger the probability that the program will also work well on other random inputs.

Another idea is caused by the fact that in most cases, there are bounds on possible values of each input. For example, for medical software, there are bounds on possible values of body temperature, there are bounds on possible values of blood pressure, etc. Extreme cases of each of the variables are rare, but it is important to make sure that the software works well in such situations. So, it is reasonable to test the software also on such extreme values.

The third idea comes from the fact that programs usually include branching, when depending on some condition the program perform different actions. It is therefore important to make sure that for each such branching, the program works well for some examples from both branches – i.e., for each branching, we need to come up with testing examples from both branches.

**Traditional approach to software testing is not always enough: enter fuzzing.** The first two ideas have helped to find many faults, including rare faults that occur when some of the inputs take extreme values. However, sometimes there are also rare faults that happen for non-extreme inputs. Such faults are difficult to catch by traditional software engineering techniques. To catch such faults, software engineers invented a new software testing technique that they called *fuzzing* – the techniques that is now actively and successfully used in software testing.

This technique is, in effect, a clever combination of what we listed as the first idea (testing on random inputs) and the third idea (testing on both branches). The main difference between fuzzing and the straightforward application of the third idea is that instead of focusing on a *single* branching, we take into account *all* the branchings that the program goes through on a given input. Specifically, for each input, we record what selection the program made on each of the branchings; this sequence of selections forms what, in this technique, is called a *path*.

In this technique, at each stage, we keep a list of inputs – which is selected in such a way that different inputs from this list lead to different paths. To achieve this goal, we start with a single random input – which forms the first element of this list. At each moment of time, we randomly – usually, with equal probability – select one of the inputs from the list, modify the selected input and test the program on this modified input. If for this input, the path is different from all the previously observed paths, this input is added to the list.

This, in a nutshell, is what fuzzing is all about.

**A toy example explains why fuzzing is often efficient.** At first glance, the above technique, while reasonable, does not sound like a spectacular idea that can successfully detect rare faults, but it does. To explain why, let us use the toy example from one of the pioneering papers on fuzzing [2, 3].

Suppose that the program is checking 4-letter combinations, and there is a bug that only occurs when you input one of the possible combinations – namely, the combination 'bad!'. In a (hopefully) easily understandable pseudocode, this bug has the following form:

```
if (letter[0] == 'b')
  {if (letter[1] == 'a')
    {if (letter[2] == 'd')
      {if (letter[3] == '!')
         {print "wrong result"}}}}
```

If we simply perform random testing by testing all possible combinations of four ASCII symbols with equal probability, then, since there are 256 such symbols,

the probability of finding this bug is equal to $1/256^4$. Thus, on average, to find this bug, we will need $256^4 = (2^8)^4 = 2^{32} \approx 4 \cdot 10^9$ iterations. In this simple example, such an exhaustive testing is possible, but in more complex situations, we will not find the bug after any realistic number of iterations.

If we use fuzzing, the number of needed iterations decreases drastically. Indeed, once we find a word that starts with the letter 'b', we invoke the first if-statement and thus, get an input for which the path is different from the path corresponding to the original input. Since the probability that we will select the first symbol out of 4 is 1/4, and the probability that the selected first symbol being 'b' is 1/256, the overall probability that we will get a word starting with 'b' is $1/4 \cdot 1/256 = 1/1024$. So, we need, on average, 1024 iterations to find such an input.

Now, according to the above-described general fuzzing algorithm, we have two inputs that we will modify. The probability that we will select to modify the second input is 1/2, and the probability that we will modify the second letter and that the modified second letter is 'a' is 1/2024. Thus, the probability that by applying a modification, we will find a word that starts with 'ba' is equal to $1/2 \cdot 1/1024 = 1/2048$. So, on average, once we find the second selected input, after $2 \cdot 1024 = 2048$ more iterations, we will find the input that starts with 'ba' and for which, thus, the path will be different. So now, we have three inputs to modify.

Similarly, once we find the third selected input, after, on average, $3 \cdot 1024$ additional iterations, we will find a word that starts with 'bad' and for which, therefore, the path will be different. Thus, we will add this word as the fourth selected input. Finally, after, on average, $4 \cdot 1024$ iterations after finding the fourth selected input, we will get the word 'bad!' and thus, find the bug.

Overall, this process requires, on average,

$$1024 + 2 \cdot 1024 + 3 \cdot 1024 + 4 \cdot 1024 = 10 \cdot 1024 = 10240 \approx 10^4$$

iterations. This is a much smaller number than more than a billion iterations needed for the random search. So, indeed, fuzzing is a very efficient testing method.

**A natural question.** The name of the fuzzing method comes from the same word "fuzzy" as fuzzy logic (see, e.g., [1, 6, 8, 9, 10, 11, 13]). So, a natural question is: Are these two ideas related?

The authors of the fuzzing method did not use fuzzy logic idea in coming with this method, so a general impression is that these two ideas are not related. But is this impression really true?

**What we do in this paper.** In this paper, we show that, contrary to the general impression, there *is* a relation between these two ideas.

Specifically, we will first describe the main idea behind fuzzing in commonsense terms. In principle, there are two main approaches to translate this idea into a precise algorithm: probabilistic approach and fuzzy approach. We will then show that:

- a straightforward probabilistic translation does not lead to an efficient algorithm, while
- a straightforward fuzzy translation leads exactly to the efficient fuzzing algorithm.

## 2 How a straightforward use of simple fuzzy techniques naturally leads to fuzzing

**Let us first describe the main idea behind fuzzing in commonsense terms.** The main idea behind fuzzing is that we start with a random input, and we repeatedly modify this input. Once we find an input that leads to a path which is different from the path corresponding to the first selected input, we add this input to the list of selected inputs, and we start modifying both selected inputs. In general, once we find an input for which the path differs from all previously observed paths, we add this input to the list of selected inputs, and we start modifying all the inputs from the new list of selected inputs.

In this general description, we do not specify the relative frequencies with which we modify each of several modified inputs. Let us use common sense to determine these frequencies.

**How to use common sense to determine the relative frequencies with which we should modify each of the selected inputs.** When we have several selected inputs, it may happen that about some of them, we are more confident that a modification of this input will lead to the fault-revealing input $i_b$ that we want to find. In this case, it makes sense to modify these more-confident inputs more frequently. So, to decide which selected inputs should be modified more frequently, let us analyze to what extent it makes sense to believe that modifying each of these inputs will lead to the fault-revealing input.

The main reason why we believe that a modification of the very first – randomly selected – input $i_1$ will eventually leads to finding the fault-revealing input is that this input is somewhat similar to the fault-revealing input $i_b$; we will denote this by $i_1 \approx i_b$. Let $d_1$ denote our degree of confidence that in this case, we will find the fault-revealing input after each iteration.

By our construction, the second selected input $i_2$ is somewhat close to the first selected input $i_1$: $i_2 \approx i_1$. From the fact that the second selected input $i_2$ is somewhat close to the first selected input $i_1$ and the first selected input $i_1$ is somewhat close to the fault-revealing input $i_b$, we can conclude that the second selected input $i_2$ is also close to the fault-revealing input $i_b$ – but probably with a lower degree of certainty $d_2$, since now the conclusion that $i_2 \approx i_b$ is based on two statements about which we are not 100% certain:

$$i_2 \approx i_1 \text{ and } i_1 \approx i_b.$$

Similarly, we can conclude that the third selected input $i_3$ is somewhat close to the fault-revealing input $i_b$, but this conclusion is based on a chain of three

statements:

$$i_3 \approx i_2, \quad i_2 \approx i_1, \text{ and } i_1 \approx i_b.$$

In general, the longer the chain, the smaller our confidence, so the degree $d_3$ to which $i_3$ is somewhat close to the fault-revealing input $i_1$ is probably even smaller than $d_2$.

In general, for each $k$, we can conclude that the $k$-th selected inputs $i_k$ is somewhat close to the fault-revealing input $i_b$, but this conclusion is based on a chain of $k$ statements:

$$i_k \approx i_{k-1}, \quad \ldots, \quad i_2 \approx i_1, \text{ and } i_1 \approx i_b.$$

The longer the chain, the smaller our confidence, so the degree $d_k$ to which $i_k$ is somewhat close to the fault-revealing input $i_1$ is probably decreasing with $k$.

Let is now apply straightforward probabilistic and straightforward fuzzy approaches to estimate the degrees $d_k$.

**What if we use a straightforward probabilistic approach.** In the usual probabilistic approach, the degree of confidence is described by a probability. In this case, all basic degrees of confidence:

- that $i_1 \approx i_b$,
- that $i_2 \approx i_1$,
- $\ldots$, and
- that $i_k \approx i_{k-1}$,

are all interpreted as probabilities. For each $k$, the conclusion that $i_k$ is somewhat close to $i_b$ can be made if $k$ basic statements hold. If we know the probability $d_1$ of each of $k$ events $i_k \approx i_{k-1}$, $\ldots$, $i_2 \approx i_1$, and $i_1 \approx i_b$, what is the probability that all these $k$ vents happen at the same time?

In general, this answer depends on the correlation between these events. However, a typical situation in probabilistic approach is that we have no information about the correlation. In this case, since we have no reasons to believe that these events are positively or negatively correlated, it makes sense to assume that these events are not correlated at all – i.e., that they are independent.

This assumption not only follows informally, it also comes from a natural idea that we should not add certainty where there is none. Uncertainty in probabilistic approach is described by Shannon's entropy. So, in precise terms, this idea means selecting, out of all possible joint distributions, the one for which the entropy is the largest possible; see, e.g., [5]. For the case of several events about which we only know their probabilities, this *Maximum Entropy* approach indeed leads to the conclusion that these events should be independent.

For independent events, the probability that they all occur together is equal to their product. In our case, the probability that $i_k \approx i_d$ – i.e., that a modification of the $k$-th selected input will lead to the fault-revealing input $i_b$ – is thus equal to $d_k = d_1^k$. In particular, it makes sense to take, as $d_1$, the actual

probability that a modification of the first selected input will lead to the fault-revealing input – i.e., in the above example, the value $1/(2 \cdot 10^9) = 0.5 \cdot 10^{-9}$. The resulting probability that a modification of the second selected input can lead to the fault-revealing input is equal to $d_1^2$ and is, thus, more than a billion times smaller than $d_1$. For other selected inputs, the resulting probability is even smaller. So, in this case, with probability close to 1 we will ignore all the selected inputs except for the first one.

So, in effect, if we use straightforward probabilistic approach, we get back to the original random search – which, as we have mentioned, is not very efficient. So, the use of straightforward probabilistic approach defeats the main purpose of adding fuzzing – to find the bugs faster.

**What if we use a straightforward fuzzy approach.** What will happen if, to describe "and", instead of the probabilistic approach, we use a fuzzy approach? Readers who *are* familiar with fuzzy logic know that, in general, there are many different operations of fuzzy logic that correspond to "and"; these operations are known as *t-norms*. However, if you ask many people who have heard about fuzzy but who are not very familiar with the details of fuzzy techniques, they will answer that in fuzzy logic, "and" corresponds to minimum. Minimum is one of the two "and"-operations initially proposed by Lotfi Zadeh in his pioneering paper [13] (the other was product); minimum is still one of the most actively used t-norms.

So what happens if in translating the above commonsense expression into a precise algorithm, we use minimum to describe "and"? Let us recall that in this expression, the statement $i_k \approx i_b$ comes from applying "and" to the following $k$ statements:

$$i_k \approx i_{k-1}, \quad \ldots, \quad i_2 \approx i_1, \quad i_1 \approx i_b,$$

each of which has the same degree $d_1$. Thus, the degree $d_k$ to which the $k$-th selected input $i_k$ is somewhat close to the buggy inout $i_b$ can be estimated as the minimum $\min(d_1, \ldots, d_1)$ of $k$ numbers each of which is equal to $d_1$. This minimum is, of course, equal to $d_1$.

So, in the straightforward fuzzy approach, for all the selected inputs, the degree to which this input is somewhat close to the fault-revealing input is the same. Thus, it makes sense to modify each of the selected with the same probability – which is exactly the idea that makes fuzzing successful.

So, a straightforward use of simple fuzzy logic indeed naturally leads to the empirically efficient fuzzing algorithm.

# 3 Let us go beyond straightforward probabilistic and fuzzy approaches

**Why it is desirable to go beyond.** Our explanation of fuzzing was based on using the simplest version of fuzzy technique – the version that uses minimum as the "and"-operation (= t-norm). This explanation may be somewhat convincing to those who are not very familiar with the variety of fuzzy techniques. However,

as we have mentioned in the previous section, those who are familiar with fuzzy techniques know that there are many different t-norms. For these readers, we need a (somewhat) more convincing explanation. This is what we will do in this section.

**Let us go back to the main ideas behind the probabilistic approach.** To come up with a more convincing explanation, let us take into account that after all, whatever approach we use, we need to describe the frequencies with which we will modify different selected inputs. In other words, what we ultimately need is probabilities. So, let us go back to the probabilistic approach.

*Comments.* This does not means that we are abandoning fuzzy ideas: while fuzzy agrees are different from the objective frequency-based probabilities, we can always view fuzzy degrees as subjective probabilities. This view helps us understand that fuzzy techniques, in effect, contributed to the probability analysis (see, e.g., [7]): namely, they show that if we only know the probabilities of several events, and we want to find the probability of them happening at the same time, we can use one of the t-norms – and we can select the t-norm that best describes this specific area. This is, in effect, what happened with MYCIN, the historically first expert systems for diagnosing and treating rare blood diseases; see, e.g., [4].

**What do we know.** The main challenge that we faced was that for several events, we only know the probabilities of different events, but we do not have any information about their correlation. In this situation, we need to find an estimate for the probability that several events occur at the same time. It is known (see, e.g., [12]), that if we have several events $E_1, \ldots, E_k$ with probabilities $p_1, \ldots, p_k$, then the probability $p$ that all these events occur at the same time can take any value between the following bounds:

$$\max(p_1 + \ldots + p_k - (k-1), 0) \le p \le \min(p_1, \ldots, p_k). \tag{1}$$

These inequalities were first proposed by Fréchet and are, thus, known as *Fréchet inequalities.*

**What can we conclude.** We want to find the value $p$ within the interval (1) that will lead to the most efficient search for a fault-revealing input. In other words, we want to find the value for which the corresponding objective function attains its largest possible value.

According to calculus, the maximum of a function of one or several variables on a bounded domain is attained either at a stationary point – i.e., at a point at which all partial derivatives are equal to 0 – or on the boundary of the domain. When the domain is small, the probability that it contains a stationary point is very small. So, with high probability, we can conclude that the maximum of the objective function is attained at the boundary of the domain.

In our case, since the values $p_i$ are very small, so the left-hand side of the inequality (1) is equal to 0 and the right-hand side is very small. Thus, the interval domain of possible value of $p$ – which is described by the inequality

(1) – is very small. So, according to our conclusion, with high probability, the optimal value $p$ is attained at the boundary of this interval domain, i.e., at one of the endpoints of this interval. In other words, we have two alternatives: either $p = 0$, or $p$ is equal to the minimum $\min(p_1, \ldots, p_k)$.

The alternative $p = 0$ means that we modify the second, third, etc., selected inputs with probability 0 – i.e., in effect, that we do not use them for modification at all. In other words, this alternative means that we only modify the very first input – in which case selecting other inputs makes no sense, we can as well go back to the original often-not-feasible random choice, with no fuzzing.

So the only remaining case is indeed the case when $p = \min(p_1, \ldots, p_k)$ – i.e., exactly the case that corresponds to the above-described straightforward simple fuzzy approach, the case that explains the empirically successful fuzzing techniques. So, this way, be get a (hopefully) more convincing fuzzy-related explanation for these techniques, an explanation that does not depend on the a priori selection of a specific "and"-operation (t-norm).

## Acknowledgments

This work was supported in part by the National Science Foundation grants 1623190 (A Model of Change for Preparing a New Generation for Professional Practice in Computer Science), HRD-1834620 and HRD-2034030 (CAHSI Includes), EAR-2225395 (Center for Collective Impact in Earthquake Science C-CIES), and by the AT&T Fellowship in Information Technology.

It was also supported by a grant from the Hungarian National Research, Development and Innovation Office (NRDI), and by the Institute for Risk and Reliability, Leibniz Universitaet Hannover, Germany.

## 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] M. Böhme, D. Liyanage, and V. Wüstholz, "Estimating residual risk in greybox fuzzing", *Proceedings of the 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering ESEC/FSE'21*, Athens, Greece, August 23–28, 2021, pp. 230–421, https://doi.org/10.1145/3468264.3468570

[3] M. Böhme, V.-T. Pham, and A. Roychoudhury, "Coverage-based greybox guzzing as Markov chain", *IEEE Transactions on Software Engineering*, 2017, pp. 489–506, https://doi.org/10.1109/TSE.2017.2785841

[4] B. G. Buchanan and E. H. Shortliffe, *Rule Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*, Addison-Wesley, Reading, Massachusetts, 1984.

[5] E. T. Jaynes and G. L. Bretthorst, *Probability Theory: The Logic of Science*, Cambridge University Press, Cambridge, UK, 2003.

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

[7] V. Kreinovich, "Lotfi Zadeh: a Pioneer in AI, a Pioneer in Statistical Analysis, a Pioneer in Foundations of Mathematics, and a True Citizen of the World", *International Journal of Intelligent Technologies and Applied Statistics*, 2018, Vol. 11, No. 2, pp. 87–96.

[8] J. M. Mendel, *Explainable Uncertain Rule-Based Fuzzy Systems*, Springer, Cham, Switzerland, 2024.

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

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

[11] W. Pedrycz, A. Skowron, and V. Kreinovich (eds.), *Handbook on Granular Computing*, Wiley, Chichester, UK, 2008.

[12] D. J. Sheskin, *Handbook of Parametric and Nonparametric Statistical Procedures*, Chapman and Hall/CRC, Boca Raton, Florida, 2011.

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