2021 Third IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA) | 978-1-6654-1623-8/21/$31.00 ©2021 IEEE | DOI: 10.1109/TPSISA52974.2021.00011

Make Consumers Happy by Defuzzifying the Service Level Agreements

Kritagya Upadhyay
Department of Computer Science and Engineering
University of North Texas
Denton, TX, 76207, USA
kritagyaupadhyay@my.unt.edu

Ram Dantu
Department of Computer Science and Engineering
University of North Texas
Denton, TX, 76207, USA
ram.dantu@unt.edu

Yanyan He
Department of Computer Science and Engineering
University of North Texas
Denton, TX, 76207, USA
yanyan.he@unt.edu

Abiola Salau
Department of Computer Science and Engineering
University of North Texas
Denton, TX, 76207, USA
abiolasalau@my.unt.edu

Syed Badruddoja
Department of Computer Science and Engineering
University of North Texas
Denton, TX, 76207, USA
syedbadruddoja@my.unt.edu

***Abstract*—A Service Level Agreement (SLA) is a special kind of legal contract that binds a vendor to its customers where the vendor commits to provide certain services in exchange for certain payments from the customers. However, when customers do not get the services that they have subscribed for, it becomes a laborious job for customers to contact or visit the company and claim the correct amount of compensation or service credits. On the other hand, a Smart Contract is a contract that is a computer program that also binds multiple parties into given agreements but is a set of precise rules and is self-enforceable and self-executable. In this paper, we have introduced a novel work where we use fuzzy logic inside the Ethereum-based smart contract for two significant objectives. The first objective is to make the claim of the compensation easier and faster for customers by translating the SLA into a smart contract. The second objective is to make the smart contract even smarter and intelligent by implementing fuzzy logic so that customers who have a hard time understanding the legal jargon and ambiguities of the legal contract and SLA to find out if the compensation amount they are getting when the service is poor is good enough. Since fuzzy logic models semantics of linguistic expressions by capturing vagueness in the fuzzy sets, it becomes easier to solve the problem of contractual ambiguities and expedite the process of claiming compensation when implemented in a Blockchain-based smart contract.**

***Index Terms*—Smart contract, fuzzy logic, service level agreement, SLA, ambiguity, complexity, smart legal contract, blockchain, ethereum, clauses, interpretations**

I. Introduction and Problem Motivation

In our everyday life, when a customer makes a complaint against the company about how bad the service they are getting is, that is against the Service Level Agreement (SLA) [1], the complaint is made in natural language. For example, *"The service is slow and has been really bad for over a month now."* is the kind of complaint made by the customer to the company that has plenty of vagueness in the statement. The company takes advantage of the customer's lack of legal and contractual knowledge and tries to escape from making the compensation to the customer. And even if the complaints are heard, it takes a long time for the customers to claim their compensation and get back their refund, which results in uninvited wastage of extra time and money just to get the compensations back.

Ambiguity and vagueness are the phenomenons that many have tried to study in natural language. We find ambiguities, fuzziness, and legal jargon abundantly that is beyond our comprehension in legal contracts [2] [3] [4] and Service Level Agreements (SLAs). Fuzzy logic is an approach to compute something that is based on degrees of truth rather than the Boolean true or false (1 or 0). Natural language has many gray areas, and nothing can always be classified either as 1 or as 0. We have used fuzzy logic because it can model the semantics of linguistic expressions. After all, fuzzy sets can capture their innate ambiguity. Fuzzy logic is also much cheaper and quicker at the same time when implemented inside the smart contract [5] for the Blockchain compared to machine learning due to the simplicity of fuzzy logic's [6] [7] rule-based system and an inference engine that makes the smart contact not only smart but also intelligent.

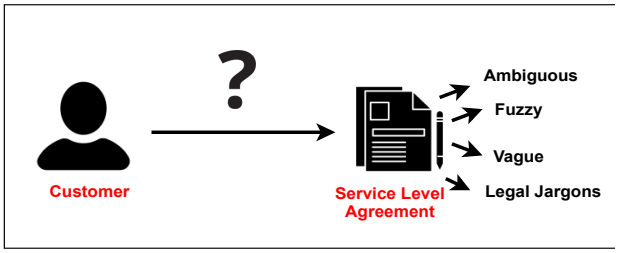

Fig. 1. A layperson does not understand the ambiguity, fuzziness, vagueness and legal jargons present in the legal contract or service level agreement.

Despite the fact that there has been substantial research going on for the smart contracts in the present day, the study specifically on the ambiguity in legal contracts and translation of the legal contracts to smart legal contracts considering ambiguity in legal contracts as the main factor has not been exhaustive. Notwithstanding that there have been innumerable types of research on ambiguities and types of ambiguities independently, there has not been any research so far on abundant legal contracts and SLAs and how we can reshape these legal contracts and SLAs into smart contracts, considering ambiguity as the main challenge. An SLA has been converted into a smart contract that can be used in Blockchain to reduce the manual effort required to claim

978-1-6654-1623-8/21/$31.00 ©2021 IEEE
DOI 10.1109/TPSISA52974.2021.00011

compensations, according to a study in [8]; nevertheless, the authors do not explain the ambiguities and legal jargon we see in the SLAs and how they were considered when converting the SLA into a smart contract. While the authors speak about the SLA management system in [9], they do not discuss the process by which we can convert an SLA into a smart contract. They also discuss only the basic functions of the SLA Management System. Furthermore, the authors discuss their proposed SLA management framework based on a two-level blockchain model, and there is a discussion of the transformation of a service level agreement into a smart contract in [10] but does not address the ambiguous requirements that can arise while writing a smart SLA.

II. Methodology

When a customer is not satisfied with the services provided by his/her company as mentioned in the company's Service Level Agreement (SLA), due to the lack of knowledge of legal jargon and vague words and phrases, it would be difficult for any customer to understand the SLA clearly and claim his/her compensation. A customer has to go through a lot of hassles even if he/she would have understood the ambiguous legal words in the SLA. Hence, in this work, we have selected a real-life SLA from a popular telecommunication vendor, Spectrum Internet from Charter Communications, and studied the vagueness and ambiguities found in the SLA. We read the whole SLA and found out that there were not any metrics properly given for the customers discussing the performance and operation of the company. Furthermore, the compensation that was provided was absolutely not in favor of the customers who are experiencing the worst internet service. Since the whole SLA was vague and the metrics were not properly set out for the customers, we concentrated and summarized the whole SLA into one general fuzzy rule. The rule is: "*If the Performance and Operation is bad, then Compensation should be high.*" Here, *Performance* is the title of Clause 3 and *Operation* is the title of Clause 4. Since the basis for calculation of compensation is the performance and operation of the company as the SLA states, we have assumed our two inputs are Performance and Operation and our output is Compensation. This fuzzy rule would be the basis for this work where we create an Ethereum-based smart contract that has fuzzy logic implementation inside. We created a smart contract that has a fuzzy inference system in it so that the smart contract can actually be smart and can decide by itself the total compensation amount to be sent back to the customer's account based on the ratings provided by the customers. There is a possibility that customers can provide fake ratings which are very low to get a higher compensation amount. However, that is the concern of this work, and since everything should be validated and everybody should come into consensus in Blockchain so cheating by providing low ratings just to get higher compensation even if a customer is getting good service is not possible. Our main focus in this work is the Ethereum-based smart contract itself that is smart and intelligent which can understand and decode the human natural language and hedges by quantifying the linguistic variables and provide us a crisp value of compensation.

Our methodology comprises of a smart contract that incorporates a fuzzy logic mechanism that has four major components and their respective functions in it. They are *Fuzzifier, Rule-based System, Inference Engine, and Defuzzifier* as shown in Fig. 2. In our methodology, we have three main phases which are explained below:

A. Inputs

First, the customer provides the crisp ratings for the vague inputs, Performance, and Operation to the smart contract. This input is measured in percentages. For example, if the customer is highly satisfied with the performance of the company but somewhat satisfied with the operations of the company, he/she would rate Performance as 90% and Operation as 40% in the smart contract.

B. Components of Smart Contract

The ratings for two inputs provided by the dissatisfied customer will now be fetched by the smart contract where it performs fuzzy logic operations inside. We have four major components inside the smart contract as further explained in detail below:

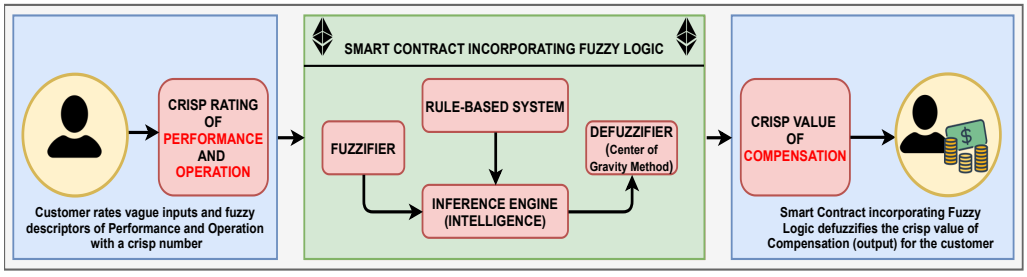

Fig. 2. Our model architecture consists of three main phases where a dissatisfied user who wants to claim compensation, provides crisp ratings of the company to the fuzzy-logic based smart which fuzzifies the inputs into linguistic variables for the generation and inference of rules and finally defuzzifies the aggregated fuzzy output into the crisp value of compensation for the customer.

1) Fuzzifier: Fuzzifier is a component that is responsible for the process of converting the crisp inputs (ratings) for Performance and Operation provided by the user/customer. The crisp ratings are converted into linguistic variables and are assigned with the membership values. The source of the membership values is either the domain expert, or intuition, or statistical analysis. In this work, the source of assignment of membership values is both domain expert and intuition. We have developed three different smart contracts called *SC 1, SC 2* and *SC 3* that have the same architecture but a different number of linguistic descriptors and hence different number of membership values for each linguistic descriptor. In our SC 1, we have the least number of descriptors for inputs, i.e., three. We increase the number of descriptors for inputs to five for SC 2. Finally, we have eight descriptors for both inputs and output in SC 3. Descriptors are fuzzy linguistic variables that describe the gray areas of the fuzzy inputs. The descriptors for inputs and output for each smart contract is provided below:

- *Smart Contract with 3 Descriptors (SC 1):*
  This smart contract has only 3 descriptors in its inputs and 5 descriptors in its output. We have also referred to this smart contract as "SC 1" in our figures below as this was the first smart contract we developed and tested. The descriptors of the inputs and outputs are provided below:
  - Performance: {*poor, good, excellent*}
  - Operation: {*slow, acceptable, rapid*}
  - Compensation: {*very low, low, normal, high, very high*}
- *Smart Contract with 5 Descriptors (SC 2):*
  Our second smart contract has 5 descriptors in its both inputs and output. The descriptors in this smart contract has been stretched out to 5 for evaluation and further research purposes. We have referred to this smart contract as "SC 2". The descriptors of the inputs and outputs are provided below:
  - Performance: {*very poor, poor, good, very good, excellent*}
  - Operation: {*very slow, slow, acceptable, fast, rapid*}

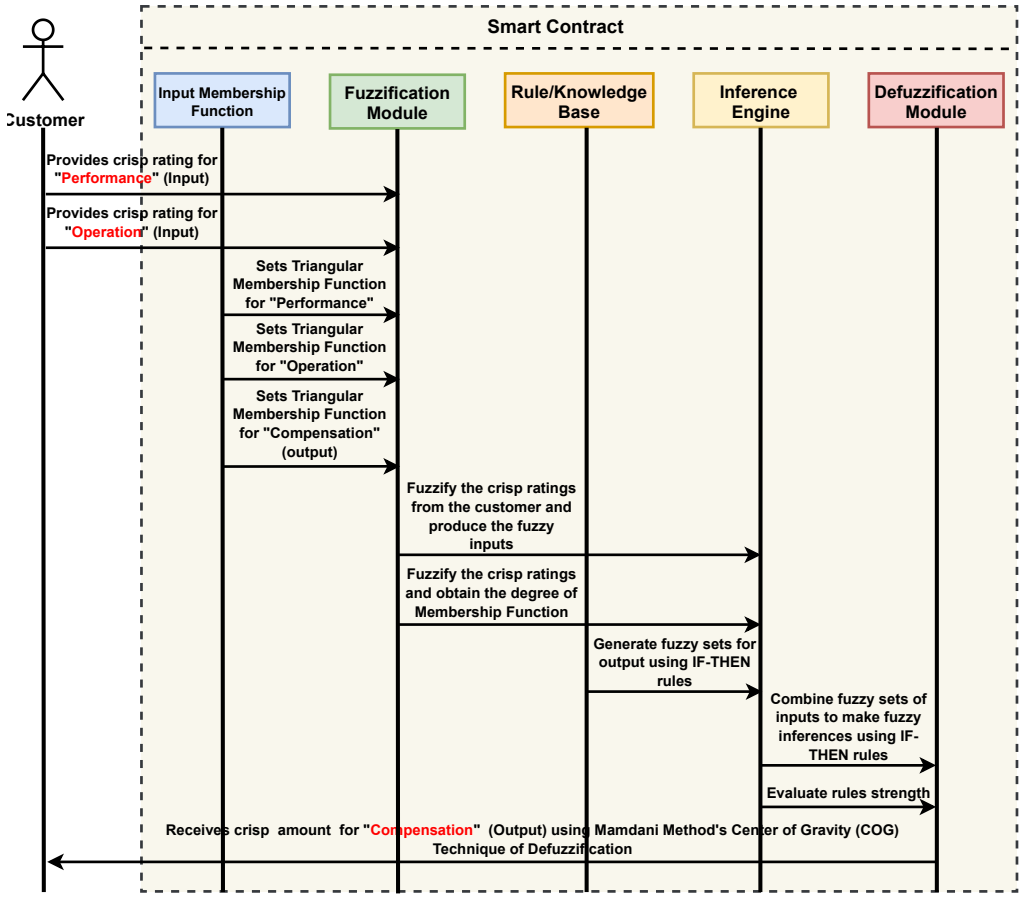

Fig. 3. Implementation of Fuzzy Logic inside Smart Contract which uses Triangular Membership Function in order to solve the problem of contractual ambiguity by fuzzifying the crisp inputs provided by the customer. With the help of a rule-based system and inference engine, the customer will get the correct amount of compensation or service credits without having to deal with the vagueness and fuzziness present in the SLA

  - Compensation: {*very low, low, normal, high, very high*}
- *Smart Contract with 8 Descriptors (SC 3):*
  Finally, our third smart contract has been even further stretched out to 8 descriptors in both inputs and outputs. The reason we also increased the number of descriptors in the output along with the inputs was that we did not want to have fewer descriptors in the output compared to the number of descriptors in the inputs. We have referred to this smart contract as "SC 3". The descriptors of the inputs and outputs for SC 3 are provided below.
  - Performance: {*extremely poor, very poor, poor, satisfactory, good, very good, extremely good, excellent*}
  - Operation: {*extremely slow, very slow, slow, mediocre, acceptable, fast, very fast, rapid*}
  - Compensation: {*extremely low, very low, low, insufficient, normal, high, very high, extremely high*}

No hard and fast rule says anything about a specific name should be given to a descriptor or there should be a specific number of descriptors in the inputs and output. We have used the aforementioned descriptors because they are suitable for this research that involves Service Level Agreements (SLA). Although there are various membership functions [11] to assign membership values after the crisp ratings are converted into linguistic descriptors, in this fuzzification process, we have used Triangular Membership Function [12]. The membership values ranges from 0 to 1 and is denoted by μ.
The Triangular membership function is defined as:

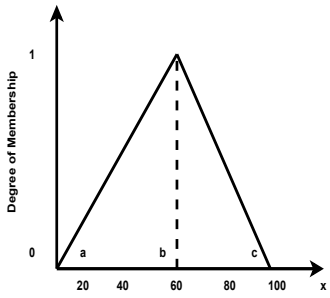

Fig. 4. An example of a Triangular Membership Function

$$\mu(x,a,b,c) = \begin{cases} 0, & \text{if } x < a \\ (x-a)/(b-a), & \text{if } a <= x <= b \\ (c-x)/(c-b), & \text{if } b < x <= c \\ 0, & \text{if } c < x \end{cases} \quad (1)$$

Where, $\mu(x,a,b,c)$ is the degree of membership of parameters a, b and c.

For example, as we can see in the sequence diagram of Fig. 3 when a customer provides the crisp ratings for Performance and Operation, the Fuzzifier takes those crisp numbers to convert them into the linguistic descriptors we have mentioned earlier. For instance, if we take the case of SC 1 which has only three linguistic descriptors when a customer rates Performance as 40%, this crisp value will be converted into a fuzzy descriptor. Hence, for some people, 40% might be *poor* and for others, the same rating of 40% might be *good* which depends on people's experience and interpretation.

2) Rule-based System: Once, the crisp values are fuzzified into descriptors and membership values are assigned for those corresponding descriptors, we construct fuzzy rules in a rule-based system that has IF, OR, AND, THEN with linguistic descriptors. These rules are very much similar to the rules from the Decision Tree. Each rule has two parts that are antecedent and consequent. Any rule can have multiple antecedents and consequents. For instance, there are three rules formed in the rule-based system after the crisp inputs are fuzzified into descriptors. Here in this instance, the descriptors are *poor* and *slow*. The antecedent is the condition and the result is the consequent. *IF the performance is "poor" OR operation is "slow"* is the antecedent, and *THEN compensation is "high"* is the consequent. There are n^2 number of rules inside the rule-based system, where *n* is the number of descriptors in inputs. As we can see in the Fig. 5, this is the matrix of rules for SC 1 that has 9 rules altogether because of three descriptors for inputs in SC 1. Similarly, our SC 2 had altogether 25 rules because five different descriptors were assigned for each input and output. Likewise, we stored a total of 64 rules in our SC 3 because the inputs and output of SC 3 had eight descriptors in its inputs and output.

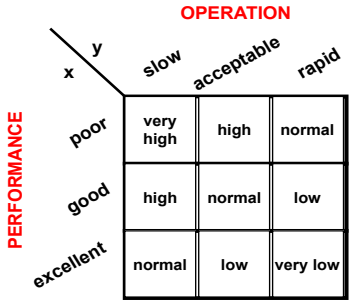

Fig. 5. Matrix of 9 rules for SC 1 as SC 1 just has three descriptors for its first input, Performance and three descriptors for its second input, Operation.

3) Inference Engine: Once the fuzzy rules are created and stored in the rule-based system, we map the fuzzy rules into the membership graphs of all the parameters. This means that we map the antecedents of a rule to the consequents of the same rule. Mapping is performed to all the rules in the rule-based system. Once the matrix is created, then the inference engine decides on what would the output become when the crisp inputs are converted to the linguistic variables. For instance, as we can see in the matrix, when Performance is *poor* and Operation is *slow*, the Compensation is *very high*. Similarly, when the Performance is *excellent* but Operation is *acceptable*, then compensation is *low*. Additionally, the Inference Engine also helps to measure the strength of the

rules and select for the final phase, defuzzification. The membership values of the antecedents are conjoined together with the intersection operator (finding the MIN or minimum) since they are connected with AND in this work. If the antecedents would be been connected with OR, then we would have used the union operator (finding the MAX or maximum). Once the membership values of the antecedents are compared and the minimum values of each rule are selected, the minimum membership value would be the unit for measuring the strength of the rules.

4) Defuzzifier: Defuzzification is the final step of fuzzy logic and hence the final component of our smart contract as well. This component is responsible for making the final decision by selecting the rule that has the highest strength. More importantly, this component is also responsible for finding the crisp value from the output of the aggregated fuzzy set. From Fig. 3, we can see that when the strength of rules is evaluated, the defuzzifier/defuzzification module transfers the fuzzy inference results back to the crisp value. This crisp value would be the final output. Although there are various defuzzification techniques such as Mean of Max method, Weighted sum method, Lambda-cut method, etc., we have used Center of Gravity (COG) method in this work. The COG is defined as:

$$X^* = (\sum_{i=1}^{n} .x_i.A_i) \div (\sum_{i=1}^{n} .A_i) \quad (2)$$

Where, X^* is the crisp output, $\sum_{i=1}^{n}$ is the sum over variable's possible values, x_i is the center of an area and A_i is the total area of the selected region.

From the sequence diagram in Fig. 3, we can also see that the final crisp value for Compensation is produced as output and sent back to the customer.

C. Output

Finally, the customer who provides the crisp ratings for the two inputs to the fuzzy logic-based smart contract, i.e., Performance and Operation, will get a crisp result back as Compensation. The Compensation is also measured in percentages just like the inputs. For example, from Fig. 6, if SC 1 is implemented, we can see when the customer provides crisp ratings for Performance and Operation as 20% and 30% respectively if the service is poor, the customer receives the Compensation as 60% of total expenses of his/her subscription to the current service. However, when the customer increases the crisp ratings for Performance and Operation to 50% and 60% respectively, the customer receives the Compensation as 26.82%. The lower the customer ratings are, the higher the compensation is and vice-versa. Nevertheless, we still discuss the accuracy of our defuzzification output in all three smart contracts below in the Results section.

III. Results

We have developed and tested three different types of smart contracts with the same architecture of fuzzy logic that we have discussed above in the Methodology section. The only way these three smarts contracts differ is by the number of descriptors and their corresponding membership functions and matrix of rules. Although the employed technique of defuzzification is the same, i.e., Center of Gravity (COG)

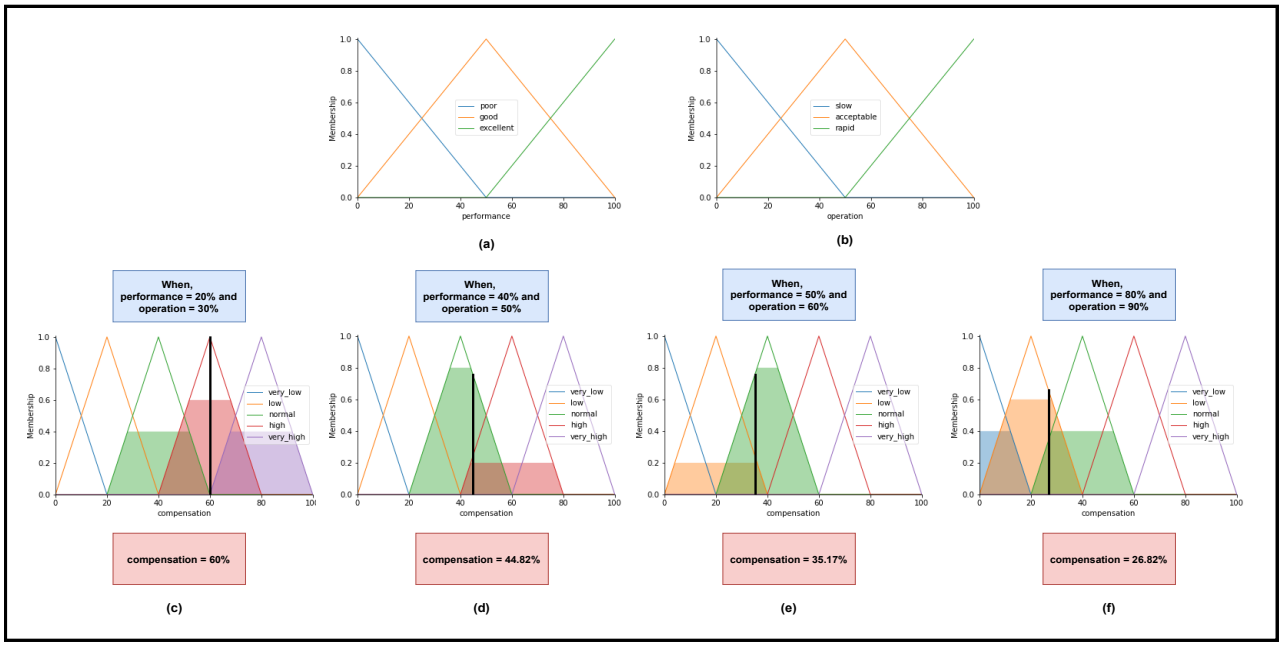

Fig. 6. Performance and Operation are the inputs of the Smart Contract incorporating Fuzzy Logic with 3 descriptors for inputs and 5 descriptors for output where Triangular MF and Center of Gravity method is used. Different series of inputs are provided to observe the varying nature of the output, i.e. Compensation.

method, these three different kinds of smart contracts have a different number of descriptors. The reason we developed and tested three different kinds of smart contracts is to successfully evaluate and analyze the performance, accuracy, and any impact of the varying number of descriptors in smart contracts when deployed into the Ethereum-based Blockchain. More is explained about these smart contracts along with their respective descriptors in detail in the following subsections.

A. *Defuzzification results from different Smart Contracts*

In our SC 1, we have got different results for Compensation when different values for inputs were provided as expected. Since the Compensation will be higher when the customer ratings are lower and will be lower when the customer ratings are higher, the inputs and output have an inverse relationship. As shown in Fig. 6 and Fig. 7, when the user provides the crisp rating of 20% and 30% in the smart contract for Performance and Operation respectively, it gives us the output of 60% for Compensation. Similarly, SC 1 gives the Compensation of 44.82% when the Performance and Operation are 40% and 60% respectively. When the inputs for Performance and Operation are 50% and 60% respectively, SC 1 gives us the Compensation of 35.17%. Finally, when the inputs for Performance and Operation are increased to 80% and 90%. the Compensation reduces to 26.82%. We can also see in Fig. 7 that the compensations calculated by the SC 1 are decreasing when the ratings for Performance and Operation are increasing.

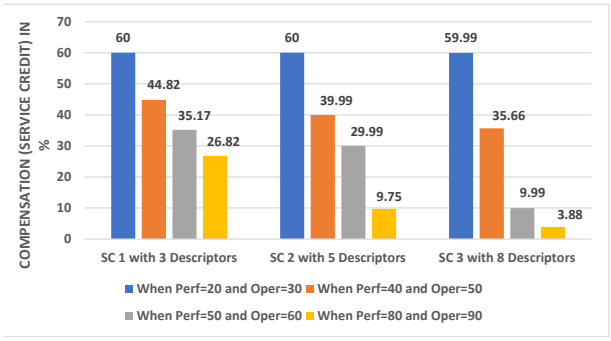

Fig. 7. Defuzzification of the output of three different Smart Contracts when different values of inputs are provided.

We provided different values of ratings as inputs again for SC 2. Although SC 2 has more number of descriptors for Performance and Operation, i.e., five, we can see that the Compensation is the same as 60% when the ratings are 20% and 30% respectively. However, when the ratings for Performance and Operation are increased to 40% and 50%, this time SC 2 gives us the Compensation of 39.99%. Similarly, when the input ratings are 50% and 60% respectively, the Compensation is 29.99%. Finally, when there are high ratings for Performance and Operation, such as 80% and 90% respectively, SC 2 gives us the Compensation of 9.75%.

Finally, for SC 3, we provided the same series of values of ratings as inputs again for SC 3 as we did for SC 1 and SC 2. From Fig. 7, we can see that when the ratings for Performance and Operation were 20% and 30% respectively, SC 3's output, i.e., Compensation is 59.99%. We can observe in the bar chart, the values for the output are decreasing when provided the same values for inputs as SC 1 and SC 2. Similarly, the Compensation was outputted as 35.66% when the ratings input were increased to 40% and 50% respectively. Likewise, when the customer ratings were increased to 50% and 60% respectively, the Compensation fell to 9.99%. Finally, when the customer ratings were at their highest, i.e., 80% and 90%, the Compensation output was just 3.88%.

From the bar chart, we can see that regardless of the kind of smart contract and the number of descriptors they have, when the two inputs, i.e., Performance and Operation are 20% and 30% respectively, the final defuzzified crisp value for Compensation is same or at least similar in all three smart contracts, SC 1, SC 2 and SC 3.

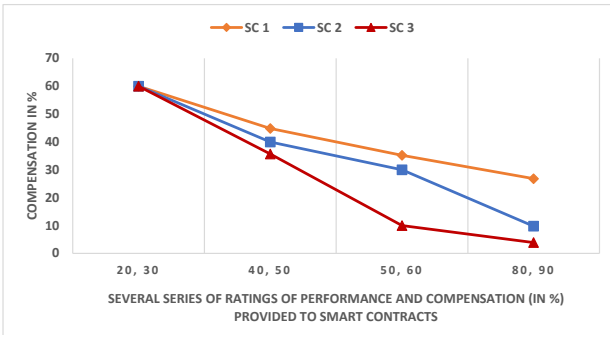

Fig. 8. SC 1 provides the least realistic and accurate output whereas SC 3 with the most realistic and accurate output.

- *Accuracy in the Smart Contract:*
  From our observations, we concluded that SC 3 with the highest number of descriptors has the highest accuracy compared to SC 1 and SC 2 because when the Performance and Operation are 80% and 90% respectively, the value of Compensation in SC 1 is 26.82% which is extremely high and unusual in real life. However, the output from SC 3 has the most accurate and realistic values of all defuzzified Compensation values compared to SC 1 and SC 2.

B. *Deployment Costs of different Smart Contracts*

We deployed all three smart contracts in Ropsten Testnet. We converted all ETH costs in United States Dollars (USD) and on August 16, 2:09 AM UTC, the conversion rate of 1 ETH was 3,315.44 USD. This data was provided by Morningstar for Currency and Coinbase for Cryptocurrency [13]. SC 1 had the lowest deployment cost, i.e., 14.02 USD.

Fig. 9. Deployment costs of SC 1, SC 2 and SC 3 in Ropsten Ethereum Testnet in USD.

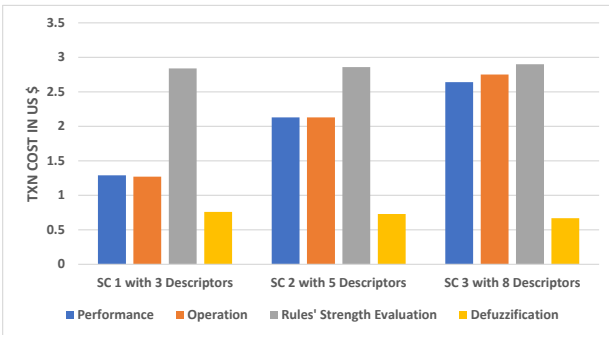

Fig. 10. TXN costs of major functions used in SC 1, SC 2 and SC 3 in USD.

The reason SC 1 had the lowest deployment cost was that it was the lightest program among all smart contracts as SC 1 only had three descriptors in its inputs, Performance, and Operation. As a result of only three descriptors, the number of membership functions for each corresponding descriptor was also lesser. However, SC 2 had five descriptors for both inputs and as a result, this smart contract had more membership functions assigned for its descriptors. Hence, the deployment cost for SC 2 was higher than SC 1, i.e. 21.24 USD. Finally, the highest deployment cost was for SC 3 because it had eight descriptors for inputs, Performance, and Operation as well as for the output, Compensation. The deployment cost for SC 3 was 30.11 USD. The deployment cost of a smart contract depends on its size. Therefore, the larger the smart contract, the higher the deployment cost is. In this case, the size of the smart contract was affected by the number of descriptors and their corresponding membership functions in the smart contract.

Additionally, we can also observe that we have a common ratio of 1.5 in this geometric series of deployment costs among the three smart contracts. The deployment cost of SC 2 is 1.5 times higher than the deployment cost of SC 1 and the same case for SC 3 and SC 2 as well. The reason we see this almost precise ratio between the deployment cost is the number of descriptors chosen for inputs in each smart contract, i,e, three, five, and eight.

C. Transaction Costs of major functions used in different Smart Contracts

We discuss the TXN costs incurred by the major four functions used in the smart contracts below.

- *Function for Performance:*
  This function is responsible for taking the crisp input from the customer for rating Performance of the company and then fuzzify the crisp input using its membership functions depending on how many descriptors it has. In SC 1, the TXN cost for Performance was 1.29 USD. The TXN cost for Performance increases to 2.13 USD in SC 2 and increases further to 2.64 USD in SC 3. The reason TXN cost is getting higher and higher is due to the increasing number of descriptors and their corresponding membership functions in smart contracts.

- *Function for Operation:*
  This is a similar function to Performance as Operation is our second input. Likewise, this function is responsible for taking the crisp input from the customer for rating Operation of the company and then fuzzify the crisp input using its membership functions depending on how many descriptors it has. Hence, the TXN costs in each smart contract for Operation is approximately the same as Performance as we can see in Fig. 10. The TXN cost for Operation in SC 1, SC 2, and SC 3 were 1.27 USD, 2.13 USD, and 2.75 USD respectively in an increasing fashion.

- *Function for Rules' Strength Evaluation:*
  Depending on the crisp ratings given by the customer, this function checks and selects the triggered rules in the rule-based system by calculating the membership values of each descriptor. Then, in the inference engine, when two antecedents are joined together using conjunctional operator, i.e., AND operator/MIN operator/Intersection operator, it compares between two membership values each time and finds the minimum value to evaluate the strength of all selected rules. Hence, because of this function's complexity, the TXN cost incurred is the highest as shown in Fig. 10 The TXN cost incurred by the function that measures the strength of the rules are approximately the same in all three smart contracts. The TXN costs for SC 1, SC 2, and SC3 are 2.84 USD, 2.86 USD, and 2.90 USD respectively. Even though the TXN cost for this function is highest in SC 3, higher in SC 2, and lowest in SC 1, there is not much significant difference in the TXN costs regardless of being from different smart contracts. The reason the TXN costs are almost similar in this case is that the number of rules this function checks to measure their strength is only four. Only four rules are selected for evaluation

of their strength because there are only two different inputs. Since we only have two inputs, only four rules are triggered and then selected from n^2, where is the number of inputs. Hence, the level of complexity of this function regardless of having a different number of descriptors in a different smart contract is the same.

- *Function for Defuzzification:*
  This function is responsible for finding the crisp output from the aggregated fuzzy set. The defuzzification technique that we have used for this work is Center of Gravity (COG) method as mentioned in the methodology section. The TXN cost of SC 1, SC 2, and SC 3 are 0.76 USD, 0.73 USD, and 0.67 USD respectively. The TXN cost incurred by this function is also almost exactly similar because of the usage and implementation of the same method across all three smart contracts.

IV. Future Work and Conclusion

Our future work would be to use other defuzzification techniques such as the Mean-Max method, Center of Sum (COS) method, and Weighted Average method in the smart contract and compare the accuracy of the defuzzification with the COG method. Future work also includes comparing the ground truth of smart contracts that implement various defuzzification methods with the legal department of the vendors that calculates and decides the compensation in their SLA.

We introduced a novel idea on how we can translate a vague legal contract by using fuzzy logic inside the smart contract that would be smart and intelligent enough to consider various human interpretations by taking several linguistic variables and descriptors into account. No matter how popular a vendor is, their SLAs can still be incomplete and ambiguous which puts a customer into a myriad of confusion and trouble. In this paper, we presented a fresh solution to an existing problem of ambiguity in legal contracts by taking a real-world legal contract and using a cheaper and faster approach, i.e., fuzzy logic to make the Ethereum-based smart contract smart and intelligent to decide the output based on several sets of different inputs. We also created three different smart contracts that employ the same logic and architecture but have different sets of linguistic variables to evaluate the performance, cost, and accuracy of each smart contract. The main purpose of this paper is to study the gray areas of natural language that create the fuzziness and vagueness in the legal contracts and how an Ethereum-based smart contract can be made even smarter and intelligent to easily handle this problem of ambiguity and multiple contract interpretations.

References

[1] S. Overby, L Greiner and L.G. Paul, "What is an SLA? Best practices for service-level agreements", [Online]. Available: https://www.cio.com/article/2438284/outsourcing-sla-definitions-and-solutions.html[Accessed: 8 Nov 2020]

[2] K. LaMance, "What is a Contract?," [Online]. Available: https://www.legalmatch.com/law-library/article/what-is-a-contract.html[Accessed: 10-Nov-2019].

[3] Customer Service Score Board [Online]. Available: https://www.customerservicescoreboard.com/CenturyLink [Accessed: 13-Sept-2019]

[4] Best Company [Online]. Available: https://bestcompany.com/isp/blog/1-star-isp-reviews [Accessed: 13-Sept-2019]

[5] N. Szabo, "The idea of smart contracts," *Nick Szabo's Papers and Concise Tutorials*, vol. 6, 1997

[6] L.A. Zadeh, "The concept of a linguistic variable and its application to approximate reasoning—I," *Information sciences*, 1975

[7] E.H. Mamdani, "Application of fuzzy algorithms for control of simple dynamic plant". *Proceedings of the Institution of Electrical Engineers*, 1974, 121 (12): 1585–1588. doi:10.1049/piee.1974.0328.

[8] E. J. Scheid, B. B. Rodrigues, L. Z. Granville and B. Stiller, "Enabling Dynamic SLA Compensation Using Blockchain-based Smart Contracts," *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, Arlington, VA, USA, 2019, pp. 53-61.

[9] R.B. Uriarte, H. Zhou, K. Kritikos, Z. Shi, Z. Zhao and R.D Nicola, "Distributed service-level agreement management with smart contracts and blockchain", *Concurrency and Computation Practice and Experience*, 2020

[10] R. B. Uriarte, R. de Nicola and K. Kritikos, "Towards Distributed SLA Management with Smart Contracts and Blockchain," *2018 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, Nicosia, 2018, pp. 266-271, doi: 10.1109/CloudCom2018.2018.00059.

[11] J. Zhao and B.K. Bose, "Evaluation of membership functions for fuzzy logic controlled induction motor drive," *IEEE 2002 28th Annual Conference of the Industrial Electronics Society. IECON 02*, 2002, pp. 229-234 vol.1, doi: 10.1109/IECON.2002.1187512.

[12] A. Jain and A. Sharma, "Membership function formulation methods for fuzzy logic systems: A comprehensive review," *Journal of Critical Reviews*, 2020

[13] Coinbase [Online]: Available: https://www.coinbase.com/ [Accessed: 16-Aug-2021]

[14] Spectrum SLA [Online]. Available: https://easyupload.io/6rse8f

[15] Ropsten Testnet Explorer [Online]. Available: https://ropsten.etherscan.io/ [Accessed: 15-Nov-2020]

[16] Solidity [Online]. Available: https://docs.soliditylang.org/en/v0.4.24/ [Accessed: 19-Sept-2020]

[17] Remix Ethereum IDE [Online]. Available: https://remix.ethereum.org/#optimize=false&runs=200&evmVersion=null& version=soljson-v0.7.4+commit.3f05b770.js [Accessed: 19-Sept-2020]

[18] Metamask Crypto Wallet [Online]. Available: https://metamask.io/ [Accessed: 25-Nov-2020]

[19] Truffle [Online]. Available: https://www.trufflesuite.com/ [Accessed: 15-Sept-2020]

[20] K. Upadhyay, R. Dantu, Z. Zaccagni and S. Badruddoja, "Is Your Legal Contract Ambiguous? Convert to a Smart Legal Contract," *2020 IEEE International Conference on Blockchain (Blockchain)*, 2020, pp. 273-280, doi: 10.1109/Blockchain50366.2020.00041.

[21] S. Badruddoja, R. Dantu, L. Widick, Z. Zaccagni and K. Upadhyay, "Integrating DOTS With Blockchain Can Secure Massive IoT Sensors," *2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2020, pp. 937-946, doi: 10.1109/IPDPSW50202.2020.00156.

[22] A. Salau, R. Dantu and K. Upadhyay, "Data Cooperatives for Neighborhood Watch," *2021 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, 2021, pp. 1-9, doi: 10.1109/ICBC51069.2021.9461056.

[23] S. Badruddoja, R. Dantu, Y. He, K. Upadhayay and M. Thompson, "Making Smart Contracts Smarter," *2021 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, 2021, pp. 1-3, doi: 10.1109/ICBC51069.2021.9461148.

[24] R. Chataut and R. Akl, "Optimal pilot reuse factor based on user environments in 5G Massive MIMO," *2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC), Las Vegas, NV*, 2018, pp. 845-851.

[25] S. Nakamoto. (2017). *Bitcoin: A Peer-to-Peer Electronic Cash System.* [Online]. Available: http://www.bitcoin.org/bitcoin.pdf

[26] K. Gidney, "Ambiguity is the killer of smart contracts -," *Enterprise Times*, 17-Jul-2018. [Online]. Available: https://www.enterprisetimes.co.uk/2018/07/18/ambiguity-is-the-killer-of-smart-contracts/. [Accessed: 23-Feb-2020].