SZAKDOLGOZAT

MISKOLCI EGYETEM

A szakdolgozat címe

Készítette:

Szakdolgozó Neve

Programtervező informatikus

Témavezető:

Témavezető neve

MISKOLC, 2020

Miskolci Egyetem
Gépészmérnöki és Informatikai Kar
Alkalmazott Matematikai Intézeti Tanszék **Szám:**

Szakdolgozat Feladat

Szakdolgozó Neve (N3P7UN) programtervező informatikus jelölt részére.

A szakdolgozat tárgyköre: kulcsszavak, hasonlók

A szakdolgozat címe: A dolgozat címe

A feladat részletezése:

Ide kell a feladatkiírásban szereplő szöveget betenni.

(Kisebb tagolás lehet benne, hogy jól nézzen ki.)

Témavezető: Témavezető neve (beosztása)

A feladat kiadásának ideje:

. . . .
szakfelelős

Eredetiségi Nyilatkozat

Alulírott **Szakdolgozó Neve**; Neptun-kód: `N3P7UN` a Miskolci Egyetem Gépészmérnöki és Informatikai Karának végzős Programtervező informatikus szakos hallgatója ezennel büntetőjogi és fegyelmi felelősségem tudatában nyilatkozom és aláírásommal igazolom, hogy *Szakdolgozat Címe* című szakdolgozatom saját, önálló munkám; az abban hivatkozott szakirodalom felhasználása a forráskezelés szabályai szerint történt.

Tudomásul veszem, hogy szakdolgozat esetén plágiumnak számít:

- szószerinti idézet közlése idézőjel és hivatkozás megjelölése nélkül;
- tartalmi idézet hivatkozás megjelölése nélkül;
- más publikált gondolatainak saját gondolatként való feltüntetése.

Alulírott kijelentem, hogy a plágium fogalmát megismertem, és tudomásul veszem, hogy plágium esetén szakdolgozatom visszautasításra kerül.

Miskolc, ...év .... .hó .... .nap

....

Hallgató

1.

szükséges (módosítás külön lapon)

A szakdolgozat feladat módosítása

nem szükséges

....

dátum témavezető(k)

2. A feladat kidolgozását ellenőriztem:

témavezető (dátum, aláírás):	konzulens (dátum, aláírás):
....	....
....	....
....	....

3. A szakdolgozat beadható:

....

dátum témavezető(k)

4. A szakdolgozat .... szövegoldalt

.... program protokollt (listát, felhasználói leírást)

.... elektronikus adathordozót (részletezve)

....

.... egyéb mellékletet (részletezve)

....

tartalmaz.

....

dátum témavezető(k)

5.

bocsátható

A szakdolgozat bírálatra

nem bocsátható

A bíráló neve: ....

....

dátum szakfelelős

6. A szakdolgozat osztályzata

a témavezető javaslata:	....
a bíráló javaslata:	....
a szakdolgozat végleges eredménye:	....

Miskolc, ...

a Záróvizsga Bizottság Elnöke

Contents

Chapter 1

Introduction

Presented in this work is an interpreter library/package for the Fuzzy Behavior Description Language (FBDL) implemented in the Octave programming language.

This initial description of the problem at hand can either be informative or utterly confusing for anyone reading it for the first time, simply because it entails many concepts perhaps still unknown to the reader. But it is quintessential to state the main topic being tackled as to not lose sight of it amidst the following discussions that will eventually lead up to the task itself. Anyone with a moderate to advanced knowledge in the field of programming can easily comprehend the workings of an interpreter if described properly, however simply showcasing that without any prior introduction to the underlying principles would still leave a fairly large gap in the reader's mind concerning the motivation behind such a language and also its use of a myriad of fuzzy logic related ideas. Therefore it is necessary to treat the subject as a whole and describe not only the technical implementations and results, but the theory as well, on which all of it is built.

With this in mind the work is split into two main sections along with this preceding foreword to allow for some clarifications and provide a greater description of the whole subject matter. The first half exposes the reader to core concepts related to fuzzy logic and slowly builds them up into its more complex and intricate applications. Also in this part the idea of behavior control and fuzzy state machines are presented along with various mathematical models to help with formalization; it concludes with the specifications of the aforementioned Fuzzy Behavior Description Language. The second half of the work delves into the implementation and inner workings of the interpreter. In the beginning, decisions regarding language specific implementation and other architectural considerations are discussed; possible alternatives are slightly touched upon. Following a general overview of the process of interpreting a language each stage and their operations are shown separately in detail along with possible corner cases that may require special attention and samples of unit tests to check the integrity and correct operation of the program. Finally the reader is provided with working examples of source code written in FBDL and also a demonstration of said code where the output of the interpreter can be verified.

The main difficulty lies in connecting the different parts of the overarching subject, so as to allow the reader to indulge in this work without getting lost consider the following short explanation as a guide to the various topics about to be presented.

The motivation behind creating a programming language, be it any kind, is always attributed to the existence of a specific problem it is trying to solve. This could range from low-level hardware management, such as those found in embedded systems, all the way to server-side applications and numerical analysis.

The language (FBDL) appearing in this work has been constructed to serve a particular application of fuzzy logic, namely that of behavior control. For example when a system, due to some event, reacts or behaves in a certain way based on predefined rules dictating its appropriate response. A fundamental property of fuzzy logic, essentially the fact that it is continuous, make it a prime candidate for such a use, since natural systems are hard, sometimes nigh impossible to accurately model with Boolean-logic.

Describing how such a system would operate, the rules it would follow and the actions it would take can be tricky to model with ordinary programming languages. Therefore an easier alternative was designed with the primary aim of facilitating the ease of use, particularly even if the user happens to lack any kind of previous formal experience in the field of programming. To further simplify the task of using this language it takes another useful property of fuzzy logic that arises from its continuity: we are able to describe the state of a fuzzy variable or in other words the degree to which it satisfies a certain statement with the help of natural language in contrast to using concrete numerical values.

There exits many applications for fuzzy logic and its extensions, some requiring complex calculations and methods that are quite conveniently present in scientific programming languages such as MATLAB and Octave. For this reason an interpreter library in these languages would provide great utility for programs already using fuzzy logic and open new opportunities for those seeking to venture into such areas.

These ideas constitute the majority of the work, so they shall be further examined at length in the following sections.

Chapter 2

Core concepts and previous research

2.1 Fuzzy logic

In order to gain a sound understanding of the idea of *fuzziness* we must first familiarize ourselves with the notion of fuzzy sets. The concept was first introduced and described by mathematician Lotfi A. Zadeh in 1965 as an extension to classical sets. The key difference between ordinary sets and fuzzy ones is simple: In the case of the former all elements are either a part of a set or not, where as in the world of fuzzy sets an element may belong to multiple sets. The measure of how much an element is part of a given set is referred to as its *degree of membership* and is calculated with the aid of the *membership function.*

2.1.1 Fuzzy set

The formal definition of a fuzzy set is as follows:

2.1. definíció. Let U, referred to as the *universe of discourse*, be a set containing all the elements we wish to describe and define $m : U \to [0, 1]$ as a membership function. The pair (U, m) forms a fuzzy set A in which $\forall x \in U$ the value given by $m(x)$ is called the degree of membership of x. The function $m(x)$ is equivalent to $\mu_a(x)$.

Taking the example from Claudio Moraga's *Introduction to fuzzy logic* (2005)[cite]: given the interval $[0, 10]$ of the real line as our universe of discourse and the statement "x is between 3 and 5", we may represent it with the function $\mu_{3-5} : [0, 10] \to [0, 1]$. Where $\mu_{3-5}(x) = 1$ if $3 \leq x \geq 5$, and $\mu_{3-5}(x) = 0$ otherwise as seen on fig1.a. This function describes the classical set $[3, 5]$. Consider now the statement "x is near 4". The proximity, or nearness to the number 4 can be represented as $4 - \epsilon$, given the assumption that ϵ is a sufficiently small positive real number. Values obtained by the continued subtraction of ϵ will have a decreasing *"degree of nearness"* to 4 until the value, and subsequently those smaller then itself, is no longer considered to be "near" the number 4. Repeating this experiment with $4 + \epsilon$ and the continued addition of ϵ will yield symmetric results. If we take the function $\mu_{near4} : [0, 10] \to [0, 1]$ to represent this statement just as previously, it becomes apparent that it cannot be of the same kind as μ_{3-5} (that lead to a classical set). If we assume that 3 and 5 are acceptable limit points for "near 4" and marking these as , $\alpha_{min} = 3, \alpha_{max} = 5, beta = 4$, then

$$\mu_{near4}(x) = \begin{cases} 0, & x \leq \alpha_{min} \text{ or } x \geq \alpha_{max} \\ 1, & x = \beta \\ \frac{x-\alpha_{min}}{\beta-\alpha_{min}}, & \alpha_{min} < x < \beta \\ \frac{\alpha_{max}-x}{\alpha_{max}-\beta}, & \beta < x < \alpha_{max}. \end{cases}$$

The function will be continuous and increasing for $3 < x < 4 and$ will be continuous and decreasing for$4 < x < 5$. Without further information, linear transitions will be chosen as shown in fig1.b. μ_{near4} represents a **fuzzy set**.

[figures of the previous example]

(a) Classical set

(b) Fuzzy set

Figure 2.1: Difference in steepness during the transition from 0 to 1.

Other than assigning values linearly to elements not fully contained inside a fuzzy set any kind of membership function can be utilized, but by far the most common is the previously mentioned linear way, which produces a trapeziod shape during visualization (Triangles arise, when the upper side of the trapezoid is a point). In terms of terminology the following expressions are defined regarding any given fuzzy set:

Core: Elements where the membership function is 1:

$$\text{core}(A) = \{x \in U | \mu_\alpha(x) \geq \alpha\}$$

Support: Elements where the membership function is greater than 0:

$$\text{support}(A) = \{x \in U | \mu_\alpha(x) > 0\}$$

Boundary: Elements where the membership function is between 0 and 1:

$$\text{boundary}(A) = \{x \in U | 0 < \mu_\alpha(x) < 1\}$$

Height: The height of the fuzzy set A is the maximum value taken on by the membership function:

$$\text{height}(A) = \{x \in U | \max(\mu_\alpha(x))\}$$

2.1.2 Various membership functions

An example of the different shapes that a membeship function may take include the following cases and their respective definitions:

Triangular (Same as in the above example, a special case of the trapezoid):

$$\mu_A(x) = \begin{cases} 0, & x \leq \alpha_{min} \text{ or } x \geq \alpha_{max} \\ 1, & x = \beta \\ \frac{x-\alpha_{min}}{\beta-\alpha_{min}}, & \alpha_{min} < x < \beta \\ \frac{\alpha_{max}-x}{\alpha_{max}-\beta}, & \beta < x < \alpha_{max}. \end{cases}$$

Trapezoidal (With the upper side (core) taking values from $[\beta_1, \beta_2]$):

$$\mu_A(x) = \begin{cases} 0, & x \leq \alpha_{min} \text{ or } x \geq \alpha_{max} \\ \frac{x-\alpha_{min}}{\beta_1-\alpha_{min}}, & \alpha_{min} < x < \beta_1 \\ \frac{\alpha_{max}-x}{\alpha_{max}-\beta_2}, & \beta_2 < x < \alpha_{max}. \end{cases}$$

Γ-membership function:

$$\mu_A(x) = \begin{cases} 0, & x < \alpha \\ 1 - e^{\gamma(x-a)^2}, & \alpha_{min} < x < \beta_1 \end{cases}$$

S-membership function:

$$\mu_A(x) = \begin{cases} 0, & x \leq \alpha_{min} \text{ or } x \geq \alpha_{max} \\ 2\left(\frac{x-\alpha_{min}}{\alpha_{max}-\alpha_{min}}\right) pow2, & \alpha_{min} < x < \beta \\ 1 - 2\left(\frac{x-\alpha_{min}}{\alpha_{max}-\alpha_{min}}\right) pow2, & \beta < x < \alpha_{max}. \end{cases}$$

Logistic function:

$$\mu_A(x) = \frac{1}{1+e^{-\gamma x}}$$

Exponential-like function:

$$\mu_A(x) = \frac{1}{1+\gamma(x-\beta)^2}$$

where $\gamma > 1$.

Gaussian function:

$$\mu_A(x) = e^{-\alpha(x-\beta)^2}$$

Besides these any function that fits the intended purpose of characterizing a certain fuzzy set is acceptable and it is left up to the discretion of experts in the given field to decide which one is most appropriate.

[figures some previous functions]

2.1.3 Dealing with multiple fuzzy sets

There are some cases, where precise numerical measurements might not be required or even be detrimental, for example stating someone's age as being 17 years 32 days and 8 hours old, does not necessarily demand such accuracy. It is much more sensible to describe that person simply as young. This notion of the use of words in our statements instead of numerical values was introduced by Zadeh in 1975 and is called a **linguistic variable**. The varying values taken by such a variable can be described by **linguistic terms**, such as *low, middle, high, very small, average, large*, meaning we are able to take advantage of natural language, thus making it easier to work with.

For a simple example consider one's age as a variable and the two sets: young and old. A person who is 5 years of age is considered very young and not at all old, similarly someone in their twenties may be called young, but slightly old as well, however a middle aged individual of 43 years is neither very young nor very old, but rather an even mix of both. Representing the two fuzzy sets, young and old, on fig3 we can see they overlap.

Any value at this given interval of overlay is a linear combination of, in this particular case, both of these fuzzy sets. The number of sets can, of course, be extended and then the values at these intersection would be a linear combination of all the defined sets, given that they overlap. From giving proper definitions of how to operate on these linguistic variables arises the notion of **fuzzy logic** and subsequently it serves as the basis for many advanced concepts such as: inference, fuzzy decision making and fuzzy control.

[Overlapping fuzzy sets with point x demonstrating linear combination of values]

2.2 Applications of fuzzy logic

Many fields make use of fuzzy logic and all the differing unique characteristics from ordinary sets that it has to offer. Fuzzy logic is best suited for problems that may be hard to define or model precisely with Boolean-logic. A collection of some notable examples of already tried and implemented solutions explored by [cite Makkar] are discussed in this section.

2.2.1 Healthcare

Due to the intrinsic non-linearity of biomedical systems it is difficult to accurately model various processes. Regulation of blood pressure in the case of medical patients has been tested with the help of a real time drug delivery system that used an integrated fuzzy controller. Separately, it has also been shown that test reports yield estimates of likelihood rather than confirmation of presence or absence of a disease, hence these empirical estimates can be treated as the output of a membership function and used as such in fuzzy inference modelling.

2.2.2 Chemical Science

A fuzzy control system was used to both apply current to a series of anodes protecting an underground pipeline and to minimize the system's power consumption. For comparison the system used 126 *fuzzy rules* (further discussed in the following chapter) and an empirically adjusted membership function to optimize the model. Another study conducted in relation to pH measurement in waste water adopted fuzzy logic to calculate errors and acceptable levels of pH in the data.

2.2.3 Optimization problems (Operations Research)

Pappis and Mamdani (1977) applied fuzzy logic to control the flow of traffic at an intersection of two one-way streets and minimize traffic obstruction. Teodorovic and

Kalie (1996) experimented with fuzzy logic based decisions for choosing the mode of transportation in order to minimize both the cost of travelling and the travel time. .Jarkko and Esko (2003) had applied fuzzy logic to minimize the waiting time and risk of collisions during the operation of traffic signals .

2.3 Fuzzy rules

The behavior of a system can be represented by a simple model if we consider only its relevant aspects. Such a construct makes use of a set of rules in the "if - then" form. Fuzzy rules are categorized by two major types, Mamdani fuzzy rules and Takagi-Sueno fuzzy rules [cite]. In the general form of a fuzzy rule a list of antecednets is followed by a number of consquents such that:

$$\text{if} < \mathit{antecedent}_1 > \text{and} \ldots \text{and} < \mathit{antecedent}_n >, \text{then} < \mathit{consequent}_1 > \text{and} \ldots \text{and} < \mathit{consequent}_n >$$

where the *antecedent* is of the form v_1 is S_1 and the consequent z_1 is W_1 respectively. v_i is an input variable belonging to the input fuzzy set S_i and z_j is an output variable of the output fuzzy set W_j. In the case of Takagi-Sueno fuzzy rules the consequents are replaced with functions of the input variables so that $z_j = f_p(v_1, \ldots, v_i)$, where f_p is any real function.

2.3.1 Fuzzy inference

In order to use fuzzy logic for any sort of application we must first consider how to integrate it with existing Boolean-logic. More precisely, we are interested in a solution that operates on linguistic variables and an outcome that relies solely on fuzzy rules along with linguistic terms. Since the input variables to any given system are usually not fuzzy ones, they must be converted to satisfy this requirement in order to then later be used within the fuzzy application. This first step is called **fuzzification** and as the name implies we make sure to supply our further calculations with variables of the correct form. By taking the desired element $x \in U$ from our universe of discourse and some fuzzy set A we convert x to a membership function value, given by $\mu_A(x)$. Repeating this procedure for every element we wish to utilize yields a degree of membership for each one, therefore translating all discrete inputs to fuzzy ones.

Now that we have a number of fuzzy variables to work with the next step is to apply predefined rules of the form described in the previous section. This step is referred to as **inference** and mathematically it is a mapping of the antecedents (input variables) to the consequents (output variables), resulting in an output fuzzy set. A degree of membership of any variable in this resultant set is depended upon the degree of membership of values in the input set or sets that have been defined by the given rule. That is to say, let A and B be fuzzy set of the antecedent and C of the consequent respectively and X, Y, Z linguistic terms. Then, according to the rule

$$\text{if } A \text{ is } X \text{ and } B \text{ is } Y \text{ then } C \text{ is } Z$$

the inference process will calculate the output fuzzy set C based on the known values of $\mu_A(X)$ and $\mu_B(Y)$.

In a similar manner to the calculation of the membership functions there are a multitude of methods which are applicable in determining the output, these are functions that do the actual mapping between the two sets of antecedent and consequent. This process of **fuzzy rule interpolation** (FRI) entails a great number of ways in which different aspects and characteristics of the membership functions are considered and thereafter the calculations made, each having their advantage and difficulties.

[Figures and further description of the FRI process] [Greater detail about the calculation of the area under the combined membership function shape, determining the core's position (eg.: cog), different methods of FRI]

[Defuzzification and graphical representation of the aggregated value given by the area of the interpolated membership functions]

2.4 State machines and fuzzy behavior

[Fuzzy logic controllers and connection to state machines] [Simple example]

2.5 Mathematical model

[Figure by Sz. Kovacs about the construction and inner workings of fuzzy automata] [Depiction of interconnected components and their functions]

2.6 Fuzzy Behavior Description Language

[Motivation behind the fuzzy behavior description language] [Language specifications and requirements: syntax, grammar with BNF and railroad diagram] [Previous works on an FBDL interpreter, Javascript library]

Chapter 3

Design

3.1 Interpreters and their operation

[General overview of interpreters and basic principles] [Short description of parts and pros against compilers]

3.2 Implementation requirements

[Due to past research the need arose for such an interpreter in MATLAB/Octave]

3.3 Lexical Analyzer

[Data structures and functional dependency between modules of the tokenizer] [Valid token list, (keywords, strings, numbers, terminals?, etc.)] [Considerations, preparation of text] [Method of analyzing and correctly checking each token] [Room for possible future extensions (string escapes, terminals and special symbols, dominates and other keywords)]

3.4 Parser

[Overview of each grammar rule] [Specific implementation to parse the given grammar with recursive descent parsing technique] [Building the syntax tree and considerations for the data structures being used] [Additional semantical checks after parsing] [Room for possible future extensions: warnings for missing elements, parsing optional arguments, rule domination hierarchy, extension of the existing grammar?, parsing techniques required in case of certain grammar modification, eg.: precedence climbing for evaluation of mathematical expressions instead of using constants]

3.5 Engine

[Calculation of fuzzy rule interpolation and inference of resultant values] [Operations based on the mathematical fuzzy automaton] [Behavior control and further reusing output values]

3.6 Error handling

[Built in Octave error handling, but a generic one should be used] [To avoid code entanglement and logically difficult to understand fault checking] [Reporting and error messages for helping find the apparent problem within the input code]

Chapter 4

Megvalósítás

Ez a fejezet mutatja be a megvalósítás lépéseit. Itt lehet az esetlegesen előforduló technikai nehézségeket említeni. Be lehet már mutatni a program elkészült részeit.

Meg lehet mutatni az elkészített programkód érdekesebb részeit. (Az érdekesebb részek bemutatására kellene szorítkozni. Többségében a szöveges leírásnak kellene benne lennie. Abból lehet kiindulni, hogy a forráskód a dolgozathoz elérhető, azt nem kell magába a dolgozatba bemásolni, elegendő csak behivatkozni.)

A dolgozatban szereplő forráskódrészletekhez külön vannak programnyelvenként stílusok. Python esetében például így néz ki egy formázott kódrészlet.

```python
import sys

if __name__ == '__main__':
    pass
```

A stílusfájlok a `styles` jegyzékben találhatók. A stílusok között szerepel még C++, Java és Rust stílusfájl. Ezek használatához a `dolgozat.tex` fájl elején `usepackage` paranccsal hozzá kell adni a stílust, majd a stílusfájl nevével megegyező környezetet lehet használni. További példaként C++ forráskód esetében ez így szerepel.

```cpp
#include <iostream>

class Sample : public Object
{
    // An empty class definition
}
```

Stílusfájlokból elegendő csak annyit meghagyni, amennyire a dolgozatban szükség van. Más, C szintaktikájú nyelvekhez (mint például a JavaScript és C#) a Java vagy C++ stílusfájlok átszerkesztésére van szükség. (Elegendő lehet csak a fájlnevet átírni, és a fájlban a környezet nevét.)

Nyers adatok, parancssori kimenetek megjelenítéséhez a `verbatim` környezetet lehet használni.

```
$ some commands with arguments
1 2 3 4 5
$ _
```

A kutatás jellegű témáknál ez a fejezet gyakorlatilag kimaradhat. Helyette inkább a fő vizsgálati módszerek, kutatási irányok kaphatnak külön-külön fejezeteket.

Chapter 5

Tesztelés

A fejezetben be kell mutatni, hogy az elkészült alkalmazás hogyan használható. (Az, hogy hogyan kell, hogy működjön, és hogy hogy lett elkészítve, az előző fejezetekben már megtörtént.)

Jellemzően az alábbi dolgok kerülhetnek ide.

- Tesztfuttatások. Le lehet írni a futási időket, memória és tárigényt.
- Felhasználói kézikönyv jellegű leírás. Kifejezetten a végfelhasználó szempontjából lehet azt bemutatni, hogy mit hogy lehet majd használni.
- Kutatás kapcsán ide főként táblázatok, görbék és egyéb részletes összesítések kerülhetnek.

Chapter 6

Összefoglalás

Hasonló szerepe van, mint a bevezetésnek. Itt már múltidőben lehet beszélni. A szerző saját meglátása szerint kell összegezni és értékelni a dolgozat fontosabb eredményeit. Meg lehet benne említeni, hogy mi az ami jobban, mi az ami kevésbé jobban sikerült a tervezettnél. El lehet benne mondani, hogy milyen további tervek, fejlesztési lehetőségek vannak még a témával kapcsolatban.

CD Használati útmutató

Ennek a címe lehet például *A mellékelt CD tartalma* vagy *Adathordozó használati útmutató* is.

Ez jellemzően csak egy fél-egy oldalas leírás. Arra szolgál, hogy ha valaki kézhez kapja a szakdolgozathoz tartozó CD-t, akkor tudja, hogy mi hol van rajta. Jellemzően elég csak felsorolni, hogy milyen jegyzékek vannak, és azokban mi található. Az elkészített programok telepítéséhez, futtatásához tartozó instrukciók kerülhetnek ide.

A CD lemezre mindenképpen rá kell tenni

- a dolgozatot egy `dolgozat.pdf` fájl formájában,
- a LaTeX forráskódját a dolgozatnak,
- az elkészített programot, fontosabb futási eredményeket (például ha kép a kimenet),
- egy útmutatót a CD használatához (ami lehet ez a fejezet külön PDF-be vagy MarkDown fájlként kimentve).