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

Tartalomjegyzék

1. fejezet

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.

2. fejezet

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*.

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 $\mu_a(x)$.

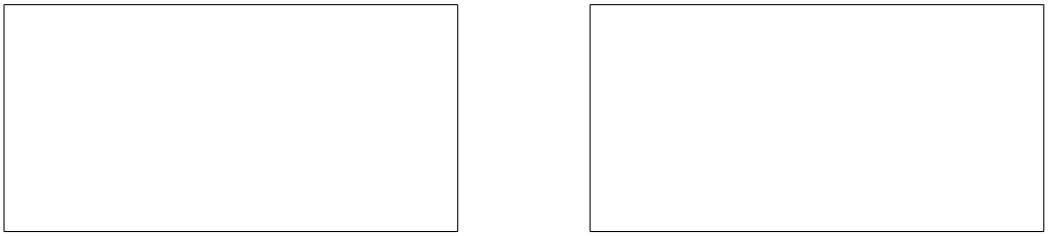
Taking the example from Claudio Moraga's *Introduction to fuzzy logic* (2005)[cite]: given the interval $[0, 10]$ of the real line as our universe of discourse and the statement „x is between 3 and 5", we may represent it with the function $\mu_{3-5} : [0, 10] \to [0, 1]$. Where $\mu_{3-5}(x) = 1$ if $3 \leq x \geq 5$, 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", then

$$\mu_{near4}(x) = \begin{cases} 0, & x < 3 \text{ or } x > 5 \\ 1, & x = 4 \\ \text{frac}(x), & 3 < x < 4 \\ 1 - \text{frac}(x), & 4 < x < 5. \end{cases}$$

The function will be continuous and increasing for $3 < x < 4and$ 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**.

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.

figure of the previous example



(a) Classical set (b) Fuzzy set

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

overlapping fuzzy sets different membership functions (non-linear)

2.2. Applications

2.3. Fuzzy rules

2.4. State machines and fuzzy behavior

2.5. Mathematical model

2.6. Fuzzy Behavior Description Language

2.7. A fejezet célja

Ez a fejezet még nem a saját eredményekkel foglalkozik, hanem bemutatja, mi a problémakör, milyen módszerekkel, milyeneredményeket sikerült elérni eddig másoknak.

A hivatkozások jelentős része ehhez a fejezethez szokott kötődni. (Egy hivatkozás például így néz ki [1].) Itt lehet bemutatni a hasonló alkalmazásokat.

2.8. Tartalom és felépítés

A fejezet tartalma témától függően változhat. Az alábbiakat attól függően különböző arányban tartalmazhatják.

- Irodalomkutatás. Amennyiben a dolgozat egy módszer kidolgozására, kifejlesztésére irányul, akkor itt lehet részletesen végignézni (módszertani vagy időrendi bontásban), hogy az eddigiekben milyen eredmények születtek a témakörben.

- Technológia. Mivel jellemzően kutatásról vagy szoftverfejlesztésről van szó, ezért annak a jellemző elemeit, technikai részleteit itt kell bemutatni. Ez tehát egy módszeres bevezetés ahhoz, hogy ha valaki nem jártas a témakörben, akkor tudja, hogy a dolgozat milyen aktuálisan elérhető eredményeket, eszközöket használt fel.
- Piackutatás. Bizonyos témáknál új termék vagy szolgáltatás kifejlesztése a cél. Ekkor érdemes annak alaposan utánanézni, hogy aktuálisan milyen eszközök érhetők el a piacon. Ez szoftverek esetében a hasonló alkalmazások bemutatását, táblázatos formában történő összehasonlítását jelentheti. Szerepelhetnek képek és észrevételek a viszonyításként bemutatott alkalmazásokhoz.
- Követelmény specifikáció. Külön szakaszban érdemes részletesen kitérni az elkészítendő alkalmazással kapcsolatos követelményekre. Ehhez tartozhatnak forgatókönyvek (*scenario*-k). A szemléletesség kedvéért lehet hozzájuk képernyőkép vázlatokat is készíteni, vagy a használati eseteket más módon szemléltetni.

2.9. Amit csak említés szintjén érdemes szerepeltetni

Az olvasóról annyit feltételezhetünk, hogy programozásban valamilyen szinten járatos, és a matematikai alapfogalmakkal sem ebben a dolgozatban kell megismertetni. A speciális eszközök, programozási nyelvek, matematikai módszerekk és jelölések persze jó, hogy ha említésre kerülnek, de nem kell nagyon belemenni a közismertnek tekinthető dolgokba.

3. fejezet

Tervezés

Itt kezdődik a dolgozat lényegi része, úgy értve, hogy a saját munka bemutatása. Jellemzően ebben szerepelni szoktak blokkdiagramok, a program struktúrájával foglalkozó leírások. Ehhez célszerű UML ábrákat (például osztály- és szekvenciadiagramokat) használni.

Amennyiben a dolgozat inkább kutatás jellegű, úgy itt lehet konkretizálni a kutatási módszertant, a kutatás tervezett lépéseit, az indoklást, hogy mit, miért és miért pont úgy érdemes csinálni, ahogyan az a későbbiekben majd részletezésre kerül.

Ebben a fejezetben az implementáció nem kell, hogy túl nagy szerepet kapjon. Ez még csak a tervezési fázis. (Nyilván ha olyan a téma, hogy magának az implementációnak a módjával foglalkozik, adott formális nyelvet mutat be, úgy a kódpéldákat már innen sem lehet kihagyni.)

3.1. Táblázatok

Táblázatokhoz a `table` környezetet ajánlott használni. Erre egy minta a 3.1. táblázat. A hivatkozáshoz az egyedi `label` értéke konvenció szerint `tab:` prefixszel kezdődik.

3.1. táblázat. Minta táblázat. A táblázat felirata a táblázat felett kell legyen!

a	b	c
1	2	3
4	5	6

3.2. Ábrák

Ábrákat a `figure` környezettel lehet használni. A használatára egy példa a 3.1. ábrán látható. Az `includegraphics` parancsba Az ábrák felirata az ábra alatt kell legyen. Az ábrák hivatkozásához használt nevet konvenció szerint `fig:`-el célszerű kezdeni.

3.3. További környezetek

A matematikai témájú dolgozatokban szükség lehet tételek és bizonyításaik megadására. Ehhez szintén vannak készen elérhető környezetek.

3.1. ábra. A Miskolci Egyetem címere.

3.1. definíció. Ez egy definíció

3.2. lemma. *Ez egy lemma*

3.3. tétel. *Ez egy tétel*

Bizonyítás. Ez egy bizonyítás □

3.4. következmény. *Ez egy tétel*

3.5. megjegyzés. Ez egy megjegyzés

3.6. példa. Ez egy példa

4. fejezet

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.

```
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.

```
#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.

5. fejezet

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.

6. fejezet

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.

Irodalomjegyzék

[1] James H Coombs, Allen H Renear, and Steven J DeRose. Markup systems and the future of scholarly text processing. *Communications of the ACM*, 30(11):933–947, 1987.

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).