Soft Computing: Project Documentation

Demonstartion of Basic Fuzzy Set Operations

December 4, 2016

Author: Attila Večerek, xvecer17@stud.fit.vutbr.cz
Faculty of Information Technolgy
Brno University of Technology

Contents

1 Introduction

This report discusses and presents the design and implementation of a desktop application capable of demonstrating the 3 basic erations over fuzzy sets:

- union,
- intersection,
- and complement.

The assignment has stated to demonstrated these operations over *discrete* fuzzy sets. However, this report discusses the solution of operations over **continuous** fuzzy sets.

2 Theory

In this section, we will briefly cover the theoretical knowledge needed for this project.

2.1 Fuzzy set

A **fuzzy set** is a generalization of an ordinary set by allowing a **degree** (or **grade**) **of membership** for each element. A membership degree is a real number on [0, 1] [1].

2.2 Membership function

The **membership function** of a set maps each element to its degree. Having the elements of the set being associated with a degree of membership is the foundation of fuzzy sets as well as fuzzy systems [1].

2.3 Universe

The **universe** represents a regular set of elements that have an associated degree of membership in a given fuzzy set.

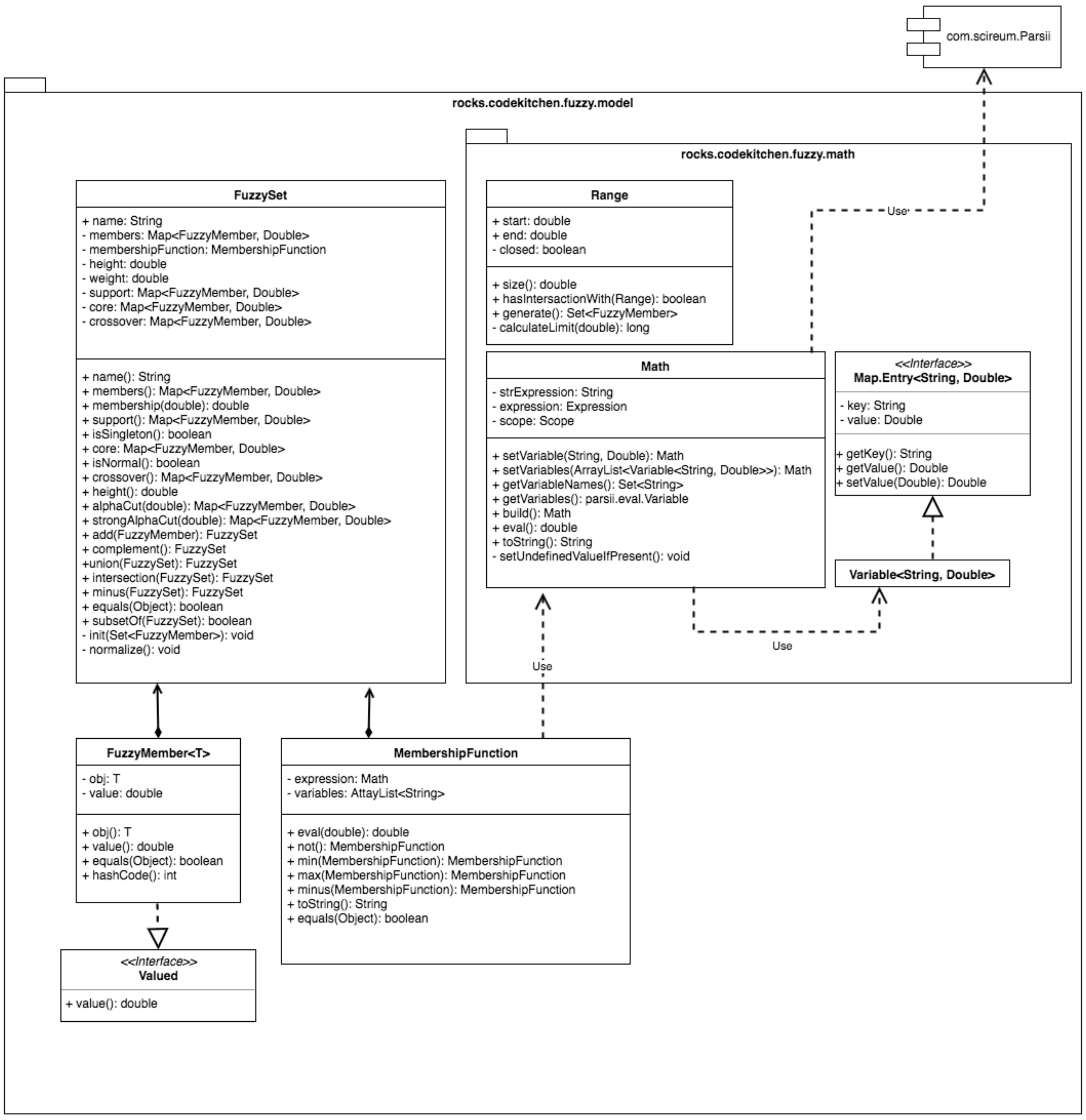
3 Design

The application has been designed to satisfy the following requirements from the perspective of a user:

1. it should be possible to define a common universe to the fuzzy sets,
2. it should be possible to define upto 2 fuzzy sets by their respective names (optionally) and membership functions,
3. it should be possible to define the membership function as a mathematical expression,
4. and it should be possible to run the operations only in cases, when all the necessary input has been given.

3.1 Logic

The logic of the application is responsible for all the computation required for parsing the user input as well as processing the fuzzy operations. The fuzzy model consists of 3 main classes: `FuzzySet`, `FuzzyMember` and `MembershipFunction`. The MembershipFunction class is dependant on the `Math` subpackage which is responsible for parsing the membership function input by the user. In order to parse and evaluate math expressions, the **com.scireum.Parsii** open-source module has been chosen as a project dependency [2].



com.scireum.Parsii
rocks.codekitchen.fuzzy.model
rocks.codekitchen.fuzzy.math
Use
FuzzySet
+ name: String
- members: Map<FuzzyMember, Double>
- membershipFunction: MembershipFunction
- height: double
- weight: double
- support: Map<FuzzyMember, Double>
- core: Map<FuzzyMember, Double>
- crossover: Map<FuzzyMember, Double>
+ name(): String
+ members(): Map<FuzzyMember, Double>
+ membership(double): double
+ support(): Map<FuzzyMember, Double>
+ isSingleton(): boolean
+ core: Map<FuzzyMember, Double>
+ isNormal(): boolean
+ crossover(): Map<FuzzyMember, Double>
+ height(): double
+ alphaCut(double): Map<FuzzyMember, Double>
+ strongAlphaCut(double): Map<FuzzyMember, Double>
+ add(FuzzyMember): FuzzySet
+ complement(): FuzzySet
+union(FuzzySet): FuzzySet
+ intersection(FuzzySet): FuzzySet
+ minus(FuzzySet): FuzzySet
+ equals(Object): boolean
+ subsetOf(FuzzySet): boolean
- init(Set<FuzzyMember>): void
- normalize(): void
Range
+ start: double
+ end: double
- closed: boolean
+ size(): double
+ hasIntersactionWith(Range): boolean
+ generate(): Set<FuzzyMember>
- calculateLimit(double): long
Math
- strExpression: String
- expression: Expression
- scope: Scope
+ setVariable(String, Double): Math
+ setVariables(ArrayList<Variable<String, Double>>): Math
+ getVariableNames(): Set<String>
+ getVariables(): parsii.eval.Variable
+ build(): Math
+ eval(): double
+ toString(): String
- setUndefinedValueIfPresent(): void
<<Interface>>
Map.Entry<String, Double>
- key: String
- value: Double
+ getKey(): String
+ getValue(): Double
+ setValue(Double): Double
Variable<String, Double>
Use
Use
FuzzyMember<T>
- obj: T
- value: double
+ obj(): T
+ value(): double
+ equals(Object): boolean
+ hashCode(): int
MembershipFunction
- expression: Math
- variables: AttayList<String>
+ eval(double): double
+ not(): MembershipFunction
+ min(MembershipFunction): MembershipFunction
+ max(MembershipFunction): MembershipFunction
+ minus(MembershipFunction): MembershipFunction
+ toString(): String
+ equals(Object): boolean
<<Interface>>
Valued
+ value(): double

Figure 1: The logic of the application

Parsii has been chosen because of its ease of use and integration as well as it is one of the fastest expression evaluators implemented in Java based on an article published by http://www.javacodegeeks.com/ [3].

To define a discrete universe to the fuzzy sets, the user needs to specify a range or several ranges of decimal numbers, i.e. the range `1..10` stands for a left-closed and right-open interval between 1 and 10, whereas `1...10` marks a closed interval. Notice the difference in the number of dots in the respective examples. The universe can also be defined by a list of ranges separated by a sequence of characters described by the regular expression `,\\s^{+}`.

3.2 Front-end

The application follows the design pattern called MVC which stands for Model-View-Controller. The Figure 2 represents the dependencies between the packages and the communication flow between the instances of the respective classes. The `MainController` consists of an instance of `FuzzyLineChart` to be able to operate with it (`initialize` and `update` operations). It also communicates with the `ApplicationHelper` class which is responsible for creating the user-defined universe. Both the `MainController` and `ApplicationController` utilize the `Model` package described in the above section to be able to create the necessary fuzzy sets and run the user-selected operations above them. The `FuzzyLineChart` class wraps a **JavaFX LineChart** and populates it with the required datapoints obtained by the `MainController`, thus rendering the expected chart output.

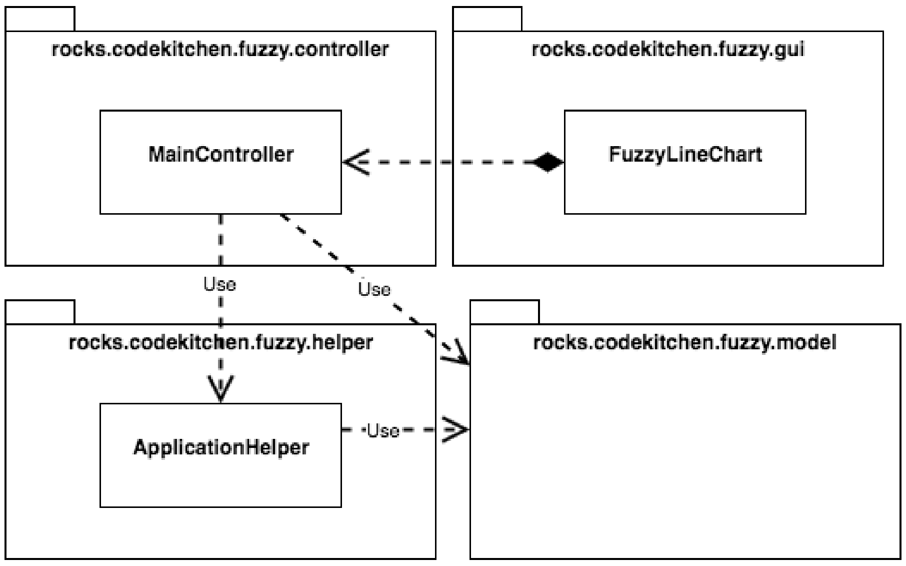
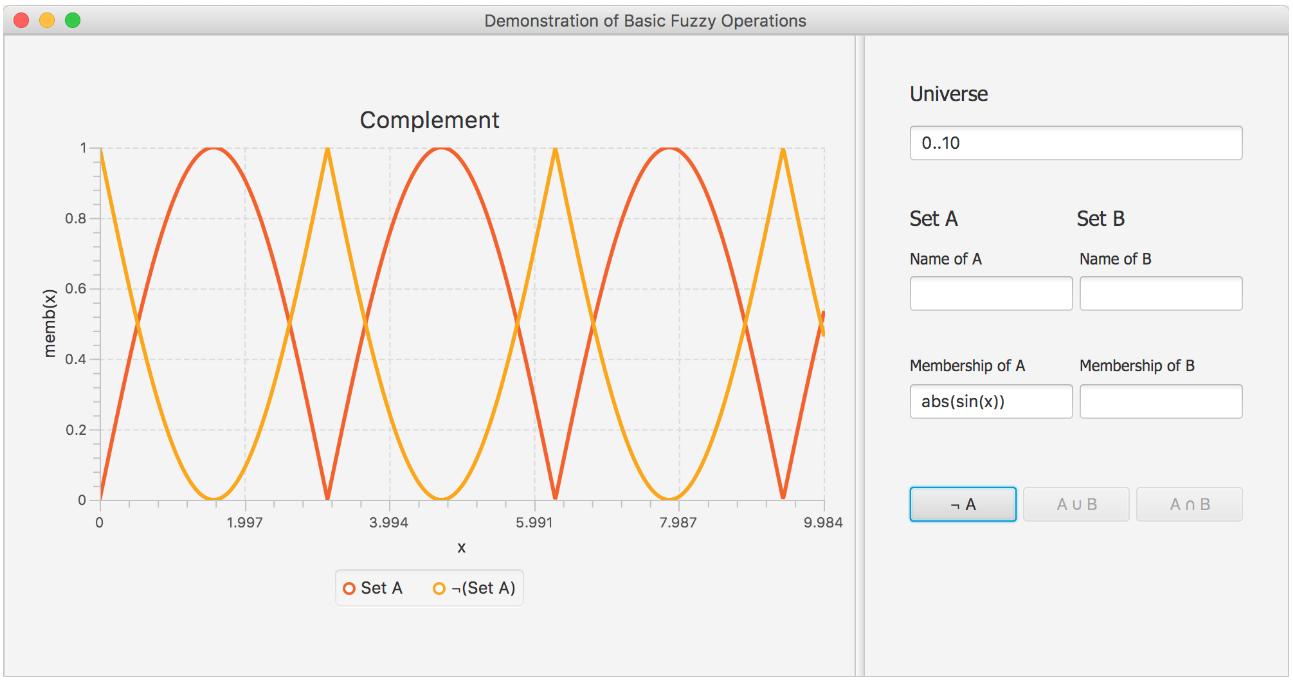

rocks.codekitchen.fuzzy.controller
MainController
rocks.codekitchen.fuzzy.gui
FuzzyLineChart
Use
Use
rocks.codekitchen.fuzzy.helper
ApplicationHelper
Use
rocks.codekitchen.fuzzy.model

Figure 2: Frontend design

3.2.1 User Experience

A good user experience is never bad. With that and the best practices in mind, the user interface has been divided into two integral parts as it is seen in Figure 3. The bigger part of the screen is assigned to the description of the fuzzy operation and its result. On the right side of the screen, there is the pane that serves the purpose of inputting all the information in relation to the fuzzy sets and the operations to be done over them.



Universe: 0..10

Membership of A: abs(sin(x))

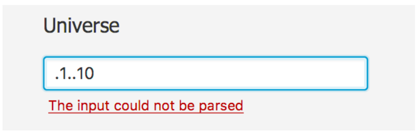
Figure 3: The graphical user interface

Error messages There are 4 possible error messages in relation to invalid input:

1. the definition of the universe is not parsable,
2. the membership function is not parsable,
3. the membership function cannot be evaluated,
4. the same name has been given to the fuzzy sets.

A valid universe definition can be either a range, list of several ranges or an enumeration of decimal numbers. In all other cases it invokes an error message which is a hyperlink leading to the Git repository of the project and its `README` file as seen in Figure 4.

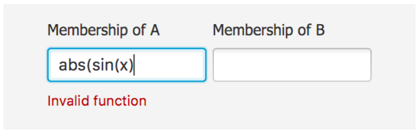
The membership function is parsable and evaluable if it is an expression parsable by the `com.scireum.Parsii` module and it contains exactly one variable, i.e. the `x` in expression `abd(sin(x))`. Otherwise the respective errors would rise up as seen in Figure 5 and 6.


Universe

.1..10

The input could not be parsed

Figure 4: Parse error U

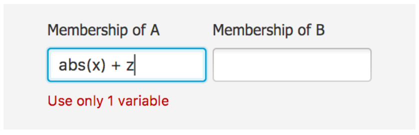

Membership of A

Membership of B

abs(sin(x)

Invalid function

Figure 5: Parse error memb()


Membership of A

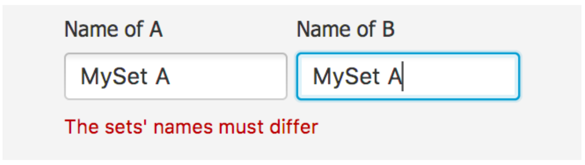
Membership of B

abs(x) + z

Use only 1 variable

Figure 6: Eval error memb()

The fourth error message occurs when both fuzzy sets have been assigned with the same name as it is shown in Figure 7.


Name of A

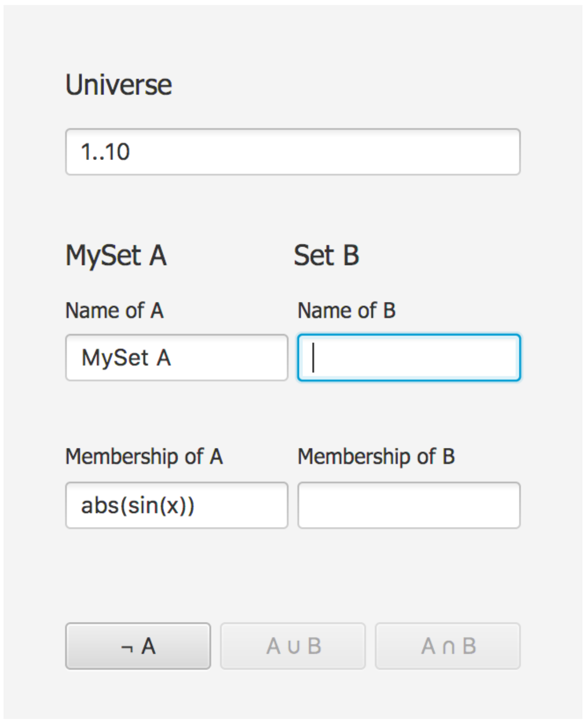
MySet A

Name of B

MySet A

The sets' names must differ

Figure 7: Identical set name error


Universe

1..10

MySet A

Set B

Name of A

MySet A

Name of B

Membership of A

abs(sin(x))

Membership of B

Figure 8: Eval error memb()

Action Buttons In order to prevent any further errors, the respective action buttons are enabled only in case that all the required fields have been filled out and the input information is valid. The Figure 8 shows that the binary operations are disabled when only the first fuzzy set has been set up.

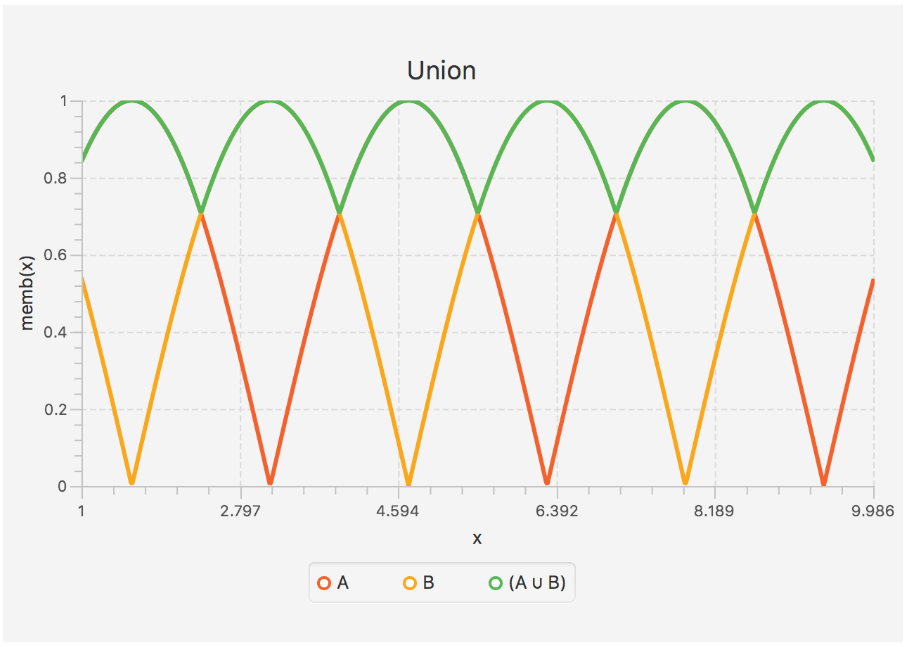


Figure 9: The line chart

Line Chart The line chart is the most important element of the UI. It presents the result of the basic operations over the specified fuzzy sets. The following information are always shown to the user, so he is always aware of the currently presented fuzzy operation and its details: *title of the operation*, the *line chart* and the *legend*. An example can be seen in Figures 3 and 9.

4 Implementation

The desktop application has been implemented in Java 8. It uses the `com.scireum.Parsii` module to parse and evaluate mathematical expressions. The GUI is implemented using the `JavaFX 8` library. The tests are written in `Groovy` using the `Spock` framework.

As it has already been mentioned, the application is capable of operating with fuzzy sets defined by a *continuous* universe. In this section we will describe and discuss the implementation of exactly this issue.

4.1 Discrete vs. continuous

A digital computer is not able to work with continuous data in their raw form. It uses sampling with an approriate sampling rate to create a set of discrete data it can finally process and operate with. However, the sampling frequency can be set very high, so the produced discrete data may seem for the human eye as continuous. That is exactly the same procedure our desktop application has adopted.

When a user specifies the universe with a range, e.g. `0...1`, the application has to decide about the appropriate sampling rate. Too high sampling could be very memory inefficient. On the other hand, too low sampling rate would cause a non-linear member function to loose its "curviness". To find an optimum, the application examines the width of the linechart in pixels. That number is considered as the maximal number of datapoints presented on the chart. Assume the formerly defined universe with the range of `0...1` and a line chart of a 100 pixels width. It means that the specified interval needs to be discretized into 100 evenly distributed data points across that interval.

The next step is to compute the *precision*. In the above example of a closed range the computation would look as following. Let us assume an unrealistic case of a 5 pixels wide linechart to make the example trivial. The step would be then equal to $\dfrac{1-0}{5-1} = 0.25$. If it were a left-closed and right-open range, the computation would be as follows: $\dfrac{1-0}{5} = 0.2$.

4.2 The algorithm

The application operates with only as many datapoints as needed in order to be memory and computation efficient. The following steps describe the workflow that takes place since the start until the end which in our case means that the line chart is populated with data and presented to the user.

1. The user enters a `String` as an input in the universe definition text field.

   (a) The `ApplicationHelper` class checks, whether the input matches an acceptable format.

(b) If the given input has been evaluated as invalid, the `MainController` sets the error message visible. Otherwise, it does not do anything.

2. The user fills out the member function text field.

   (a) The `MainController` invokes a method chain leading to the `Parsii` module which parses the member function.

   (b) If the given input is not parsable or contains more than 1 variable, an appropriate error message will be set visible.

3. The user clicks the Complement operation action button.

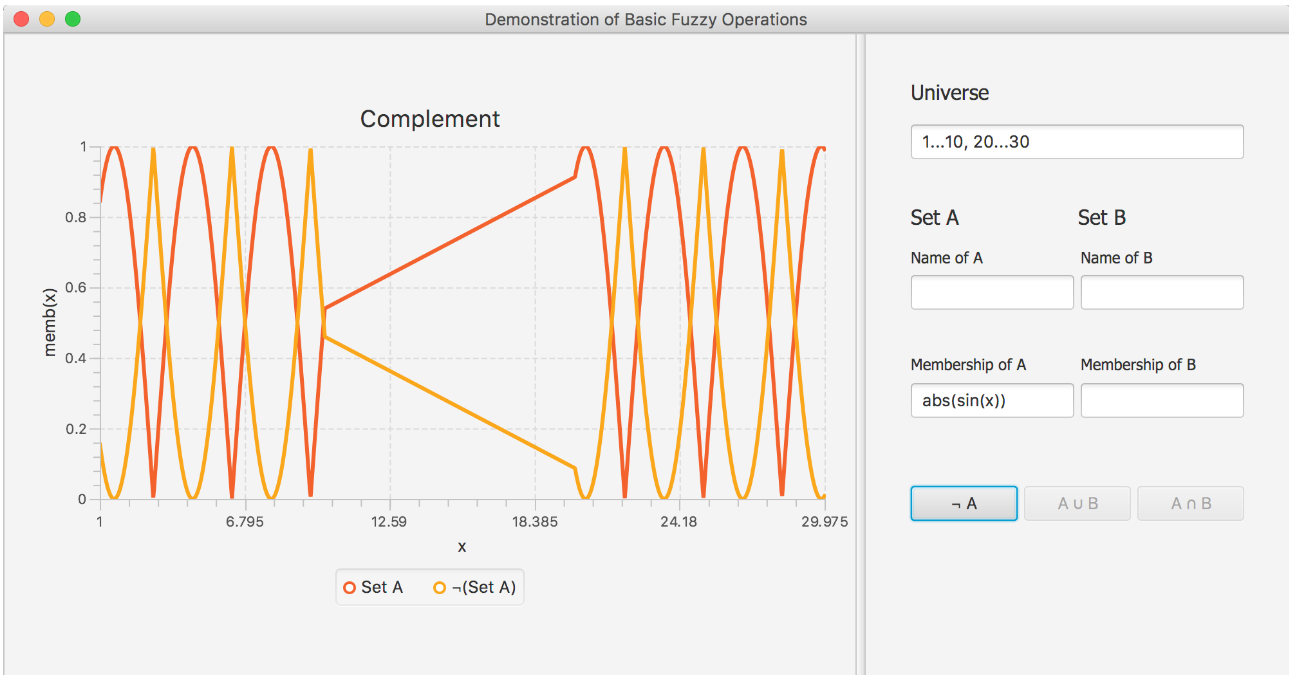
   (a) The `MainController` obtains the width of the line chart and sets the `precision` accordingly.

   (b) Then, it creates the universe which is a `Set` of `FuzzyMember`s generated by the `Range` class.

   (c) After the universe is created, the user-specified `FuzzySet` is instantiated.

   (d) In the following step, the complement of the instantiated `FuzzySet` is created.

   (e) Lastly, the `MainController` populates the line chart with the created fuzzy sets in the exact same order as they have been created. It assures that the data points of the complement fuzzy set will overwrite the data points of the user-specified fuzzy set in the chart.



Universe: 1...10, 20...30

Membership of A: abs(sin(x))

Figure 10: Line chart lacking discontinuity

4.3 Known issues

The application is able to present fuzzy sets defined by continuous universes. However, it cannot handle disconinuity. Assume a case when multiple ranges have been specified as the universe by the user, e.g. `1...10, 20...30`. The LineChart of the JavaFX library does not offer any tool to handle such situations. It connects the specified datapoints with a straight line. Having said that, such definitions of the universe may lead to results as seen in Figure 10. A possible solution would be to inject another datapoint before the start and after the end of each range with a membership degree of `Double.NaN`. This way the last point of a range and the first point of the following would be connected somewhere outside - in the undefined part of the line chart.

5 Conclusion

This report introduced a desktop application capable of operating with fuzzy sets defined by a continuous univers and a member function as a mathematical expression and demonstrate the 3 basic fuzzy operations over them. We presented the design of the application along with the implementation details and had a more in-depth discussion about challanges met during the development of the project regarding the representation of continuous data sets on a discrete system.

References

[1] T. Munakata. Fundamentals of the New Artificial Intelligence. Springer Science, 2008

[2] Andreas Haufler. scireum/parsii. https://github.com/scireum/parsii

[3] Andreas Haufler. How to write one of the fastest expression evaluators in Java? https://www.javacodegeeks.com/2014/01/how-to-write-one-of-the-fastest-expression-evaluators-in-java.html