Thesis Title: Specification and Formal Verification of Fuzzy Information Processing for the Case of Edge Detection.

Author: Kemal Keskin.

Department: Electrical and Computer Engineering

Approved for Thesis Requirements of the Master of Science Degree:

Thesis Advisor: Prof. Mieczyslaw M. Kokar

Thesis Reader: Prof. Stefano Basagni

Thesis Reader: Prof. Bahram Shafai

Chairman of Department:

Graduate School Notified of Acceptance:

Director of the Graduate School: Yaman Yener
SPECIFICATION AND FORMAL VERIFICATION OF
FUZZY INFORMATION PROCESSING FOR THE
CASE OF EDGE DETECTION

A Thesis Presented
by
Kemal Keskin
to
the Graduate School of Engineering
in Partial Fulfillment of the Requirements
for the Degree of
Master of Science
in Electrical Engineering
in the field of
Computer Engineering
Northeastern University
Boston, Massachusetts
August 2009
Abstract

This thesis explores how an information fusion system can be verified before it is built. Towards this aim, a system is first specified mathematically in a formal language. The specification then can be analyzed by formulating various conjectures (theorems) and proving (or disproving) them using an automatic theorem prover. In our work we use Metaslang as the formal specification language and Specware, which is a tool that supports category theory based algebraic specification of software. One of the most important aspects in information fusion is the uncertainty of the fused decisions that are based on uncertain sources. Out of many possible uncertainty types we focus on fuzzy set theory. We have specified a library of fuzzy set theory that can be used to build specifications of fuzzy information processing systems. As an example, a (fuzzy) edge detection algorithm was implemented and then verified using two theorem provers - Snark and Isabelle. With the help of this approach, reasoning about the impact of the uncertainty of input information is specified formally in every step of the fusion process and can be verified before the system is coded in a programming language.
Acknowledgements

I would like to thank to Professor Mieczyslaw M. Kokar for his guidance throughout my research and his patience to my questions. I would also like to thank to Dr. Stephen Westfold for his support with Specware and Isabelle. Finally, I would like to mention Gülşah Çakıroğlu for her specifications that motivated me in my research.
Contents

Abstract iii

Acknowledgements iv

1 INTRODUCTION 1

1.1 Introduction ........................................... 1
1.2 Thesis Overview ...................................... 3

2 CATEGORY THEORY 4

2.1 Introduction ........................................... 4
2.2 Definition Of Category ................................. 4
2.3 The Category Of Signatures ......................... 5
   2.3.1 Signature ........................................ 5
   2.3.2 Signature Morphism .............................. 6
2.4 The Category of Specifications ................... 6
   2.4.1 Specification .................................... 7
   2.4.2 Specification Morphisms ....................... 7
2.5 Diagrams ............................................ 7

3 SPECWARE OVERVIEW 9

3.1 What is Specware? .................................... 9
3.2 MetaSlang ............................................. 10
   3.2.1 Specs ......................................... 11
3.2.2 Types .................................................. 11
3.2.3 Operations(ops) and Definitions(defs) .............. 12
3.2.4 Claims: Axioms, Conjectures, and Theorems .......... 14
3.2.5 Lambda-Forms ....................................... 14
3.3 Specware Shell ......................................... 15
3.4 Specware and SNARK .................................... 16
3.5 Specware and Isabelle ................................. 16

4 ISABELLE .................................................. 19
4.1 Introduction ............................................. 19
4.2 Apply-style Proofs ...................................... 20
  4.2.1 apply(auto) ........................................ 20
  4.2.2 apply(induct-tac x) .............................. 20
  4.2.3 apply(simp add: x1 x2) ......................... 20

5 SETUP OF SPECWARE, ISABELLE and XEMACS IN UBUNTU 22
5.1 XEmacs 21.4.21 ....................................... 22
5.2 Specware 4.2.5 ....................................... 23
5.3 Isabelle ............................................... 23

6 FUZZY SETS .................................................. 24
6.1 Introduction to Fuzzy Sets ............................. 24
6.2 Basic Concepts ......................................... 25
6.3 Fuzzy Set Operations .................................. 27
  6.3.1 Fuzzy Complement ................................ 27
  6.3.2 Fuzzy Union : t - CONORM ...................... 28
  6.3.3 Fuzzy Intersection : t - NORM .................. 29
  6.3.4 Properties of Fuzzy Sets ......................... 31
6.4 Fuzzy Arithmetic ..................................... 31
  6.4.1 Fuzzy Numbers .................................... 31
6.4.2 Arithmetic Operations On Intervals . . . . . . . . . . . 32
6.4.3 Arithmetic Operations On Fuzzy Numbers . . . . . . . 34
6.4.4 Lattice Of Fuzzy Numbers . . . . . . . . . . . . . . . . 38
6.5 Fuzzy Reasoning Operators . . . . . . . . . . . . . . . . . 40
6.6 Fuzzy Set Specification . . . . . . . . . . . . . . . . . . . . 46
6.7 Fuzzy Information Processing . . . . . . . . . . . . . . . . . 49
   6.7.1 Fuzzification . . . . . . . . . . . . . . . . . . . . . . . 50
   6.7.2 Fuzzy Reasoning . . . . . . . . . . . . . . . . . . . . . 52
   6.7.3 Defuzzification . . . . . . . . . . . . . . . . . . . . . 53

7 AN EXAMPLE: EDGE DETECTION ALGORITHM 55
   7.1 Edge Detection Algorithm . . . . . . . . . . . . . . . . . . 55
   7.2 Fuzzy Edge Detection Modeling . . . . . . . . . . . . . . . 58
      7.2.1 Fuzzy Set . . . . . . . . . . . . . . . . . . . . . . . . 59
      7.2.2 Fuzzy Number (FuzzyNumber.sw) . . . . . . . . . 60
      7.2.3 Fuzzy Arithmetic (FuzzyArithm.sw) . . . . . . . . 61
      7.2.4 Fuzzy Reasoning (FuzzyReasoning.sw) . . . . . . . 62
      7.2.5 Image (Image.sw) . . . . . . . . . . . . . . . . . . 63
      7.2.6 Fuzzification (Fuzzification.sw) . . . . . . . . . 63
      7.2.7 Triangular Fuzzification (TriFuzzify.sw) . . . . . . 64
      7.2.8 Defuzzification (defuzzification.sw) . . . . . . . . 66
      7.2.9 Fuzzy Edge (fuzzyEdge.sw) . . . . . . . . . . . . 68
      7.2.10 Proving in SNARK and Isabelle . . . . . . . . . . . 76

8 CONCLUSION 83
   8.1 Conclusion . . . . . . . . . . . . . . . . . . . . . . . . . . 83
   8.2 Future Works . . . . . . . . . . . . . . . . . . . . . . . . 86

Bibliography 88
List of Figures

3.1 Sample spec. ............................................. 11
3.2 Some type declarations. ............................. 12
3.3 Some op declarations. ............................... 12
3.4 Op declaration and definition (separate). ........ 13
3.5 Op declaration and definition (combined). ....... 13
3.6 Axiomatically op definiton. ....................... 13
3.7 A Theorem sample. ................................... 14
3.8 Usage of gen-obligations command. ............... 17
3.9 Sample Specware Specification. .................... 17
3.10 Isabelle obligation theory translated from sample Specware specification above. .................. 18

4.1 Screenshot of XEmacs. ............................... 19
4.2 Example of a sub-goal. .............................. 20
4.3 The sub-goal in figure 4.2 is proved by apply(auto). ...... 20
4.4 The sub-goal before induction is applied .......... 21
4.5 The sub-goal after induction is applied .......... 21

5.1 Command to install XEmacs. ....................... 23
5.2 Commands to install Isabelle [15]. ............... 23

6.1 Complement of fuzzy set $A$. ....................... 28
6.2 Union of fuzzy sets $A$ and $B$. ................... 29
6.3 Intersection of fuzzy sets $A$ and $B$. . . . . . . . . . . . . . . . . 30
6.4 Addition of fuzzy sets $A$ and $B$. . . . . . . . . . . . . . . . . . 36
6.5 Substraction of fuzzy sets $A$ and $B$. . . . . . . . . . . . . . . . 37
6.6 Multiplication of fuzzy sets $A$ and $B$. . . . . . . . . . . . . . . 37
6.7 Division of fuzzy sets $A$ and $B$. . . . . . . . . . . . . . . . . . 38
6.8 fuzzy-min of fuzzy sets $A$ and $B$. . . . . . . . . . . . . . . . . . 39
6.9 fuzzy-max of fuzzy sets $A$ and $B$. . . . . . . . . . . . . . . . . . 40
6.10 Diagram for fuzzy-set. . . . . . . . . . . . . . . . . . . . . . . . 47

7.1 Unit Interval specification . . . . . . . . . . . . . . . . . . . . . . 59
7.2 Type Fuzzy Set definition . . . . . . . . . . . . . . . . . . . . . . 60
7.3 Declaration of Fuzzy Set operations . . . . . . . . . . . . . . . . 60
7.4 Definition of Fuzzy Set operations . . . . . . . . . . . . . . . . . 60
7.5 Type Fuzzy Number definition . . . . . . . . . . . . . . . . . . . 61
7.6 Definition of normality . . . . . . . . . . . . . . . . . . . . . . . 61
7.7 Definition of convexity . . . . . . . . . . . . . . . . . . . . . . . 61
7.8 Op fuzzy_add declaration . . . . . . . . . . . . . . . . . . . . . . 62
7.9 Op fuzzy_add definition . . . . . . . . . . . . . . . . . . . . . . . 62
7.10 Op fuzzy_min definition . . . . . . . . . . . . . . . . . . . . . . 63
7.11 Declaration of fuzzy reasoning operators . . . . . . . . . . . . . 63
7.12 Definition of fuzzy reasoning operators . . . . . . . . . . . . . . 64
7.13 Type Image definition . . . . . . . . . . . . . . . . . . . . . . . 64
7.14 Type Nz definition . . . . . . . . . . . . . . . . . . . . . . . . . 64
7.15 Specification of Fuzzy Set . . . . . . . . . . . . . . . . . . . . . . 65
7.16 Declaration and definition of uni_min and uni_max . . . . . . 65
7.17 Triangular fuzzification op declarations . . . . . . . . . . . . . . 65
7.18 Definitions of tri_fuzzify and tri_fuzzify_2 . . . . . . . . . . 66
7.19 Op tri_fuzzify_image definition . . . . . . . . . . . . . . . . . 66
7.20 Op defuzzify_1 declaration and definition . . . . . . . . . . . . 67
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.21 Declaration and definition of operations used in defuzzify_1</td>
<td>67</td>
</tr>
<tr>
<td>7.22 Op defuzzify_2 declaration and definition</td>
<td>68</td>
</tr>
<tr>
<td>7.23 Op alpha_intvl declaration and definition</td>
<td>68</td>
</tr>
<tr>
<td>7.24 Laplacian declaration and definition as an operation</td>
<td>69</td>
</tr>
<tr>
<td>7.25 Ops fuzzy_zero_crossing_1 and fuzzy_zero_crossing_2 declarations and definitions</td>
<td>71</td>
</tr>
<tr>
<td>7.26 Specification of Fuzzy Variance</td>
<td>74</td>
</tr>
<tr>
<td>7.27 Op Fuzzy_edge_point declaration and definition</td>
<td>75</td>
</tr>
<tr>
<td>7.28 Theorem fuzzyEdge</td>
<td>76</td>
</tr>
<tr>
<td>7.29 SNARK with no options.</td>
<td>77</td>
</tr>
<tr>
<td>7.30 SNARK with options.</td>
<td>77</td>
</tr>
<tr>
<td>7.31 Proof of theorem Complement</td>
<td>78</td>
</tr>
<tr>
<td>7.32 Proof of theorem Commutativity</td>
<td>78</td>
</tr>
<tr>
<td>7.33 Proof of theorem Assosiativity</td>
<td>79</td>
</tr>
<tr>
<td>7.34 Proof of theorem Monotonicity</td>
<td>79</td>
</tr>
<tr>
<td>7.35 Proof of theorem Fuzzify</td>
<td>79</td>
</tr>
<tr>
<td>7.36 Proof of theorem Uni_min</td>
<td>80</td>
</tr>
<tr>
<td>7.37 Subgoal for Monotonicity.</td>
<td>80</td>
</tr>
<tr>
<td>7.38 Theorem Monotonicity proved in step 1 in Isabelle.</td>
<td>81</td>
</tr>
<tr>
<td>7.39 Theorem Monotonicity.</td>
<td>81</td>
</tr>
<tr>
<td>7.40 Theorem fuzzyEdge proving step 1 in Isabelle.</td>
<td>81</td>
</tr>
<tr>
<td>7.41 Theorem fuzzyEdge proving step 2 in Isabelle.</td>
<td>82</td>
</tr>
<tr>
<td>7.42 Other steps in Isabelle.</td>
<td>82</td>
</tr>
</tbody>
</table>
List of Tables

6.1 Logical Mapping Table ........................................... 46
6.2 Arithmetic Mapping Table ....................................... 49
Chapter 1

INTRODUCTION

1.1 Introduction

Among the various paradigmatic changes in science and mathematics in this century, one such change concerns the concept of uncertainty. In science, this change has been manifested by a gradual transition from the traditional view, which insists that uncertainty is undesirable in science and should be avoided by all possible means, to an alternative view, which is tolerant of uncertainty and insists that science cannot avoid it.

According to the traditional view, science should strive for certainty in all manifestations (precision, specificity, sharpness, consistency, etc.); hence, uncertainty (imprecision, nonspecificity, vagueness, inconsistency, etc.) is regarded as unscientific. According to the alternative (modern) view, uncertainty is considered essential to science; it is not only an unavoidable plague, but it has, in fact, a great utility [1].

In order to develop an information fusion system, it is required to deal with uncertainty in a formal way. Formal verification tools, such as theorem provers and software development systems, help to build specifications for an information fusion system and verify their correctness mathematically before they are built.
Specware is a formal specification tool and an automated software development system [12] developed by Kestrel Institute. This tool is used in this thesis. It is known that building correct and clear specifications is difficult. But Specware can help by exploiting Category Theory, which includes algebraic theories, and allowing for building specifications progressively from smaller to larger components.

SNARK, is an automated theorem-proving program developed in Common Lisp [16], and Isabelle is a generic proof assistant that allows mathematical formulas to be expressed in a formal language and provides support for proving those formulas in a logical calculus [15], Isabelle is also used in this thesis. It will be demonstrated that a specification in Specware can be translated to Isabelle using the Specware interface and verified using some proving capabilities of Isabelle.

In this thesis, fuzzy set theory is used to handle uncertainty. So fuzzy information fusion system specifications are built. Fuzzy set, fuzzy number, fuzzy arithmetic operations, and various fuzzification methods are specified and stored in a specification library. The library is then used to build an edge detection algorithm. Proof obligations are then extracted and proved using both Isabelle and SNARK.

Why Is Fuzzy Set Used?

From the beginning of modern science until the end of the nineteenth century, uncertainty was generally viewed as undesirable in science and the idea was to avoid it. This attitude gradually changed with the emergence of statistical mechanics at the beginning of the twentieth century. To deal with the unmanageable complexity of mechanical processes on the molecular level, statistical mechanics resorted to the use of statistical mechanics, probability theory has been successfully applied in many other areas of science. However, in spite of
its success, probability theory is not capable of capturing uncertainty in all its manifestations [2]. So these limitations are part of the reason why fuzzy sets are used in this thesis.

1.2 Thesis Overview

The thesis is organized as follows. In Chapter 2, necessary aspects of category theory are presented. In Chapter 3, the tool Specware and the Metaslang language are explained. In Chapter 4, theorem prover Isabelle is introduced. In Chapter 5, the setup of Specware environment is presented. In Chapter 6, fuzzy sets and fuzzy reasoning operators are explained. In Chapter 7, fuzzy system modeling and an example - fuzzy edge detection - are presented. Conclusions and future works are proposed in Chapter 8.
Chapter 2

CATEGORY THEORY

2.1 Introduction

Category theory is a relatively young branch of mathematics, stemming from algebraic topology, and designed to describe various structural concepts from different mathematical fields in a uniform way. Indeed, category theory provides a bag of concepts (and theorems about those concepts) that form an abstraction of many concrete concepts in diverse branches of mathematics, including computing science. Hence it will come as no surprise that the concepts of category theory form an abstraction of many concepts that play a role in algorithmics [6]. So category theory is gracefully concise and simple language for describing information fusion systems.

Followings include some definitions that describe Category Theory and taken from [5]

2.2 Definition Of Category

A category $C$ of

- a collection of $Obj$ called $C$-objects;
- a collection of $Arw$ called $C$-arrows;
• operations assigning to each C-arrows $f$ a C-object $\text{dom } f$ (the domain of $f$) and a C-object $\text{cod } f$ (the "codomain " of $f$). If $a = \text{dom } f$ and $b = \text{cod } f$ this is displayed as

$$f : a \rightarrow b \text{ or } a \overset{f}{\rightarrow} b$$

• an operation, ” $\circ$ ”, called composition, assigning to each pair $\langle g, f \rangle$ of C-arrows with $\text{dom } g = \text{cod } f$, a C-arrow $g \circ f : \text{dom } f \rightarrow \text{cod } g$, the composite of $f$ and $g$ such that the Associative Law holds : Given the configuration

$$a \rightarrow b \rightarrow c \rightarrow d$$

of C – objects and C – arrows, then

$$h \circ (g \circ f) = (h \circ g) \circ f$$

• an assignment to each C-object, b, a C-arrow, $\text{id}_{b:b} = b$, called the identity arrow on $b$, such that the Identity Law holds : For any C-arrows $f : a \rightarrow b$ and $g : b \rightarrow c$

$$\text{id}_{b} \circ f = f \text{ and } g \circ \text{id}_{b} = g$$

2.3 The Category Of Signatures

In algebraic specifications, the structure of a specification is defined in terms of an abstract collection of values, called sorts and operations over those sorts. This structure is called a signature. A signature describes the structure that an implementation must have to satisfy the associated specification; however, a signature does not specify the semantics of the specification. The semantics are added later via axioms.
2.3.1 Signature

A signature $\Sigma = \langle S, \Omega \rangle$, consists of a set $S$ of sorts and a set $\Omega$ of operation symbols, defined over $S$. Associated with each operation’s symbol is a sequence of sorts called its rank. For example, $f : s_1, s_2, ..., s_n \rightarrow s$ indicates that $f$ is the name of an n-ary function, taking arguments of sorts $s_1, s_2, ..., s_n$ and producing a result of sort $s$. A nullary operation symbol, $c : \rightarrow s$ is called a constant of sort $s$.

For signatures, the $C$-arrows are called signature morphisms. Signatures and their associated signature morphisms form the category, Sign.

2.3.2 Signature Morphism

Given two signatures $\Sigma = \langle S, \Omega \rangle$ and $\Sigma' = \langle S', \Omega' \rangle$ a signature morphism $\sigma : \Sigma \rightarrow \Sigma'$ is a pair of functions $\langle \sigma_s : S \rightarrow S', \sigma_\Omega : \Omega \rightarrow \Omega' \rangle$, mapping sorts to sorts and operations to operations such that the sort map is compatible with the ranks of the operations, i.e., for all operation symbols $f : s_1, s_2, ..., s_n \rightarrow s$ in $\Omega$ the operation symbol $\sigma_\Omega(f) : \sigma_s(s_1), \sigma_s(s_2), ..., \sigma_s(s_n) \rightarrow \sigma_s(s)$ is in $\Omega'$. The composition of two signature morphisms, obtained by composing the functions comprising the signature morphisms, is also a signature morphism.

The identity signature morphism on a signature maps each sort and each operation onto itself. Signatures and signature morphisms form a category, Sign, where the signatures are the $C$-objects and the signature morphisms are the $C$-arrows.

2.4 The Category of Specifications

To model the semantics, signatures are extended with axioms that define the intended semantics of the signature operations. A signature with associated axioms is called a specification.
2.4.1 Specification

A specification $SP$ is a pair $\langle \Sigma, \Omega \rangle$ consisting of a signature $\Sigma = \langle S, \Phi \rangle$ and a collection $\Phi$ of $\Sigma$-sentences (axioms).

A specification only defines the semantics required of a valid implementation. In fact, for most specifications, there are a number of implementations that satisfy the specification.

As signatures have signature morphisms, specifications also have specification morphisms.

2.4.2 Specification Morphisms

Specification morphisms are signature morphisms that ensure that the axioms in the source specification are theorems (are provable from the axioms) in the target specification. Showing that the axioms of the source specification are theorems in the target specification is a proof obligation that must be shown for each specification morphism. Specification and specification morphisms enable the creation and modification of specifications that correspond to valid signatures within the category $\text{Sign}$.

A specification morphism from a specification $SP = \langle \Sigma, \Phi \rangle$ to a specification $SP' = \langle \Sigma', \Phi' \rangle$ is a signature morphism $\sigma: \Sigma \to \Sigma'$ such that for every model $M \in \text{Mod}[SP]$, $M|_\sigma \in \text{Mod}[SP']$. The specification morphism is also denoted by the same symbol, $\sigma: \Sigma \to \Sigma'$.

2.5 Diagrams

A diagram in a category $C$ is a collection of vertices and directed edges, consistently labeled with objects and arrows of $C$, where ”consistently” means that if an edge in the diagram is labeled with an arrow and has domain $A$ and codomain $B$, then the endpoints of this edge must be labeled with $A$ and $B$. 
Diagrams are often used for stating and proving properties of categorical constructions. Such properties can often be expressed by saying that a particular diagram commutes.
Chapter 3

SPECWARE OVERVIEW

3.1 What is Specware?

It is a tool for building and manipulating a collection of related specifications. Specware can be considered [13] :

- *a design tool*, because it can represent and manipulate designs for complex systems, software or otherwise

- *a logic*, because it can describe concepts in a formal language with rules of deduction

- *a programming language*, because it can express programs and their properties

- *a database*, because it can store and manipulate collections of concepts, facts, and relationships

It can be used to develop domain theories, develop code from specifications and develop specifications from code [13].

Specware is designed with the idea that large and complex problems can be specified by combining small and simple specifications. The problem specifications may be further refined into a working system by the controlled stepwise introduction of implementation design decisions, in such a way that the refined specifications and ultimately the working code is a provably correct refinement
of the original problem specification [13]

Specware uses notions and procedures based on category theory and related mathematics to manipulate specifications [10]. The advantage of category theory as the foundation of Specware is that it enables the production of a well-defined stepwise refinement from an abstract specification to concrete implementation. Specification morphisms preserve the structure of one specification through the translation to another specification and preserve theorems across the specifications [11].

Specware produces logical inference using external theorem provers such as; SRI’s Snark and Isabelle. External provers are connected to Specware through logic morphisms, which relate logics to each other. While Snark is an automated theorem prover, Isabelle is more interactive theorem prover that user can move forward step by step.

For this project, Specware 4.2.5 version was used.

\section*{3.2 MetaSlang}

MetaSlang is the specification language of Specware. It is a version of high-order logic like PVS\footnote{PVS is a specification language integrated with support tools and a theorem prover. For more information: http://pvs.csl.sri.com/} and HOL\footnote{HOL is a programming environment in which theorems can be proved and proof tools implemented. For more information http://hol.sourceforge.net/}.

MetaSlang is a functional language which is valuable for proving properties regarding functions. It includes syntactic constituents for describing functional semantics within a specification as well as constructs for describing composition, refinement, code generation, and proof capabilities. Specification constituents include types, expressions, and axioms which can be used to describe domain-specific formalisms [14]. All about MetaSlang in this section is a basic explanation. The \textit{Specware Language Manual} contains a detailed
description of the MetaSlang grammar, including a BNF description.

3.2.1 Specs

A specification is a finite presentation of a theory in higher-order logic [11]. Specifications, or specs, provide the means to describe abstract concepts of the problem domain. A basic specification consists of a set of specification constituents. The first one is *types*, which indicate a set of values syntactically. The second element is *operations* (or *ops*, for short), which denote an element of the set denoted by its type [13]. The last is *axioms* and *definitions*, which define actions of types and operations. A spec can be expanded by importing other specs. This process can be useful for more complicated problems. Specs are also the objects used in morphisms which define the *part-of* or *is-a* relationships between two specs. Morphisms allow for refinement of specs and provide the utility to take simple abstract specifications and refine them to more concrete, complex specifications [14].

```
FuzzySet = spec
declaration
...
endspec
```

Figure 3.1: Sample spec.

3.2.2 Types

A type is a syntactic entity that denotes a set of values. Types are collections or sets of objects and expressions that characterize those objects. Specware has several built-in types such as Boolean, Integer, etc. These can be used in specifications directly. They are imported automatically for every spec by
Specware. Besides the built-in types, users can define new types or combine existing types to build more complex types. Some examples are shown in Figure 3.2.

```plaintext
| type Nz = {i:Nat | i ~ 0} |
| type Image = Integer * Integer -> Nat |
| type Fuzzy_set a = a -> Uni_intvl |
```

Figure 3.2: Some type declarations.

### 3.2.3 Operations(ops) and Definitions(defs)

An operation or op is a syntactic symbol accompanied by a type. An op is an element of the set that is denoted by its type. A specification has a number of built-in operations, such as logical connectives.

Ops can be polymorphic. While a monomorphic op denotes an element of the set denoted by the type of the op, a polymorphic op denotes a function that, given a set for each parameter type of the polymorphic type of the op, returns an element of the set obtained by applying to such parameter sets the function denoted by the type of the op [19].

```plaintext
% Monomorphic op
op delta : Nz
% Polymorphic op
op inf : Intvl -> Nat
op fequal : Fuzzy_number * Fuzzy_number -> Uni_intvl
```

Figure 3.3: Some op declarations.

The behavior and constraint of an op can be shown by definitions. If definition is used for an op it must match the signature of the op declaration.
Also there is a possibility to combine defs and ops using a construct. Examples show that we can use defs and ops in a combined way.

% Op declaration

\begin{verbatim}
op [a] fuzzy_complement : Fuzzy_set a -> Fuzzy_set a
\end{verbatim}

% Op definition

\begin{verbatim}
def [a] fuzzy_complement(f1) = fn d -> 1 - f1(d)
\end{verbatim}

\begin{figure}[h]
\centering
\begin{verbatim}
% Op declaration and definition
op min (x1: Nat, x2: Nat): Nat = if x1 <= x2 then x1 else x2
\end{verbatim}
\caption{Op declaration and definition (combined).}
\end{figure}

An op definition can be considered as a special notation for an axiom. However defining ops axiomatically may not be always a good idea since an axiom is automatically assumed to be true and has no obligations while defs have proof obligations associated with them. So it is encouraged to use defs as much as possible.

% Op declaration

\begin{verbatim}
op uni_min : Uni_intvl * Uni_intvl -> Uni_intvl
\end{verbatim}

% Axiomatically op definition

\begin{verbatim}
axiom uni_min_def is
fa(u1: Uni_intvl, u2: Uni_intvl)
uni_min(u1,u2) = (if u1 <= u2 then u1 else u2)
\end{verbatim}

\begin{figure}[h]
\centering
\begin{verbatim}
% Op declaration
op uni_min : Uni_intvl * Uni_intvl -> Uni_intvl
% Axiomatically op definition
axiom uni_min_def is
fa(u1: Uni_intvl, u2: Uni_intvl)
uni_min(u1,u2) = (if u1 <= u2 then u1 else u2)
\end{verbatim}
\caption{Axiomatically op definiton.}
\end{figure}
3.2.4 Claims: Axioms, Conjectures, and Theorems

There are three kinds of claims that are usable in Speceware: axioms, conjectures, and theorems. Axioms, conjectures and theorems are formulas in a specification that use types and ops. They are a term of type Boolean.

In detail, an axiom is a logical sentence that is asserted to hold for all the objects and functions discussed by the specification while a theorem is a provable consequence of all the axioms of the theory. Conjectures are generated automatically by Specware, but the user can also create conjectures as well [14].

\[
\begin{align*}
% \text{Alpha-cut Theorem} \\
\textbf{theorem} \alpha\text{-cut}_\text{closed} \textbf{ is} \\
\text{fa}(f \text{: Fuzzy\_number, } a \text{: Uni\_intvl, } S \text{: Set\_of\_Nat)} \\
(\alpha\text{-cut}(f, a) = S \iff (\text{fa}(d \text{: Nat})(d \text{ in? } S) \Rightarrow \\
(\text{ex}(d1 \text{: Nat, } d2 \text{: Nat})(d1 \leq d2) \&\& (d1 \leq d) \&\& (d \leq d2))))
\end{align*}
\]

Figure 3.7: A Theorem sample.

3.2.5 Lambda-Forms

Lambda calculus can be used to define functions that take inputs and produces outputs. A form used to describe a function it is called a lambda expression. In Specware, lambda forms are represented by the symbol “fn”; the value of a lambda-form is a partial or total function. For instance, the meaning of a given lambda-form of type $S \rightarrow T$ is the function $f$ mapping each inhabitant $x$ of $S$ to a value $y$ of type $T$, where $y$ is the return value of $x$ for the match of the lambda-form. If the match accepts each $x$ of type $S$, function $f$ is total; otherwise it is partial, and undefined for those values $x$ rejected [13]. A simple usage of lambda-forms can be seen below.

\texttt{Uni\_intvl =}
\texttt{sup(map (fn (x,y) \rightarrow min(f1 x, f2 y)) (fn (x,y) \rightarrow x + y = z))}

\section{3.3 Specware Shell}

Specware specifications are processed in Specware Shell. Only units in specifications can be processed with commands. However, Specware shell can be run without a text editor, it can be run on XEmacs environment. Working on XEmacs environment has some advantages like; when an unit is processed if an error occurs XEmacs shows where the error is. Also Xemacs has syntax highlighting option that helps the user to edit specifications easily.

Processing a unit causes the recursive processing of the units referenced in that unit’s term. For instance, if spec A extends spec B which in turn extends spec C, then when A is processed, B and C are also processed. There must be no circularities in the chain of unit dependencies [18].

When a unit is processed, processing results are saved to an internal cache by Specware. When the user tries to get Specware to process the same unit, Specware checks the cache which carries the unit’s identifier and avoids recomputations if there is not any change on the file.

The Specware Shell consist of some commands that are used for basic file operations such as cd and ls. Also there are some other commands that are special to Specware such as show and proc.
3.4 Specware and SNARK

SNARK (SRI’s New Automated Reasoning Kit) is an automated theorem-proving program developed in Common Lisp. Its principal inference rules are resolution and paramodulation. SNARK’s style of theorem proving is similar to Otter’s [16].

Specware comes packaged with the SNARK first-order theorem prover. In order to get SNARK to prove theorems, first the user should add proof terms in specifications and then these specifications should be processed by Specware.

Specware will report an error if the claim to be proved does not occur in the spec, or if not all claims following occur in the spec before the claim to be proved [18]. SNARK saves all information about execution process in a log file. However, log files can be more complicated for some users, they can be helpful to understanding why the proofs succeeded or failed.

3.5 Specware and Isabelle

Isabelle is a generic proof assistant. It allows mathematical formulas to be expressed in a formal language and provides tools for proving those formulas in a logical calculus. The main application is the formalization of mathematical proofs and in particular formal verification, which includes proving the correctness of computer hardware or software and proving properties of computer languages and protocols [15].

Specware interface allows the use of Isabelle to formulate and prove obligations. The interface is essentially just an emacs command that converts a Specware spec to an Isabelle theory, along with extensions in the Specware syntax to allow Isabelle proof scripts to be embedded in Specware specs, and to allow the user to specify translation of Specware ops and types to existing
Isabelle constants and types [17]. User can translate Specware declarations, definitions, axioms and theorems to the corresponding Isabelle versions by using gen-obligations command.

```
* gen-obligations FuzzyEdgeDetection#FUZZY_NUMBER
"/home/user/Specware-4-2-5/Codes/Isa/
FuzzyEdgeDetection_FUZZY_NUMBER.thy"
```

Figure 3.8: Usage of gen-obligations command.

```
type Fuzzy_number = Fuzzy_set Nat
type Set_of_Nat = FiniteSet Nat
axiom normality is
fa(f:Fuzzy_number)
height(f) = 1
theorem alpha_cut_closed is
fa(f:Fuzzy_number, a:Uni_intvl, S:Set_of_Nat)
(alpha_cut(f, a) = S <=> (fa(d:Nat)(d in? S) =>
(ex(d1:Nat, d2:Nat)(d1<=d2) && (d1<=d) && (d<=d2))))
```

Figure 3.9: Sample Specware Specification.

A Specware definition may translate into one of four different kinds of Isabelle definitions: defs, recdefs and the newer funs and functions. Simple recursion on coproduct constructors translates to fun, but more complicated recursion is usually translated to fun. Some recursion still translates to recdef because the fun and function support is new, but the user can force translation to function. Non-recursive functions are translated to defs, except in some cases they are translated to fun which allows more pattern matching [17].
types Fuzzy_number = "nat Fuzzy_set"
types Set_of_Nat = "nat Set_FiniteSet"

theorem normality_Obligation_subtype:
  \begin{mma}
  \text{Fun_PR\ between\ zero\ one\ p\ (f::nat \rightarrow \text{Uni_intvl});}
  \text{\forall (x::int).}
  x = \text{int (height f)}
  \text{\rightarrow between\ zero\ one\ p\ (nat x) \land x \geq 0}
  \text{height f \geq 0}\
  \text{by auto}
\end{mma}

axioms normality:
"\begin{mma}
  \text{Fun_PR\ between\ zero\ one\ p\ f}
  \text{\rightarrow height\ f = 1}\
\end{mma}"

theorem alpha_cut_closed:
"\begin{mma}
  \text{Fun_PR\ between\ zero\ one\ p\ (f::nat \rightarrow \text{Uni_intvl});\ between\ zero\ one\ p\ (a::nat);\ finite}
  \text{S\rightarrow (alpha\_cut\_closed\_f,\ a) =}
  \text{\forall (d::nat).\ d \in S}
  \text{\rightarrow (exists\ (d1::nat)\ (d2::nat).\ d1 \leq d2 \land d \leq d2))}
\end{mma}"

Figure 3.10: Isabelle obligation theory translated from sample Specware specification above.
Chapter 4

ISABELLE

4.1 Introduction

Isabelle and Specware can be run separately under their own Xemacs jobs. However, it is possible to run them under the same XEmacs. At the present time, Specware and Isabelle cannot be run together under Windows. This only can be done under Linux by using IsabelleSpecware command.

Figure 4.1: Screenshot of XEmacs.

Isabelle has two proof styles. The first is Apply-style Proofs and the other is Structured Isar Proofs.
CHAPTER 4. ISABELLE

4.2 Apply-style Proofs

4.2.1 apply(auto)

This command tells Isabelle to apply a proof strategy called auto to all sub-goals. Essentially, auto tries to simplify the subgoals [8]. Figure shows a sample proof that is proved with apply(auto).

```
proof (prove): step 0
goal {1 subgoal}:
  1. [| Fun_PR between_zero_one_p f1; 
      ALL x. x = int (f1 d) --> between_zero_one_p (nat x) & 0 <= x |]
    ==> 0 <= f1 d
```

Figure 4.2: Example of a sub-goal.

```
theorem 
  |- norm_Obligation_subtype: 
    [| Fun_PR between_zero_one_p ?f1.0; 
      ALL x. x = int (?f1.0 ?d) --> between_zero_one_p (nat x) & 0 <= x |]
    ==> 0 <= ?f1.0 ?d
```

Figure 4.3: The sub-goal in figure 4.2 is proved by apply(auto).

4.2.2 apply(induct-tac x)

This command tells Isabelle to apply a proof strategy called induct_tac to perform induction to the variable x. Figure 4.4 shows an example of sub-goals in Isabelle before induction is applied, while Figure 4.5 shows an example of induct_tac being applied to the variable a.

4.2.3 apply(simp add: x1 x2)

This command tells Isabelle to apply a simplification proof strategy by adding x1 and x2, which are rules. Also there is another modifier: del. Modifier add can be changed to modifier del to delete simplification rules.
proof (prove): step 0

goal (1 subgoal):
1. \[ \text{Fun}_{PR} \text{ between_zero_one_p f; between_zero_one_p a; finite S } \implies (\alpha \text{_cut } (f, a) = S) = \]
\[ (\forall d. d : S \implies (\exists d1 d2. d1 <= d2 \land d1 <= d \land d <= d2)) \]

Figure 4.4: The sub-goal before induction is applied

proof (prove): step 1

goal (2 subgoals):
1. \[ \text{Fun}_{PR} \text{ between_zero_one_p f; between_zero_one_p a; finite S } \implies (\alpha \text{_cut } (f, 0) = S) = \]
\[ (\forall d. d : S \implies (\exists d1 d2. d1 <= d2 \land d1 <= d \land d <= d2)) \]
2. \[ \text{Fun}_{PR} \text{ between_zero_one_p f; between_zero_one_p a; finite S; } \]
\[ (\alpha \text{_cut } (f, n) = S) = \]
\[ (\forall d. d : S \implies (\exists d1 d2. d1 <= d2 \land d1 <= d \land d <= d2)) \]

Figure 4.5: The sub-goal after induction is applied
Chapter 5

SETUP OF SPECWARE, ISABELLE and XEMACS IN UBUNTU

To use Specware and Isabelle under same XEmacs, we need to work under a Linux operating system. In this work the choice was UBUNTU 9.04. Specware and Isabelle development environment requires the following software installed.

- Specware 4.2.5
- Isabelle 2008
- XEmacs 21.4.21

5.1 XEmacs 21.4.21

XEmacs can be installed by using the apt-get command in Ubuntu. It is installed automatically. But there must be an internet connection since XEmacs must be downloaded before the installing process. The figure below shows the command line.
CHAPTER 5. SETUP OF SPECWARE, ISABELLE AND XEMACS IN UBUNTU

keskemal@ubuntu:~$ sudo apt-get install xemacs21

Figure 5.1: Command to install XEmacs.

5.2 Specware 4.2.5

Specware installation is similar to install any software in Windows. Just execute setuplinux.bin and follow the instructions.

5.3 Isabelle

The following files are required for the Isabelle 2008 installation:

- Isabelle2008.tar.gz
- ProofGeneral.tar.gz
- Polyml_x86-linux.tar.gz
- HOL_x86-linux.tar.gz

Installation of Isabelle/HOL on Linux (x86 and x86_64) works by downloading and unpacking the relevant packages. In the subsequent example the installation location is /usr/local, but any other directory works as well [15]:

```bash
$ tar -C /usr/local -xzf Isabelle2009.tar.gz
$ tar -C /usr/local -xzf ProofGeneral.tar.gz
$ tar -C /usr/local -xzf polyml_x86-linux.tar.gz
$ tar -C /usr/local -xzf HOL_x86-linux.tar.gz
```

Figure 5.2: Commands to install Isabelle [15].
Chapter 6

FUZZY SETS

6.1 Introduction to Fuzzy Sets

In classical sets the transition for an element between membership and non-membership in the universe of a given set is abrupt. While the transition for an element that is in the universe of fuzzy sets can be gradual. This transition takes different degrees of membership that shows the boundaries of fuzzy sets are vague and ambiguous. Hence, the membership of an element is defined by describing the vagueness and ambiguity using the membership function. In general, membership functions take values in the unit interval $[0,1]$.

We can define fuzzy sets in multiple ways. In the following we show two examples of fuzzy set definition.

Definition 1: Fuzzy set $A$ is a function

$$A : X \rightarrow [0, 1]$$

where $X$ is the universe of discourse.

Definition 2: When the universe of discourse, $X$, is discrete and finite, fuzzy set $A$ is:

$$A = \left\{ \frac{\mu_A(x_1)}{x_1} + \frac{\mu_A(x_2)}{x_2} + \ldots \right\} = \left\{ \sum_i \frac{\mu_A(x_i)}{x_i} \right\}$$
When the universe, \( X \), is continuous and infinite the fuzzy set \( A \) is:

\[
A = \left\{ \int \frac{\mu_A(x)}{x} \right\}
\]

The "+" sign in the first notation and the integral sign in the second notation represent the function-theoretic union.

### 6.2 Basic Concepts

One of the most important concepts of fuzzy sets is the concept of an \( \alpha \) - cut. Definition of \( \alpha \) - cut is

\[
\alpha A = \{ x | \mu_A(x) \geq \alpha \}
\]

where \( 0 \leq \alpha \leq 1 \). In the definition, \( \mu_A(x) \) denotes the membership function and the set \( \alpha A \) is a crisp set which derived from fuzzy set \( A \). Any element \( x \) from \( \alpha A \) (i.e. \( x \in \alpha A \)) that is greater than or equal to the value \( \alpha \) has a grade of membership in fuzzy set \( A \).

One other concept is strong \( \alpha \) - cut. Definition of strong \( \alpha \) - cut is

\[
\alpha^+ A = \{ x | \mu_A(x) > \alpha \}
\]

**Examples for \( \alpha \) - cut and strong \( \alpha \) - cut.**

Let us consider an universe \( X = \{a, b, c, d\} \) and a fuzzy set \( A \) on this universe.

\[
A = \left\{ \frac{1}{a} + \frac{0.5}{b} + \frac{0.2}{c} + \frac{0.8}{d} \right\}
\]

Let us define \( \alpha \) - cut values: \( \alpha = 0, 0.4, 0.5, 0.8 \).

\[
\begin{align*}
0A &= \{a, b, c, d\} \\
0.4A &= \{a, b, d\} \\
0.5A &= \{a, b, d\} \\
0.8A &= \{a, d\}
\end{align*}
\]

\[
\begin{align*}
0^+A &= \{a, b, c, d\} \\
0.4^+A &= \{a, b, d\} \\
0.5^+A &= \{a, d\} \\
0.8^+A &= \{a\}
\end{align*}
\]
The next concept support is the $\alpha$ - cut set of $^\alpha A$. Symbolically,

$$supp(A) = ^{0+}A = \{x \in X|A(x) > 0\}$$

where $\alpha = 0$. As seen from the definition, it contains all elements in $A$, except zero. Clearly, the support of $A$ is exactly the same as the strong $\alpha$ - cut of $A$ for $\alpha = 0$.

Another important concept is core. Core of $A$ contains all elements for which the degree of membership is 1 in $A$.

$$core(A) = ^1 A = \{x \in X|A(x) = 1\}$$

The largest value of $\alpha$ for which the $\alpha$ - cut is not empty is called the height of fuzzy set $A$.

$$h(A) = sup_{x \in X} A(x)$$

If the height of fuzzy set $A$ equals to one (i.e. $h(A) = 1$), fuzzy set $A$ is called normal, otherwise it is called subnormal.

If for any elements $x, y, z$ in fuzzy set $A$, the relation $z < y < x$ implies that

$$\mu_A(y) \geq min[\mu_A(z), \mu_A(x)]$$

then $A$ is a convex fuzzy set for $x, y, z \in \mathbb{R}$.

Lastly, three basic operations in crisp sets can be generalized as standard fuzzy set operations. These are complement, intersection and union.

The most natural way to express standard complement is

$$\bar{A}(x) = 1 - A(x)$$

for all $x \in X$.

The standard intersection, denoted $A \cap B$, can be defined by this equation
\[(A \cap B)(x) = \min [A(x), B(x)]\]

Also the standard union, denoted \(A \cup B\), can be defined in the same way,
\[(A \cup B)(x) = \max [A(x), B(x)]\]

For the last two equation, min and max denote the minimum and maximum operator.

### 6.3 Fuzzy Set Operations

In this section standard fuzzy operations (fuzzy complement, intersection and union) are introduced.

#### 6.3.1 Fuzzy Complement

Let \(A\) is a fuzzy set defined on a universal set \(X\), its complement \(\bar{A}\) is another fuzzy set on \(X\) such that for each \(x \in X\), \(A(x)\) denotes the degree to which \(x\) belongs to \(A\) while \(\bar{A}(x)\) denotes the degree to which \(x\) does not belong to \(A\).

\[\bar{A}(x) = 1 - A(x)\]

for all \(x \in X\).

Fuzzy sets overlap their complements as seen from figure 6.1. When a member belongs to the fuzzy set \(A\) with the degree of 0.5, then it also belongs to the fuzzy set \(\bar{A}(x)\) with a degree of 0.5. This is the main difference between fuzzy set theory and classical set theory since a set never overlaps its complement in classical set theory.

There are two characterization theorems of fuzzy complements.

**Theorem 1:** Let \(c\) be a function from \([0,1]\) to \([0,1]\). Then \(c\) is a fuzzy complement (involutive) iff there exists a continuous function \(g\) from \([0,1]\) to \(\mathbb{R}\) such that \(g(0) = 0\), \(g\) is strictly increasing, and
CHAPTER 6. FUZZY SETS

28

Figure 6.1: Complement of fuzzy set $A$.

$$c(a) = g^{-1}(g(1) - g(a))$$

for all $a \in [0,1]$.]

Theorem 2: Let $c$ be a function from $[0,1]$ to $[0,1]$. Then $c$ is a fuzzy complement iff there exists a continuous function $f$ from $[0,1]$ to $\mathbb{R}$ such that $f(1) = 0$, $g$ is strictly decreasing, and

$$c(a) = f^{-1}(f(0) - f(a))$$

for all $a \in [0,1]$.

6.3.2 Fuzzy Union : t - CONORM

Let $A$ and $B$ are fuzzy sets defined on a universal set $X$. The standard fuzzy union of $A$ and $B$, denoted by $A \cup B$, is defined by using their membership functions.

$$(A \cup B)(x) = max[A(x), B(x)]$$

for all $x \in X$.

In classical set theory, the equation

$$A \cup \bar{A} = X$$
is always true. However it does not hold for fuzzy sets under standard fuzzy union. It can be seen easily that this equation is invalid for all elements \( x \in X \) such that \( A(x) \notin 0, 1 \). For example, \( A(x) = 0.8 \), then \( \bar{A} = 1 - 0.8 = 0.2 \). Hence,

\[
(A \cup B)(x) = \max[0.8, 0.2] = 0.8
\]

If this was a classical set, \( x \) was expected a member of \( X \) with full membership \( (x = 1) \), but it is obvious that it does not hold for fuzzy sets under standard fuzzy union.

![Figure 6.2: Union of fuzzy sets A and B.](image)

There is one characterization theorem of \( t \)-conorms.

**Theorem:** Let \( u \) be a binary operation on the unit interval. Then, \( u \) is an Archimedean \( t \)-conorm if and only if there exists an increasing generator \( g \) such that

\[
u(a, b) = g^{-1}(g(a) + g(b))
\]

for all \( a, b \in [0, 1] \). [1]

### 6.3.3 Fuzzy Intersection: \( t \)-NORM

Like fuzzy union, let \( A \) and \( B \) are fuzzy sets defined on a universal set \( X \). The standard fuzzy intersection of \( A \) and \( B \), denoted by \( A \cap B \), is defined by using their membership functions.
\[(A \cap B)(x) = \min[A(x), B(x)]\]

for all \(x \in X\).

In classical set theory, the equation

\[A \cap \bar{A} = \emptyset\]

is always true. However, it does not hold for fuzzy sets under standard fuzzy intersection. It can be seen easily that this equation is invalid for all elements \(x \in X\) such that \(A(x) \notin 0, 1\). For example, \(A(x) = 0.8\), then \(\bar{A} = 1 - 0.8 = 0.2\). Hence,

\[(A \cap B)(x) = \min[0.8, 0.2] = 0.2\]

If this was a classical set, \(x\) was expected a member of \(X\) with the degree of \(0\) \((x = 0)\), but it is obvious that it does not hold for fuzzy sets under standard fuzzy intersection.

![Figure 6.3: Intersection of fuzzy sets A and B.](image)

There is one characterization theorem of t-norms.

**Theorem:** Let \(i\) be a binary operation on the unit interval. Then, \(i\) is an *Archimedean* t-norm if and only if there exists a decreasing generator \(f\) such that
\[ i(a, b) = f^{(-1)}(f(a) + f(b)) \]

for all \( a, b \in [0, 1]. \) \[1\]

### 6.3.4 Properties of Fuzzy Sets

Fuzzy sets follow the same properties as classical sets. The following is a list of frequently used properties of fuzzy sets.

<table>
<thead>
<tr>
<th>Property</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Commutativity</strong></td>
<td>( A \cup B = B \cup A )</td>
</tr>
<tr>
<td><strong>Associativity</strong></td>
<td>( A \cup (B \cup C) = (A \cup B) \cup C )</td>
</tr>
<tr>
<td></td>
<td>( A \cap (B \cap C) = (A \cap B) \cap C )</td>
</tr>
<tr>
<td><strong>Distributivity</strong></td>
<td>( A \cup (B \cap C) = (A \cup B) \cap (A \cup C) )</td>
</tr>
<tr>
<td></td>
<td>( A \cap (B \cup C) = (A \cap B) \cup (A \cap C) )</td>
</tr>
<tr>
<td><strong>Idempotency</strong></td>
<td>( A \cup A = A \quad \text{and} \quad A \cap A = A )</td>
</tr>
<tr>
<td><strong>Identity</strong></td>
<td>( A \cup \emptyset = A \quad \text{and} \quad A \cap X = A )</td>
</tr>
<tr>
<td></td>
<td>( A \cap \emptyset = \emptyset \quad \text{and} \quad A \cup X = X )</td>
</tr>
<tr>
<td><strong>Transitivity</strong></td>
<td>If ( A \subseteq B \subseteq C ), then ( A \subseteq C )</td>
</tr>
<tr>
<td><strong>Involution</strong></td>
<td>( \overline{A} = A )</td>
</tr>
</tbody>
</table>

### 6.4 Fuzzy Arithmetic

#### 6.4.1 Fuzzy Numbers

A fuzzy number is described in terms of a number and a linguistic modifier such as *approximately, around, or nearly*. For example *around seven* is fuzzy, since it contains some numbers that are on either side of the central value *seven*. The membership function decreases from 1 to 0 both sides of central value. This kind of fuzzy sets are called *fuzzy numbers*. Membership functions can be shown in the form:
A : \mathbb{R} \rightarrow [0, 1]

Not all membership functions in this form are fuzzy numbers. In order to qualify as a fuzzy number, a fuzzy set A on \mathbb{R} must provide at least the following three properties:

1. Fuzzy set A must be normal. The only condition for a fuzzy set to be normal is its height must equal to 1.

\[ h(A) = \sup_{x \in X} A(x) = 1 \]

2. Alpha-cuts of fuzzy set A must be a closed interval for every \( \alpha \in (0, 1] \).

Since \( \alpha \)-cuts of fuzzy number are required to be closed intervals, every fuzzy number is a convex fuzzy set. If for any elements \( x, y, z \) in fuzzy set A, the relation \( z < y < x \) implies that

\[ \mu_A(y) \geq \min[\mu_A(z), \mu_A(x)] \]

then A is a convex fuzzy set for \( x, y, z \in \mathbb{R} \).

3. The support of fuzzy set A, \( 0^+A \), must be bounded.

These properties are necessary for defining meaningful arithmetic operations on fuzzy numbers.

6.4.2 Arithmetic Operations On Intervals

Fuzzy set and fuzzy number can be described by the intervals associated with \( \alpha \)-cuts. As a result of that, we can define arithmetic operations on fuzzy numbers in terms of arithmetic operations on closed intervals.

Let’s consider wo interval numbers \( I_1 \) and \( I_2 \)...
\[ I_1 = [a, b] \quad \text{where} \quad a \leq b \]
\[ I_2 = [c, d] \quad \text{where} \quad c \leq d \]

Denote a general arithmetic operation for these two interval numbers:
\[ I_1 \ast I_2 = [a, b] \ast [c, d] \]

We can represent the basic four arithmetic interval operations using this scheme:
\[
[a, b] + [c, d] = [a + c, b + d] \\
[a, b] - [c, d] = [a - d, b - c] \\
[a, b] \cdot [c, d] = [\min(ac, ad, bc, bd), \max(ac, ad, bc, bd)] \\
[a, b] \div [c, d] = [a, b] \cdot \frac{1}{\frac{1}{d}, \frac{1}{c}} \quad 0 \notin [c, d]
\]

\[ \alpha[a, b] = \begin{cases} 
(\alpha a, \alpha b) & \text{for } \alpha > 0 \\
(\alpha a, \alpha b) & \text{for } \alpha < 0 
\end{cases} \]

where \( ac, ad, bc, \) and \( bd \) are arithmetic products and \( 1/d \) and \( 1/c \) are quotients.

Arithmetic operations on closed intervals satisfy some useful properties. To overview them, let \( A = [a_1, a_2], B = [b_1, b_2], C = [c_1, c_2], 0 = [0, 0], 1 = [1, 1] \). Using these symbols, the properties are formulated as follows [1]:

1. \( A + B = B + A \) 
   \( A \cdot B = B \cdot A \) (commutativity)
2. \( (A + B) + C = A + (B + C) \) 
   \( (A \cdot B) \cdot C = A \cdot (B \cdot C) \) (associativity)
3. \( A = 0 + A = A + 0 \) 
   \( A = 1 \cdot A = A \cdot 1 \) (identity)
4. \( A \cdot (B + C) \subseteq A \cdot B + A \cdot C \) (subdistributivity)

5. If \( b \cdot c \geq 0 \) for every \( b \in B \) and \( c \in C \), then \( A \cdot (B + C) = A \cdot B + A \cdot C \) (distributivity). Furthermore, if \( A = [a, a] \), then \( a \cdot (B + C) = a \cdot B + a \cdot C \).

6. \( 0 \in A - A \) and \( 1 \in A/A \).

7. If \( A \subseteq E \) and \( B \subseteq F \), then:

\[
\begin{align*}
A + B & \subseteq E + F \\
A - B & \subseteq E - F \\
A \cdot B & \subseteq E \cdot F \\
A/B & \subseteq E/F \text{ (inclusion monotonicity)}
\end{align*}
\]

6.4.3 Arithmetic Operations On Fuzzy Numbers

In order to use fuzzy numbers in practice, it is required to define usual operations on fuzzy numbers, such as addition, subtraction, multiplication, and division. From last section, it is known that fuzzy numbers can be described by the intervals associated with \( \alpha \)-cuts and these are all closed intervals of real numbers. Also it is known how to apply four basic operations to closed intervals. Hence, arithmetic operations on fuzzy numbers can be defined.

Let \( A \) and \( B \) denote two fuzzy numbers and \( \ast \) denote any of the four basic arithmetic operations. Then, \( A \ast B \), by defining its \( \alpha \)-cut, \( ^\alpha (A \ast B) \), as

\[
^\alpha (A \ast B) = ^\alpha A \ast ^\alpha B
\]

for any \( \alpha \in (0, 1] \). It can also be expressed as

\[
A \ast B = \bigcup_{\alpha \in [0,1]} (^\alpha A \ast ^\alpha B) \times \alpha
\]

so the result will be fuzzy number.[1]

As an example of employing equations above, let’s consider two triangular-shape fuzzy numbers \( A \) and \( B \) defined as follows:
\[ A(x) = \begin{cases} 
0, & \text{for } x \leq -1 \text{ and } x > 3 \\
\frac{x}{2}, & \text{for } -1 \leq x < 1 \\
\frac{(4-x)}{2}, & \text{for } 1 < x \leq 3
\end{cases} \]

\[ B(x) = \begin{cases} 
0, & \text{for } x \leq 1 \text{ and } x > 5 \\
\frac{(x+2)}{2}, & \text{for } 1 < x \leq 3 \\
\frac{(2-x)}{2}, & \text{for } 3 < x \leq 5.
\end{cases} \]

From these formulas, \( \alpha \)-cuts:

\[ ^\alpha A = [^{\alpha} a_1, ^\alpha a_2] \]

\[ ^\alpha B = [^{\alpha} b_1, ^\alpha b_2] \]

Then,

\[ A(^\alpha a_1) = (^{\alpha} a_1 + 1)/2 = \alpha \]

\[ A(^\alpha a_2) = (3 - ^\alpha a_2)/2 = \alpha \]

From these equations,

\[ ^\alpha a_1 = 2\alpha - 1 \]

\[ ^\alpha a_2 = 3 - 2\alpha \]

Hence,

\[ ^\alpha A = [2\alpha - 1, 3 - 2\alpha] \]

When the same process is applied to \( B \) we obtain

\[ ^\alpha B = [2\alpha + 1, 5 - 2\alpha] \]
Using these intervals,

\[
(A + B)(x) = \begin{cases} 
0, & \text{for } x \leq 0 \text{ and } x > 8 \\
\frac{x}{4}, & \text{for } 0 \leq x < 4 \\
\frac{(8-x)}{4}, & \text{for } 4 < x \leq 8 
\end{cases}
\]

\[
(A - B)(x) = \begin{cases} 
0, & \text{for } x \leq -6 \text{ and } x > 2 \\
\frac{(x+6)}{4}, & \text{for } -6 \leq x < -2 \\
\frac{(2-x)}{4}, & \text{for } -2 < x \leq 2 
\end{cases}
\]

\[
(A \cdot B)(x) = \begin{cases} 
0, & \text{for } x < -5 \text{ and } x \geq 15 \\
\frac{[3-(4-x)^{1/2}]}{2}, & \text{for } -5 \leq x < 0 \\
\frac{(1+x)^{1/2}}{2}, & \text{for } 0 \leq x < 3 \\
\frac{[4-(1+x)^{1/2}]}{2}, & \text{for } 3 \leq x < 15 
\end{cases}
\]

\[
(A/B)(x) = \begin{cases} 
0, & \text{for } x \leq -1 \text{ and } x \geq 3 \\
\frac{(x+1)}{(2-2x)}, & \text{for } -1 \leq x < 0 \\
\frac{(5x+1)}{(2x+2)}, & \text{for } 0 \leq x < 1/3 \\
\frac{(3-x)}{(2x+2)}, & \text{for } 1 < x \leq 3 
\end{cases}
\]

The extension principle can be used to define usual operations on fuzzy num-
bers. Let $A$ and $B$ denote two fuzzy numbers and $\star$ denote any of the four basic arithmetic operations. Then, $A \star B$ is defined by the equation,

$$(A \star B)(z) = \sup_{z=x+y} \min[A(x), B(y)]$$

for all $z \in \mathbb{R}$. More specifically, the four arithmetic equations are defined as:

$$(A + B)(z) = \sup_{z=x+y} \min[A(x), B(y)]$$
Figure 6.7: Division of fuzzy sets $A$ and $B$.

\[(A - B)(z) = \sup_{z=x-y} \min[A(x), B(y)]\]

\[(A \cdot B)(z) = \sup_{z=x \cdot y} \min[A(x), B(y)]\]

\[(A/B)(z) = \sup_{z=x/y} \min[A(x), B(y)]\]

### 6.4.4 Lattice Of Fuzzy Numbers

The set $\mathbb{R}$ of real numbers is linearly ordered. For every pair of real numbers, $x$ and $y$, either $x \leq y$ or $y \leq x$. The pair $(\mathbb{R}, \leq)$ is a lattice, which is also expressed in terms of two lattice operations,

\[
\min(x, y) = \begin{cases} 
x & \text{if } x \leq y \\
y & \text{if } y \leq x
\end{cases}
\]

\[
\max(x, y) = \begin{cases} 
y & \text{if } x \leq y \\
x & \text{if } y \leq x
\end{cases}
\]
for every pair \( x, y \in \mathbb{R} \). The linear ordering of real numbers does not extend to fuzzy numbers, but fuzzy numbers can be ordered partially in a natural way and that this partial ordering forms a distributive lattice. So the lattice operations on fuzzy numbers can be shown as: for any two fuzzy numbers \( A \) and \( B \),

\[
MIN(A, B)(z) = \sup_{z = \min(x, y)} \min[A(x), B(y)]
\]

\[
MAX(A, B)(z) = \sup_{z = \max(x, y)} \min[A(x), B(y)]
\]

for all \( z \in \mathbb{R} \). The operations MIN and MAX are totally different from the standard fuzzy intersection and union, \( \min \) and \( \max \). This difference can be seen from the following equations:

\[
A(x) = \begin{cases} 
0, & \text{for } x < -2 \text{ and } x > 4 \\
\frac{(x+2)}{3}, & \text{for } -2 \leq x \leq 1 \\
\frac{(4-x)}{3}, & \text{for } 1 \leq x \leq 4 
\end{cases}
\]

\[
B(x) = \begin{cases} 
0, & \text{for } x < 1 \text{ and } x > 3 \\
x - 1, & \text{for } 1 \leq x < 2 \\
3 - x, & \text{for } 2 \leq x \leq 3 
\end{cases}
\]

Figure 6.8: fuzzy-min of fuzzy sets \( A \) and \( B \).
6.5 Fuzzy Reasoning Operators

In this section basic fuzzy reasoning operators are extended from the crisp logic. Fuzzy logic connectives such as fuzzy implication, fuzzy biimplication, fuzzy and, fuzzy or and fuzzy negation are specified. Fuzzy equal, fuzzy less than and equal, as well as fuzzy greater than and equal are three operators that have been specified.

Depending on partial ordering of fuzzy numbers, sometimes two fuzzy
numbers are comparable. In many fuzzy decision problems, the output alternatives are often represented in terms of fuzzy numbers. The requirement might be to express a crisp preference of alternatives, so it might need a method for construction a crisp total ordering of fuzzy numbers.

Numerous methods for total ordering of fuzzy numbers have been investigated and suggested, but the issue of choosing a proper ordering method in a certain circumstance is an open research. Here two methods suggested in [2] are listed.

The first method uses Hamming Distance. For any given fuzzy numbers $A$ and $B$, the Hamming Distance, $d(A, B)$ is defined by the formula:

$$d(A, B) = \int_R |A(x) - B(x)| \, dx$$

For any fuzzy numbers $A$ and $B$, first fuzzy-min$(A, B)$ is calculated, which is the least upper bound in the lattice. Then the Hamming distances $d(\text{fuzzy-min}(A, B), A)$ and $d(\text{fuzzy-min}(A, B), B)$ are calculated.

The compatibility of partial ordering of comparable fuzzy numbers with the ordering defined by the Hamming distance can be proven, because if $\tilde{A} \leq \tilde{B}$, then fuzzy-min$(A, B)$=$B$ and hence, $A \leq B$.

Another method is based on $\alpha$-cuts. Given fuzzy numbers $A$ and $B$, select a particular value of $\alpha \in [0, 1]$ and determine the $\alpha$-cuts $^\alpha A = [a_1, a_2]$ and $^\alpha B = [b_1, b_2]$. Then the total ordering of two fuzzy numbers can be defined as:

$$A \leq B \text{ iff } a_2 \leq b_2$$

This definition is dependent upon the choice of a value $\alpha$. It is usually required that $\alpha > 0.5$.

However, in order to do fuzzy reasoning, the whole process has to be done in a fuzzified way, so that the fuzziness, or the level of uncertainty, can be maintained and could be reasoned about. In other words, fuzzy logic has to
be used. *Fuzzy equal, fuzzy less than and equal*, as well as *fuzzy greater than and equal* are three fuzzy logic operators are to be used in the approach. And *fuzzy implication, fuzzy biimplication, fuzzy and, fuzzy or* and *fuzzy negation* are fuzzy logical connectives that should be defined.

Each fuzzy logical operator or connective is associated with a mapping $[0, 1]^2 \rightarrow [0, 1]$. So the truth value of a proposition is admitted as the range of truth values. And the truth value of a combined composition can be determined from the truth values of the atomic propositions.

**Definition :** Let $T$ be a t-norm. Fuzzy implication $\Rightarrow$ is defined as

$$\Rightarrow: [0, 1] \times [0, 1] \rightarrow [0, 1],$$

$$\beta \Rightarrow \gamma = \sup\{\alpha \in [0, 1] | T(\alpha, \beta) \leq \gamma\}$$

**Definition :** Let $T$ be continuous t-norm.

$$\Rightarrow: [0, 1] \times [0, 1] \rightarrow [0, 1],$$

and

$$\alpha \Rightarrow \beta = T(\alpha \Rightarrow \beta, \beta \Rightarrow \alpha)$$

are called the fuzzy implications and fuzzy biimplication, respectively, induced by $T$.

This definition is motivated by the equivalence

$$\alpha \Leftrightarrow \beta \equiv (\alpha \Rightarrow \beta) \land (\beta \Rightarrow \alpha)$$

The following theorem can make the calculation of fuzzy biimplication easier.

**Theorem :** Let $T$ be a t-norm, $\alpha, \beta \in [0, 1]$. Then

$$\alpha \Leftrightarrow \beta = \max(\alpha, \beta) \Rightarrow \min(\alpha, \beta)$$

holds.
Depending on which t-norm is used, fuzzy implication and fuzzy biimplication could be in different versions.

Assume t-norm $T = T_{Luka}$, then

$$\alpha \Rightarrow \beta = \sup\{\gamma \in [0, 1] | \max(\alpha + \gamma - 1, 0) \leq \beta\}$$

$$= \sup\{\gamma \in [0, 1] | \alpha + \gamma - 1, 0 \leq \beta\}$$

$$= \sup\{\gamma \in [0, 1] | \gamma \leq 1 - \alpha + \beta\}$$

$$= \min(1 - \alpha + \beta, 1)$$

which is called Lukasiewicz implication. The corresponding biimplication is:

$$\alpha \Leftrightarrow \beta = \min(1 - \max(\alpha, \beta) + \min(\alpha, \beta), 1)$$

$$= 1 - |\alpha - \beta|$$

Assume t-norm $T = T_{min}$, then

$$\alpha \Rightarrow \beta = \sup\{\gamma \in [0, 1] | \min(\alpha, \gamma) \leq \beta\}$$

$$= \begin{cases} 1, & \text{if } \alpha \leq \beta \\ \beta, & \text{otherwise} \end{cases}$$

which is called Gödel implication. The corresponding biimplication is:

$$\alpha \Leftrightarrow \beta = \max(\alpha, \beta) \Rightarrow \min(\alpha, \beta)$$

$$= \begin{cases} 1, & \text{if } \alpha = \beta \\ \min(\alpha, \beta), & \text{otherwise} \end{cases}$$

Assume t-norm $T = T_{prod}$, then
\[ \alpha \sim \beta = \sup\{ \gamma \in [0,1] | \alpha \cdot \gamma \leq \beta \} \]
\[ = \begin{cases} \frac{\beta}{\alpha}, & \text{if } \beta < \alpha \\ 1, & \text{otherwise} \end{cases} \]

which is called Goguen implication. The corresponding biimplication is:

\[ \alpha \sim \beta = \max(\alpha, \beta) \sim \min(\alpha, \beta) \]
\[ = \begin{cases} 1, & \text{if } \alpha = \beta \\ \frac{\min(\alpha, \beta)}{\max(\alpha, \beta)}, & \text{otherwise} \end{cases} \]

The basic concept of the crisp logic (classical logic or two-valued logic) is that every proposition is either True or False. In fuzzy logic, the truth value could be extended to a value between 0 and 1, that is \([0,1]\), instead of \(\{0,1\}\). This value represents the uncertainty level of the proposition. Let \(\Upsilon(\phi) \in [0,1]\) denotes the truth value of the proposition \(\phi\). The truth values of statements that contain quantifiers are normally determined as:

\[ \Upsilon(\forall x : P(x)) = \inf \{ \Upsilon(P(x)) | x \} \]

\[ \Upsilon(\forall x : P(x)) = \sup \{ \Upsilon(P(x)) | x \} \]

With the previous support, now we can consider the truth value of fuzzy equal, fuzzy less than and equal, as well as fuzzy greater than and equal.

For two fuzzy sets, or fuzzy numbers in particular, \(\mu\) and \(\nu\), the truth value of the value of the statement \(\mu \sim \nu\), where \(\sim\) represents fuzzy equal, can be defined as:

\[ \Upsilon(\mu \sim \nu) = \Upsilon(\forall x : (x \in \mu \sim x \in \nu)) \]
\[ = \Upsilon(\forall x : ((x \in \mu \sim x \in \nu) \land (x \in \nu \sim x \in \mu))) \]
So the truth value of $\mu \tilde{\geq} \nu$ depends on the truth functions chosen for the fuzzy implication $\tilde{\Rightarrow}$ and the fuzzy conjunction $\tilde{\wedge}$. Using Lukasiewicz implication yield:

$$\Upsilon(\mu \tilde{\geq} \nu) = \inf\{1 - |\mu(x) - \nu(x)| | x \in \text{Real}\}$$

whereas if Gödel implication is used, it will yield:

$$\Upsilon(\mu \tilde{\geq} \nu) = \inf\{\text{göd}(\mu(x), \nu(x)) | x \in \text{Real}\}$$

where

$$\text{göd}(\alpha, \beta) = \begin{cases} 1, & \text{if } \alpha = \beta \\ \min(\alpha, \beta), & \text{otherwise} \end{cases}$$

For two fuzzy numbers $\mu$ and $\nu$, the truth value of the statement ‘$\mu \tilde{\geq} \nu$’, where $\tilde{\geq}$ represents fuzzy greater than and equal, can be defined as:

$$\Upsilon(\mu \tilde{\geq} \nu) = \Upsilon(\forall x : (x \in \nu \tilde{\Rightarrow} x \in \mu))$$

So the truth value of $\mu \tilde{\geq} \nu$ depends on the truth functions chosen for the fuzzy implication $\tilde{\Rightarrow}$. Using Lukasiewicz implication yield:

$$\Upsilon(\mu \tilde{\geq} \nu) = \inf\{\min(1 - \nu(x) + \mu(x), 1) | x \in \text{Real}\}$$

For two fuzzy numbers $\mu$ and $\nu$, the truth value of the statement ‘$\mu \tilde{\leq} \nu$’, where $\tilde{\leq}$ represents fuzzy greater than and equal, can be defined as:

$$\Upsilon(\mu \tilde{\leq} \nu) = \Upsilon(\forall x : (x \in \mu \tilde{\Rightarrow} x \in \nu))$$

So the truth value of $\mu \tilde{\leq} \nu$ depends on the truth functions chosen for the fuzzy implication $\tilde{\Rightarrow}$. Using Lukasiewicz implication yield:

$$\Upsilon(\mu \tilde{\leq} \nu) = \inf\{\min(1 - \mu(x) + \nu(x), 1) | x \in \text{Real}\}$$

Logical Mapping Table for Reasoning Operators gives the mapping of crisp reasoning operators to fuzzy equivalent.

Where the truth value symbol $\Upsilon$ is eliminated. A complete mapping table of crisp to fuzzy is given at the end as Appendix A.
6.6 Fuzzy Set Specification

In this work, Specware is chosen as a tool for the handling of the fuzzy uncertainty. And since Specware requires that all functions be total, the first definition of fuzzy set (i.e., a fuzzy set is represented as a function) in Chapter 1 is chosen for building specifications. The diagram of the specification of fuzzy set is shown in figure 6.4.

<table>
<thead>
<tr>
<th>Logical Mapping Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRISP</td>
</tr>
<tr>
<td>Number</td>
</tr>
<tr>
<td>True, False</td>
</tr>
</tbody>
</table>

\[ \Rightarrow: [0,1] \times [0,1] \rightarrow [0,1] \]

1. \( a \Rightarrow b = (1, 1 - a + b) \)
2. \( a \Rightarrow b = \begin{cases} 1, & \text{if } a \leq b \\ b, & \text{other} \end{cases} \)

\[ \Leftrightarrow: [0,1] \times [0,1] \rightarrow [0,1] \]

1. \( a \Leftrightarrow b = 1 - |a - b| \)
2. \( a \Leftrightarrow b = \begin{cases} 1, & \text{if } a = b \\ \min(a, b), & \text{other} \end{cases} \)

\[ =: FN \times FN \rightarrow [0,1] \]

1. \( \mu = v = \inf \{1 - |\mu(x) - v(x)||x \in \mathbb{R}\} \)
2. \( \mu = v = \inf \{g(\mu(x), v(x))||x \in \mathbb{R}\} \)

where \( g(a, b) = \begin{cases} 1, & \text{if } a = b \\ b, & \text{other} \end{cases} \)

\[ \geq: FN \times FN \rightarrow [0,1] \]

\( \mu \geq v = \inf \{\min(1 - v(x) + \mu(x), 1)|x \in \mathbb{R}\} \)

\[ \leq: FN \times FN \rightarrow [0,1] \]

\( \mu \leq v = \inf \{\min(1 - \mu(x) + v(x), 1)|x \in \mathbb{R}\} \)

Table 6.1: Logical Mapping Table
The spec UNI-INTVL imports REAL and introduces a new sort : Uni-intvl, which characterizes unit interval [0, 1].

```
UNI_INTVL = spec
import Real
op between_zero_one?(x:Real): Boolean = x < one && zero <= x
type Uni_intvl = (Real | between_zero_one?)
```

FUZZY-SET is a definitional extension of the colimit of UNI-INTVL and SET. It defines a function sort :

```
FUZZY_SET = spec
import /Library/General/Set, UNI_INTVL
type Fuzzy_set a = a -> Uni_intvl
```
where a is the type of elements in Set. In the FUZZY-SET spec, the basic fuzzy concepts such as fuzzy complement, union, and intersection are also specified which can also be found as Appendix B.

Fuzzy number is imported from the spec FUZZY_SET. In FUZZY_NUMBER spec new type Set_of_Real is defined for using real numbers set.

```
FUZZY_NUMBER = spec
import FUZZY_SET
type Fuzzy_number = Fuzzy_set Real
type Set_of_Real = Set Real
```

In chapter 6.4, the arithmetic operations on intervals and fuzzy numbers were discussed. In the next sections an example of fuzzy edge detection will be given. This example deals with fuzzy numbers, so the arithmetic operations on fuzzy numbers have to be specified. This is done in FUZZY_ARITHM spec. Fuzzy arithmetic operators are specified in this spec, which is a definitional extension of FUZZY_NUMBER, with fuzzy operations being of the following types:

```
op fuzzy_add : Fuzzy_number * Fuzzy_number -> Fuzzy_number
op fuzzy_sub : Fuzzy_number * Fuzzy_number -> Fuzzy_number
op fuzzy_mult : Fuzzy_number * Fuzzy_number -> Fuzzy_number
op fuzzy_div : Fuzzy_number * Fuzzy_number -> Fuzzy_number
op fuzzy_min : Fuzzy_number * Fuzzy_number -> Fuzzy_number
op fuzzy_max : Fuzzy_number * Fuzzy_number -> Fuzzy_number
```

The definition of each operation is given in the spec FUZZY_ARITHM, which is included in the Appendix B.

Let $A$ and $B$ denote two fuzzy numbers and $*$ denote any of the four basic arithmetic operations. Then, $A * B$ is defined by the equation,
(A \ast B)(z) = \sup_{z=x+y} \min[A(x), B(y)]

for all \( z \in \mathbb{R} \). More specifically, the four arithmetic operations are defined as,

\[
(A + B)(z) = \sup_{z=x+y} \min[A(x), B(y)]
\]

\[
(A - B)(z) = \sup_{z=x-y} \min[A(x), B(y)]
\]

\[
(A \cdot B)(z) = \sup_{z=x-y} \min[A(x), B(y)]
\]

\[
(A/B)(z) = \sup_{z=x/y} \min[A(x), B(y)]
\]

<table>
<thead>
<tr>
<th>Arithmetic Mapping Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRISP</td>
</tr>
<tr>
<td>Number</td>
</tr>
<tr>
<td>+</td>
</tr>
<tr>
<td>&amp;</td>
</tr>
<tr>
<td>-</td>
</tr>
<tr>
<td>&amp;</td>
</tr>
<tr>
<td>\cdot</td>
</tr>
<tr>
<td>&amp;</td>
</tr>
<tr>
<td>\div</td>
</tr>
<tr>
<td>&amp;</td>
</tr>
</tbody>
</table>

Table 6.2: Arithmetic Mapping Table

Arithmetic Mapping Table gives the mapping of crisp operators to fuzzy equivalent where \( \ast \) represents a fuzzy operator.

### 6.7 Fuzzy Information Processing

Fuzzy information processing consists of three stages which are: fuzzification, fuzzy reasoning, and defuzzification. They are explained in the following
6.7.1 Fuzzification

There are two kinds of approaches for fuzzification. The first approach is to fuzzify the input measurements by introducing one kind of fuzzification functions to each input variable to express the associated measurement uncertainty. The purpose of the fuzzification function is to interpret measurements of input variables, each expressed by a real number, for instance the intensity value of each pixel, as more realistic fuzzy approximation of the respective real numbers. This fuzzification function, which is also called membership function can be of the types triangular, trapezoidal, gaussian and so on.

Another way for fuzzification is to generate several fuzzy property sets out of the original image. For example, Dark, Normal, Bright fuzzy sets can be generated from an image, with each pixel corresponding to a value between 0 and 1, representing the likelihood the pixel belongs to each set.

Here the first approach is chosen since the fuzzy numbers can be used in this approach. For a given value $c$, the triangular fuzzy number $A$ is defined, such that for all $x \in \text{Real}$, $A(x)$ satisfies the equation:

$$A(x) = \begin{cases} 
0, & \text{if } x < c - \delta \\
\delta, & \text{if } c - \delta \leq x \leq c \\
\delta, & \text{if } c \leq x \leq c + \delta \\
0, & \text{if } x > c + \delta 
\end{cases}$$

In this equation, $\delta$ represents the uncertainty level. The larger the $\delta$, the more uncertain the input data.

And also for a given value $c$, the trapezoidal fuzzy number $B$ is defined, such that for all $x \in \text{Real}$, $B(x)$ satisfies the equation:
In this equation, $\delta_{1,2}$ represents the uncertainty level. The larger $\delta_{1,2}$ is, the more uncertain the input data is.

For a given value $c$, the Gaussian fuzzy number $C$ is defined, such that for all $x \in \text{Real}$, $C(x)$ satisfies the equation:

$$C(x) = \exp\left(-\left(\frac{x - \mu}{\sigma}\right)^2\right)$$

One kind of typical input data for an information fusion system is image, which is generally sampled into a rectangular array of pixels. Each pixel has an x-y coordinate that corresponds to its location within the image, and an intensity value representing brightness of a pixel.

The spec IMAGE imports INTEGER and Real, and defines a type.

```
IMAGE = spec
import Real
type Image = Integer * Integer -> Real
endspec
```

The spec FUZZIFICATION is generated by taking the colimit of IMAGE and FUZZY_REASONING, which is going to be explained in the next subsection.
FUZZIFICATION = spec
import IMAGE, FUZZY_REASONING
type Nz = {i:Real | i ~= zero}
endspec

Using the FUZZIFICATION spec, three different versions of fuzzification are generated: TRI_FUZZIFY, TRA_FUZZIFY and GAUSS_FUZZIFY, representing triangular, trapezoidal and gaussian fuzzification methods. The detailed specs are included at the end in Appendix B.

In each spec, two operations are defined. For instance in TRI_FUZZIFY, the following two operators are defined:

\[
\begin{align*}
\text{op tri\_fuzzify} : & \text{Real } \times \text{Nz } \rightarrow \text{Fuzzy\_number} \\
\text{op tri\_fuzzify\_2} : & \text{Real } \rightarrow \text{Fuzzy\_number}
\end{align*}
\]

where operation \text{tri\_fuzzify} takes a crisp number and some uncertainty level, and generates a fuzzy triangular number. The operation \text{tri\_fuzzify\_2} deals with the situation when the uncertainty level is zero, which means there is no fuzziness about the result. The latter operation is specified so that a crisp number can also be regarded as a fuzzy number.

### 6.7.2 Fuzzy Reasoning

In chapter 6.5, basic fuzzy reasoning operators were extended from the crisp logic. In order to do fuzzy reasoning, the whole reasoning process has to be done in fuzzy logic so that the fuzziness, or the level of uncertainty, can be maintained and could be reasoned. In another word, fuzzy logic have to be used. \text{Fuzzy equal}, \text{fuzzy less than and equal}, as well as \text{fuzzy greater than and equal} are three fuzzy logic operators going to be used in the approach. And \text{fuzzy implication}, \text{fuzzy biimplication}, \text{fuzzy and}, \text{fuzzy or} and \text{fuzzy negation} are fuzzy logical connectives that should be defined.
Each fuzzy logical operator or connective is associated with a mapping \([0, 1]^2 \to [0, 1]\). So the truth value of a proposition is admitted as the range of truth values. And the truth value of a combined composition can be determined from the truth values of the atomic propositions. These were all explained in details in chapter 6.5.

Fuzzy reasoning operators are specified in the spec FUZZY_REASONING, which is a definition extension of the spec FUZZY_ARITHM, with the fuzzy operations being of the following type:

\[
\begin{align*}
\text{op fequal} & : \text{Fuzzy number} \times \text{Fuzzy number} \to \text{Uni_intvl} \\
\text{op fgeq} & : \text{Fuzzy number} \times \text{Fuzzy number} \to \text{Uni_intvl} \\
\text{op fleq} & : \text{Fuzzy number} \times \text{Fuzzy number} \to \text{Uni_intvl}
\end{align*}
\]

In the spec FUZZY_REASONING only Lukasiewicz implication was specified. There are also other applicable versions which were explained in details, but can be specified as a future work. The crisp to fuzzy logic table for these reasoning operators were shown in Table 6.1.

### 6.7.3 Defuzzification

This is the process of calculating single-output numerical value for a fuzzy output variable on the basis of the inferred resulting membership function for this variable. The input in the defuzzification process is a fuzzy number and the output is a crisp number.

There are several defuzzification methods proposed in the literature: centroid calculation, which returns the center of the area under the curve of the fuzzy number; center of maximum, which returns the average of the maximum value of the fuzzy number; largest of maximum, and smallest of maximum. As
an example for the specifications the center of maximum method is chosen to implement the defuzzification process.

In this method, the defuzzified value, \( \text{defuzzify}(F) \), is defined as the average of the smallest value and the largest value, for which \( \text{height}(F) \) is 1:

Defuzzification is implemented in the spec \text{DEFUZZIFICATION}, which is a definitional extension of the spec \text{FUZZY\_NUMBER}. The defuzzify operation is defined as:

\[
\text{defuzzify}(F) = \frac{\text{inf}(M) + \text{sup}(M)}{2}
\]

where \( M \) = an interval \([z]\) s.t. \( F(z) = \text{height}(F) = 1 \)

Another version of this method is also specified by using \( \alpha \)-cut:

\[
\text{defuzzify}(F) = \frac{\text{inf}(M) + \text{sup}(M)}{2}
\]

where \( M \) = an interval \([z]\) s.t. \( F(z) = \alpha F \)

All specifications about this spec is also included in the Appendix B.
Chapter 7

AN EXAMPLE: EDGE DETECTION ALGORITHM

7.1 Edge Detection Algorithm

An edge in an image could be considered as a boundary at which a significant change of intensity, $I$, occurs.

Detecting an edge is very useful in object identification, because edges represent shapes of objects. There are many algorithms for edge detection. The objective of an edge detection algorithm is to locate the regions where the intensity is changing rapidly. So the whole process can be decomposed into two steps, the first is to derive edge points in an image, the second is to apply edge detection method only to these points.

For instance, if the gradient of this signal is taken (which, in one dimension, is just the first derivative with respect to $t$) the following occurs:
Clearly, the gradient has a large peak centered around the edge. By comparing the gradient to a threshold, an edge can be detected whenever the threshold is exceeded (as shown above). In this case, the edge is found, but the edge has become "thick" due to the thresholding. However, since it is known that the edge occurs at the peak, it can be localized by computing the Laplacian (in one dimension, the second derivative with respect to $t$) and finding the zero crossings.

The above figure shows the Laplacian of one-dimensional signal. As expected, the edge corresponds to a zero crossing, but other zero crossings are also seen which correspond to small ripples in the original signal.

Here, the Laplacian-based method is used to derive edge points. Edge points are where the second-order derivatives of the points are zero \cite{7} . So edge points can be searched by looking for zero-crossing points of $\nabla^2 I(x, y)$, which can be calculated by the equation
\[ \nabla^2 I(x, y) = I(x + 1, y) + I(x - 1, y) + I(x, y + 1) + I(x, y - 1) - 4I(x, y) \]

In order to avoid false edge points, local variance is estimated and compared with a threshold. The local variance can be estimated by

\[ \sigma^2(x, y) = \frac{1}{(2M + 1)^2} \sum_{k_1 = x-M}^{x+M} \sum_{k_2 = y-M}^{y+M} [(I(k_1, k_2) - m(k_1, k_2))^2] \]

where

\[ m(x, y) = \frac{1}{(2M + 1)^2} \sum_{k_1 = x-M}^{x+M} \sum_{k_2 = y-M}^{y+M} I(k_1, k_2) \]

with \( M \) typically chosen around 2. Since \( \sigma^2(x, y) \) is compared with a threshold, the scaling factor \( \frac{1}{(2M + 1)^2} \) can be eliminated.

Zero-crossing detection means comparing a pixel (say \( x, y \)) with the adjacent ones on the next row and next column. Declaring zero-crossing points as edges results in a large number of points being declared to be edge points.

If there is an 'zero-crossing' action, that particular pixel is accepted as one of the edge components, where

if \( \text{pixel}(x, y) < 0 \) and \( \text{pixel}(x, y + 1) > 0 \) or \( \text{pixel}(x + 1, y) > 0 \),
then \( \text{pixel}(x, y) = \) zero-crossing point

else if \( \text{pixel}(x, y) > 0 \) and \( \text{pixel}(x, y + 1) < 0 \) or \( \text{pixel}(x + 1, y) < 0 \),
then \( \text{pixel}(x, y) = \) zero-crossing point

A pixel at \( (x, y) \) satisfies an edge point if and only if the second-order derivatives of the points are zero-crossing points and the local variance is greater than or equal to the threshold. The illustration of this definition is shown in figure below:
axiom edge_point_def is

fa(I: Image, x: Integer, y: Integer)

(edge_point?(I, x, y) <=>
(zero_crossing_1(I,x,y) || zero_crossing_2(I,x,y)) &&
(thrd < var(I, x, y)))

7.2 Fuzzy Edge Detection Modeling

After setting up the Specware and Isabelle environment, the following Specware specifications were built that are needed to model Fuzzy Edge Detection.

- Fuzzy Set
- Fuzzy Number
- Fuzzy Arithmetic
- Fuzzy Reasoning
- Image
- Fuzzification
- Triangular Fuzzification
- Defuzzification
- Edge Point
- Edge and Fuzzy Edge

SNARK and Isabelle theorem provers were used to prove proof obligations. SNARK, proves theorems by using automatic tactics, however it could prove only simple theorems. However some options were changed. So the Isabelle
theorem prover was chosen to prove proof-obligations for all. Although most of proof-obligations could be proved by Isabelle, one of the theorems about FuzzyEdgeDetection still could not be proved.

Since Specware has not provided Real numbers library, it is needed to define Real numbers as Natural numbers in specifications.

\[
\begin{align*}
\text{Real Number} & \quad [0, 1] \\
\text{Natural Numbers} & \quad [0, 100]
\end{align*}
\]

### 7.2.1 Fuzzy Set

Fuzzy set constitutes the base of a fuzzy system. Unit interval is needed to define fuzzy set. So UNI\_INTVL spec is as follows:

\[
\begin{align*}
\text{UNI\_INTVL} &= \text{spec} \\
\text{op} \between\text{zero}\_\text{one}\?(x:\text{Nat}): \text{Boolean} &= \\
x < 100 \&\& 0 \leq x \\
\text{type} \ Uni\_intvl &= (\text{Nat} \mid \between\text{zero}\_\text{one}\?) \\
\text{endspec}
\end{align*}
\]

Figure 7.1: Unit Interval specification

Here, operation \( \between\text{zero}\_\text{one}\? \) is a boolean function; it returns true if number \( x \) is between 0 and 100. Otherwise it returns false. Using this operation a new type \( Uni\_intvl \) was built. It is a set that includes natural numbers only between 0 and 100.

Hence, fuzzy set is declared as a type by importing FiniteSet and UNI\_INTVL. It consists of a finite set and creates a new set in \( Uni\_intvl \) as shown in figure 7.2. Here \( a \) denotes a finite set.

Fuzzy set operations - fuzzy complement, intersection, union, alpha-cut, and height - are declared as operations as shown in figure 7.3.

Fuzzy set operations are defined in figure 7.4. They are binary operations on the unit interval.
7.2.2 Fuzzy Number (FuzzyNumber.sw)

A fuzzy number is declared as a type by importing FUZZY_SET as shown in figure 7.5. A fuzzy number is a function, thus the type of fuzzy numbers is a type of functions.

When the height of fuzzy number is equal to 1 it is normal. Since unit interval was defined as a set of numbers between 0 and 100, 100 is used instead of 1 to denote the full membership.

Also convexity can be defined as in figure 7.7.
7.2.3 Fuzzy Arithmetic (*FuzzyArithm.sw*)

In this specification, arithmetic operations (addition, subtraction, multiplication, division) on fuzzy numbers are declared and defined. Fuzzy number specification is required to define arithmetic operations, so first FUZZY_NUMBER spec is imported. Fuzzy addition is declared as an operation that consists of two fuzzy numbers and it yields a fuzzy number. Fuzzy addition declaration is shown in figure 7.8. Other operations are declared in a similar way.

Definitions of these operations are specified following the second method suggested by Klir. Here the fuzzy numbers are represented by continuous membership functions.

\[
\star \text{ in } z = x \star y \text{ equation denotes any of the four basic arithmetic operations.}
\]

\[
\text{axiom convexity is} \quad \begin{align*}
\text{fa}(f:Fuzzy\_number, \ x1:\text{Nat}, \ x2:\text{Nat}, \ lamd:\ \text{Uni}\_\text{intvl}) \\
0.5 * f(lamd * x1) + (0.5 - lamd) * x2) \geq \min(f(x1), f(x2))
\end{align*}
\]

Figure 7.7: Definition of convexity
% FUZZY NUMBER spec is imported.

\textbf{import} FUZZY\_NUMBER

% Fuzzy addition declaration

\textbf{op} fuzzy\_add : Fuzzy\_number * Fuzzy\_number \rightarrow Fuzzy\_number

% Substraction, Multiplication, and Division are declared by same way.

\begin{figure}
\centering
\includegraphics[width=\textwidth]{fig7_8.png}
\caption{Op fuzzy\_add declaration}
\end{figure}

\begin{figure}
\centering
\includegraphics[width=\textwidth]{fig7_9.png}
\caption{Op fuzzy\_add definition}
\end{figure}

So changing all $z = x + y$ to $z = x - y$, $z = x \times y$, and $x = z \times y$ in specification above yields other operations on fuzzy numbers. These specifications are available in the appendix.

Lattice operations on fuzzy numbers can be specified as figure 7.10.

\subsection{7.2.4 Fuzzy Reasoning (FuzzyReasoning\_sw)}

Fuzzy equal, fuzzy less than and equal, as well as fuzzy greater than and equal are three operators that have been specified in this part. They are declared as operations such that each operation consists of two fuzzy number and yields a number in unit interval.

Definitions for these three operations are shown in figure 7.12.
**7.2.5 Image (Image.sw)**

A type named `image` is declared such that it consists of pixels, each including two integers plus a natural number. The two integer numbers such as `x`, and `y` denote pixel of an image. The intensity is represented by a natural number.

Type declaration for `image` can be seen in figure 7.13.

**7.2.6 Fuzzification (Fuzzification.sw)**

In this specification, IMAGE and FUZZY_REASONING are imported and a type named `Nz` is declared which is a set of natural numbers except zero. This spec is written to be used in the fuzzification method (triangular, trapezoidal, gaussian) specifications. The Fuzzification spec is shown in Figure 7.14.
Axiom \texttt{fequal} is
\[
\texttt{fa}(f1:\text{Fuzzy\_number}, f2:\text{Fuzzy\_number}, x:\text{Nat}) \\
\texttt{fequal}(f1, f2) = \inf (100 - (\text{abs}(f1(x) - f2(x))))
\]

Axiom \texttt{fgeq} is
\[
\texttt{fa}(f1:\text{Fuzzy\_number}, f2:\text{Fuzzy\_number}, x:\text{Nat}) \\
\texttt{fgeq}(f1, f2) = \inf (\min ((100 - f1(x) + f2(x)), 100))
\]

Axiom \texttt{fleq} is
\[
\texttt{fa}(f1:\text{Fuzzy\_number}, f2:\text{Fuzzy\_number}, x:\text{Nat}) \\
\texttt{fleq}(f1, f2) = \inf (\min ((100 - f2(x) + f1(x)), 100))
\]

Figure 7.12: Definition of fuzzy reasoning operators

\begin{verbatim}
IMAGE = spec
  type Image = Integer * Integer -> Nat
endspec
\end{verbatim}

Figure 7.13: Type Image definition

7.2.7 Triangular Fuzzification (\texttt{TriFuzzify\_sw})

Triangular fuzzification is specified by importing FUZZIFICATION. A type named \texttt{Tri\_fuzzy\_image} is declared as shown in figure 7.15. This type denotes triangular fuzzified images such that two integers denote a pixel with a fuzzy number representing the fuzzy intensity of the pixel.

Minimum and maximum as lattice operations on unit interval are defined next.

\begin{verbatim}
FUZZIFICATION = spec
  import IMAGE, FUZZY\_REASONING
  type Nz = \{i:Nat | i\sim=0\}
endspec
\end{verbatim}

Figure 7.14: Type Nz definition
import FUZZIFICATION
type Tri_fuzzy_image = Integer * Integer -> Fuzzy_number

Figure 7.15: Specification of Fuzzy Set

op uni_min : Uni_intvl * Uni_intvl -> Uni_intvl
op uni_max : Uni_intvl * Uni_intvl -> Uni_intvl
def uni_min(u1,u2) = if u1 <= u2 then u1 else u2
def uni_max(u1,u2) = if u2 <= u1 then u2 else u1

Figure 7.16: Declaration and definition of uni_min and uni_max

To generate a triangular membership function for each crisp value, an operation named tri_fuzzify is defined as shown in figure 7.17. The uncertainty level(delta) is given as an operation (constant). Definition of tri_fuzzify_2 deals with the situation when uncertainty level is zero.

op tri_fuzzify : Nat * Nz -> Fuzzy_number
op tri_fuzzify_2 : Nat -> Fuzzy_number
op tri_fuzzify_image : Image * Nz -> Tri_fuzzy_image
op fuzzify : Nat * Nz -> Fuzzy_number
op fuzzify_1 : Nat * Nz -> Fuzzy_number
op delta : Nz

Figure 7.17: Triangular fuzzification op declarations

For a given value c, the triangular fuzzy number A is defined, such that for all $z \in \mathbb{R}$, $A(z)$ satisfies the equation:

$$A(z) = \begin{cases} 
0, & \text{if } z < c - \delta \text{ or } z > c + \delta \\
\frac{(z-c+\delta)}{\delta}, & \text{if } c - \delta \leq z \leq c \\
\frac{(c+\delta-z)}{\delta}, & \text{if } c \leq z \leq c + \delta 
\end{cases}$$
Specifications for \textit{tri}fuzzify, and \textit{tri}fuzzify\textsubscript{2} are shown in figure 7.18.

\begin{verbatim}
axiom tri_fuzzify_def is
fa(e: Nat, f: Fuzzy_number)
(tri_fuzzify(e, delta) = f <=>
  (fa(x: Nat)(x < (e - delta) || (e + delta) < x <=
    f(x) = 0 || (e - delta) <= x && x <= e <=>
    f(x) = div((x - e + delta), (delta)) ||
    e <= x && x <=
    f(x) = div((e - x + delta), (delta)))))

axiom tri_fuzzify_2_def is
fa(e: Nat, f: Fuzzy_number)
(tri_fuzzify_2(e) = f <=>
  (fa(x: Nat)(x < e || e < x <=
    f(x) = 0 || x <= e && e <= x <=>
    f(x) = 1)))
\end{verbatim}

Figure 7.18: Definitions of \textit{tri}fuzzify and \textit{tri}fuzzify\textsubscript{2}

Operation \textit{tri}fuzzify\textsubscript{image} yields fuzzy images that have triangular membership degrees in type \textit{tri}fuzzy\textsubscript{image}.

\begin{verbatim}
axiom tri_fuzzify_image_def is
fa(I: Image, F:Tri_fuzzy_image)
(tri_fuzzify_image(I, delta) = F <=> (fa(x: Integer, y: Integer)
  (F(x,y) = tri_fuzzify(I(x, y), delta))))
\end{verbatim}

Figure 7.19: Op \textit{tri}fuzzify\textsubscript{image} definition

In order to use the triangular fuzzy numbers, the arithmetic equations on necessary operations were given subsection 7.2.3.

### 7.2.8 Defuzzification (\textit{defuzzification}\textsubscript{.sw})

The defuzzify operation is defined as:
\[
\text{defuzzify}(F) = \frac{(\inf(M) + \sup(M))}{2}
\]

where \( M \) = an interval \([z]\) s.t. \( F(z) = \text{height}(F) = 1 \)

Hence, specification of defuzzification is:

```reasoning
op defuzzify_1 : Fuzzy_number -> Nat
def defuzzify_1(F) = div((\inf(\text{height_intvl}(F)) + \sup(\text{height_intvl}(F))),(1 + 1))
```

Figure 7.20: Op \text{defuzzify}_1\text{ declaration and definition}

Here, \( \inf \) and \( \sup \) denote infimum and supremum. \text{height_intvl} is an operation takes a fuzzy number and yields a set of natural numbers. Specifications for \( \inf, \sup \) and \text{height_intvl} can be seen in figure 7.21.

```reasoning
op inf : Intvl -> Nat
op sup : Intvl -> Nat
op height_intvl : Fuzzy_number -> Intvl
axiom inf_def is
fa(I: Intvl, a: Nat, x: Nat)
(\inf(I) = a <=>
(x in? I) => a <= x)
axiom sup_def is
fa(I: Intvl, a: Nat, x: Nat)
(\sup(I) = a <=>
(x in? I) => x <= a)
axiom height_intvl_def is
fa(F: Fuzzy_number, I: Intvl, x: Nat)
(\text{height_intvl}(F) = I <=>
F(x) = 100 <=> (x in? I))
```

Figure 7.21: Declaration and definition of operations used in \text{defuzzify}_1

Another version of defuzzification is also specified by using \( \alpha - \text{cut} \) :

\[
\text{defuzzify}(F) = \frac{(\inf(M) + \sup(M))}{2}
\]
where \( M = \) an interval \([z]\) s.t. \( F(z) = ^{\alpha} F \)

Hence,

\[
\text{op } \text{defuzzify}_2 : \text{Fuzzy\_number} -> \text{Nat} \\
\text{def } \text{defuzzify}_2(F) = \text{div}(\text{inf}(\alpha\_intvl(F)) + \text{sup}(\alpha\_intvl(F))), (1 + 1)
\]

Figure 7.22: Op \text{defuzzify}_2 declaration and definition

Specifications for \( \text{inf} \) and \( \text{sup} \) are same as above. \( \alpha\_intvl \) definition is shown in figure 7.23.

\[
\text{op } \alpha\_intvl : \text{Fuzzy\_number} -> \text{Intvl} \\
\text{op } \alpha : \text{Uni\_intvl} % \alpha \text{ is constant} \\
\text{axiom } \alpha\_intvl\_def \text{ is} \\
\text{fa}(F: \text{Fuzzy\_number}, I: \text{Intvl}, x: \text{Nat}) \\
(\alpha\_intvl(F) = I <=> F(x) = \alpha <=> (x \text{ in? } I))
\]

Figure 7.23: Op \text{alpha\_intvl} declaration and definition

### 7.2.9 Fuzzy Edge (fuzzyEdge.sw)

In this part of specifications, \text{Laplacian}, \text{zerocrossing} and \text{local variance} are formulated and defined according to the arithmetic operations of triangular fuzzy numbers. The first part which has to be defined in the definition of \text{fuzzy\_edge\_point} is the Laplacian part. Since Laplacian is;

\[
\nabla^2 I(x, y) = I(x + 1, y) + I(x - 1, y) + I(x, y + 1) + I(x, y - 1) - 4I(x, y)
\]

Hence, Laplacian is specified like in figure 7.24.
CHAPTER 7. AN EXAMPLE: EDGE DETECTION ALGORITHM

\begin{verbatim}
op flap : Tri_fuzzy_image * Integer * Integer -> Fuzzy_number

axiom flap_def is
fa(F: Tri_fuzzy_image, x: Integer, y: Integer)
flap(F, x, y) =
fuzzy_sub (fuzzy_add (fuzzy_add (F((x+1), y),
F((x-1), y)),
fuzzy_add (F(x, (y+1)),
F(x, (y-1)))),
fuzzy_add (fuzzy_add (F(x,y), F(x,y)),
fuzzy_add (F(x,y), F(x,y))))
\end{verbatim}

Figure 7.24: Laplacian declaration and definition as an operation

In order to explain the definition of `flap`, it is decomposed into parts which are named as follows, and then the arithmetic operation definitions are substituted into particular parts.

\[
L_1 = F((x + 1), y) \\
L_2 = F((x - 1), y) \\
L_3 = F(x, (y + 1)) \\
L_4 = F(x, (y - 1)) \\
L_5 = F(x, y) \\
L_{12} = fuzzy_add(F((x + 1), y), F((x - 1), y)) \\
L_{34} = fuzzy_add(F(x, (y + 1)), F(x, (y - 1))) \\
L_{1234} = fuzzy_add(L_{12}, L_{34}) \\
L_6 = fuzzy_mult(4, L_5)
\]

\[
flap = fuzzy_sub(L_{1234}, L_6)
\]

After separating each part of the definition, the definition of fuzzy Laplacian operator can be substituted into the particular parts. Arithmetic equations of `flap`:

\[
\mu_{L_{12}}(z) = \mu_{L_1}(z) \overline{+} \mu_{L_2}(z)
\]
\[ \mu_{L_{12}}(z) = \begin{cases} 0, & \text{if } z < L_1 + L_2 - 2\delta \text{ or } z > L_1 + L_2 + 2\delta \\ \frac{z-(L_1+L_2)+2\delta}{2\delta}, & \text{if } L_1 + L_2 - 2\delta \leq z \leq L_1 + L_2 \\ \frac{(L_1+L_2+2\delta-z)}{2\delta}, & \text{if } L_1 + L_2 \leq z \leq L_1 + L_2 + 2\delta \end{cases} \]

\[ \mu_{L_{34}}(z) = \mu_{L_3}(z) + \mu_{L_4}(z) \]

\[ \mu_{L_{34}}(z) = \begin{cases} 0, & \text{if } z < L_3 + L_4 - 2\delta \text{ or } z > L_3 + L_4 + 2\delta \\ \frac{z-(L_3+L_4)+2\delta}{2\delta}, & \text{if } L_3 + L_4 - 2\delta \leq z \leq L_3 + L_4 \\ \frac{(L_3+L_4+2\delta-z)}{2\delta}, & \text{if } L_3 + L_4 \leq z \leq L_3 + L_4 + 2\delta \end{cases} \]

\[ \mu_{L_{1234}}(z) = \mu_{L_{12}}(z) + \mu_{L_{34}}(z) \]

\[ \mu_{L_6}(z) = 4\delta \mu_{L_5}(z) \]

\[ \mu_{L_6}(z) = \begin{cases} 0, & \text{if } z < 4(L_5 - \delta) \\ \frac{z-4(L_5-\delta)}{4\delta}, & \text{if } 4(L_5 - \delta) \leq z \leq 4(L_5) \\ \frac{4(L_5+\delta)-z}{4\delta}, & \text{if } 4(L_5) \leq z \leq 4(L_5 + \delta) \end{cases} \]

\[ \mu_{\text{flop}}(z) = \mu_{L_{1234}}(z) - \mu_{L_6}(z) \]

\[ \mu_{\text{flop}}(z) = \begin{cases} 0, & \text{if } z < L_1 + L_2 + L_3 + L_4 - 4L_5 - 8\delta \\ \frac{z-(L_1+L_2+L_3+L_4-4L_5-8\delta)}{8\delta}, & \text{if } L_1 + L_2 + L_3 + L_4 - 4L_5 - 8\delta \leq z \leq L_1 + L_2 + L_3 + L_4 - 4L_5 \\ \frac{L_1+L_2+L_3+L_4-4L_5+8\delta-z}{8\delta}, & \text{if } L_1 + L_2 + L_3 + L_4 - 4L_5 \leq z \leq L_1 + L_2 + L_3 + L_4 + 4L_5 + 8\delta \end{cases} \]
Zero-crossing detection means comparing a pixel (say \((x, y)\)) with the adjacent ones on the next row and next column. Declaring zero-crossing points as edge results in a large number of points being declared to be edge points. In the specs, zero-crossing is divided into two parts; \(\text{zero}_1\) and \(\text{zero}_2\), which are fuzzified as shown in figure 7.25.

\[
\begin{align*}
\text{op fuzzy zero crossing}_1 : \\
\text{Tri fuzzy image} \times \text{Integer} \times \text{Integer} & \to \text{Uni intvl} \\
\text{axiom fuzzy zero crossing}_1 \text{ def is} \\
\text{fa}(F: \text{Tri fuzzy image}, x: \text{Integer}, y: \text{Integer}) \\
(\text{fuzzy zero crossing}_1(F, x, y) &= \\
(\text{uni min} (\text{fleq} (\text{flap}(F, x, y), \text{tri fuzzify}(0, \delta))), \\
(\text{uni max} (\text{fgeq} (\text{flap}(F, x, (y+1)), \text{tri fuzzify}(0, \delta))), \\
\text{fgeq} (\text{flap}(F, (x+1), y), \text{tri fuzzify}(0, \delta)))))) \\
\end{align*}
\]

\[
\begin{align*}
\text{op fuzzy zero crossing}_2 : \\
\text{Tri fuzzy image} \times \text{Integer} \times \text{Integer} & \to \text{Uni intvl} \\
\text{axiom fuzzy zero crossing}_2 \text{ def is} \\
\text{fa}(F: \text{Tri fuzzy image}, x: \text{Integer}, y: \text{Integer}) \\
(\text{fuzzy zero crossing}_2(F, x, y) &= \\
(\text{uni min} (\text{fgeq} (\text{flap}(F, x, y), \text{tri fuzzify}(0, \delta))), \\
(\text{uni max} (\text{fleq} (\text{flap}(F, x, (y+1)), \text{tri fuzzify}(0, \delta))), \\
\text{fleq} (\text{flap}(F, (x+1), y), \text{tri fuzzify}(0, \delta)))))) \\
\end{align*}
\]

Figure 7.25: Ops \(\text{fuzzy zero crossing}_1\) and \(\text{fuzzy zero crossing}_2\) declarations and definitions

For \(\text{fuzzy zero crossing}_1\):

\[
\mu_{\text{flap}}(z) \leq \mu_{\text{zero}}(z) = \inf \{ \min (1 - \mu_{\text{flap}}(z) + \mu_{\text{zero}}(z), 1) | z \in \text{Real} \}
\]

which has \text{Tri fuzzy image} as \((F, x, y)\).

Since \(\mu_{\text{flap}}(z), \mu_{\text{zero}}(z)\) are fuzzy numbers, it is obvious that the result will be

\[
(1 - \mu_{\text{flap}}(z) + \mu_{\text{zero}}(z)), \text{ if } (1 - \mu_{\text{flap}}(z) + \mu_{\text{zero}}(z)) \text{ is named as } l_1, \text{ then for } \text{Tri fuzzy image} \ (F, x, y),
\]

\[
\mu_{\text{flap}}(z) \leq \mu_{\text{zero}}(z) \equiv l_1
\]
\[ \mu_{\text{flap}}(z) \geq \mu_{\text{zero}}(z) = \inf \{ \min(1 - \mu_{\text{zero}}(z) + \mu_{\text{flap}}(z), 1) \mid z \in \text{Real} \} \]

which has Tri_fuzzy_image as \((F, x, y + 1)\).

Since \(\mu_{\text{flap}}(z), \mu_{\text{zero}}(z)\) are fuzzy_numbers, it is obvious that the result will be

\[ (1 - \mu_{\text{zero}}(z) + \mu_{\text{flap}}(z)), \text{ if } (1 - \mu_{\text{zero}}(z) + \mu_{\text{flap}}(z)) \text{ is named as } l_2, \text{ then} \]

for Tri_fuzzy_image \((F, x, y + 1)\),

\[ \mu_{\text{flap}}(z) \geq \mu_{\text{zero}}(z) \equiv l_2 \]

\[ \mu_{\text{flap}}(z) \geq \mu_{\text{zero}}(z) = \inf \{ \min(1 - \mu_{\text{zero}}(z) + \mu_{\text{flap}}(z), 1) \mid z \in \text{Real} \} \]

which has Tri_fuzzy_image as \((F, x + 1, y)\).

Since \(\mu_{\text{flap}}(z), \mu_{\text{zero}}(z)\) are fuzzy_numbers, it is obvious that the result will be

\[ (1 - \mu_{\text{zero}}(z) + \mu_{\text{flap}}(z)), \text{ if } (1 - \mu_{\text{zero}}(z) + \mu_{\text{flap}}(z)) \text{ is named as } l_3, \text{ then} \]

for Tri_fuzzy_image \((F, x + 1, y)\),

\[ \mu_{\text{flap}}(z) \geq \mu_{\text{zero}}(z) \equiv l_3 \]

and it is known from the specs that

\(\text{op uni}_{\text{min}} : \text{Uni}_{\text{intvl}}, \text{Uni}_{\text{intvl}} \rightarrow \text{Uni}_{\text{intvl}}\)

\(\text{op uni}_{\text{max}} : \text{Uni}_{\text{intvl}}, \text{Uni}_{\text{intvl}} \rightarrow \text{Uni}_{\text{intvl}}\)

The notation \(\wedge \equiv \text{uni}_{\text{min}}\) and \(\vee \equiv \text{uni}_{\text{max}}\) are used for the operators above.

\[ \wedge : [0, 1] \times [0, 1] \rightarrow [0, 1] \]

\[ a \wedge b = \min(a, b) \]

\[ \vee : [0, 1] \times [0, 1] \rightarrow [0, 1] \]

\[ a \vee b = \max(a, b) \]

So, fuzzy_zero_crossing_1 can be written as

\[ \text{fuzzy}_\text{zero}_\text{crossing}_1(F, x, y) = l_1 \wedge (l_2 \vee l_3), \text{ if } (l_2 \vee l_3) \equiv l_4 \text{ then} \]

\[ \text{fuzzy}_\text{zero}_\text{crossing}_1(F, x, y) = \min(l_1, (\max(l_2, l_3) = \min(l_1, l_4)) = t_1 \]

And for fuzzy_zero_crossing_2
\( \mu_{\text{flap}}(z) \geq \mu_{\text{zero}}(z) = \inf\{ \min(1 - \mu_{\text{zero}}(z) + \mu_{\text{flap}}(z), 1) | z \in \text{Real} \} \) which has Tri_fuzzy_image as \((F, x, y)\).

Since \( \mu_{\text{flap}}(z), \mu_{\text{zero}}(z) \) are fuzzy_numbers, it is obvious that the result will be

\[(1 - \mu_{\text{zero}}(z) + \mu_{\text{flap}}(z)), \text{ if } (1 - \mu_{\text{zero}}(z) + \mu_{\text{flap}}(z)) \text{ is named as } g_1, \text{ then for Tri_fuzzy_image } (F, x, y), \]

\( \mu_{\text{flap}}(z) \geq \mu_{\text{zero}}(z) \equiv g_1 \)

\( \mu_{\text{flap}}(z) \leq \mu_{\text{zero}}(z) = \inf\{ \min(1 - \mu_{\text{flap}}(z) + \mu_{\text{zero}}(z), 1) | z \in \text{Real} \} \) which has Tri_fuzzy_image as \((F, x, y + 1)\).

Since \( \mu_{\text{flap}}(z), \mu_{\text{zero}}(z) \) are fuzzy_numbers, it is obvious that the result will be

\[(1 - \mu_{\text{flap}}(z) + \mu_{\text{zero}}(z)), \text{ if } (1 - \mu_{\text{flap}}(z) + \mu_{\text{zero}}(z)) \text{ is named as } g_2, \text{ then for Tri_fuzzy_image } (F, x, y + 1), \]

\( \mu_{\text{flap}}(z) \leq \mu_{\text{zero}}(z) \equiv g_2 \)

\( \mu_{\text{flap}}(z) \leq \mu_{\text{zero}}(z) = \inf\{ \min(1 - \mu_{\text{flap}}(z) + \mu_{\text{zero}}(z), 1) | z \in \text{Real} \} \) which has Tri_fuzzy_image as \((F, x + 1, y)\).

Since \( \mu_{\text{flap}}(z), \mu_{\text{zero}}(z) \) are fuzzy_numbers, it is obvious that the result will be

\[(1 - \mu_{\text{flap}}(z) + \mu_{\text{zero}}(z)), \text{ if } (1 - \mu_{\text{flap}}(z) + \mu_{\text{zero}}(z)) \text{ is named as } g_3, \text{ then for Tri_fuzzy_image } (F, x + 1, y), \]

\( \mu_{\text{flap}}(z) \leq \mu_{\text{zero}}(z) \equiv g_3 \)

So, fuzzy_zero_crossing_2 can be written as

\[ \text{fuzzy_zero_crossing}_2(F, x, y) = g_1 \land (g_2 \lor g_3), \text{ if } (g_2 \lor g_3) \equiv g_4 \text{ then } \]

\[ \text{fuzzy_zero_crossing}_2(F, x, y) = \min(g_1, \max(g_2, g_3) = \min(g_1, g_4) = t_2 \]

Fuzzy variance part consists of five operations. They are declared in figure 7.26.
The local variance is
\[
\sigma^2(x, y) = \frac{1}{(2M + 1)^2} \sum_{k_1 = x - M}^{x + M} \sum_{k_2 = y - M}^{y + M} [(I(k_1, k_2) - m(k_1, k_2))^2]
\]

where
\[
m(x, y) = \frac{1}{(2M + 1)^2} \sum_{k_1 = x - M}^{x + M} \sum_{k_2 = y - M}^{y + M} I(k_1, k_2)
\]

with \(M\) typically chosen around 2. Since \(\sigma^2(x, y)\) is compared with a threshold, the scaling factor can be eliminated. In the \(m\) formula, \(I(k_1, k_2)\) is the intensity value. Within the intervals given in the formula, there will be different values for each intensity. When the formula \(m\) is fuzzified, all these different intensity values will have the same uncertainty level, which is \(\delta\).

The result of the addition for two of these fuzzified intensity values will be
\[
\mu_{I_1 + I_2}(z) = \begin{cases} 
0, & \text{if } z < I_1 + I_2 - 2\delta \text{ or } z > I_1 + I_2 + 2\delta \\
\frac{z - (I_1 + I_2 + 2\delta)}{2\delta}, & \text{if } I_1 + I_2 - 2\delta \leq z \leq I_1 + I_2 \\
\frac{(I_1 + I_2 + 2\delta - z)}{2\delta}, & \text{if } I_1 + I_2 \leq z \leq I_1 + I_2 + 2\delta
\end{cases}
\]

where \(I_1, I_2\) are two different intensity values with same uncertainty level \(\delta\) (\(\delta \in N\)).

Since variance is going to be compared with a threshold, \(\text{threshold}\) is also fuzzified:
\[
\mu_{\text{threshold}}(z) = \begin{cases} 
0, & \text{if } z < I_{\text{threshold}} \text{ or } z > I_{\text{threshold}} \\
1, & \text{if } z = I_{\text{threshold}}
\end{cases}
\]
So, the comparison between fuzzy variance and fuzzy threshold can be written as follows:

\[ \mu_{\text{var}}(z) \geq \mu_{\text{threshold}}(z) = \inf\{\min(1 - \mu_{\text{threshold}}(z) + \mu_{\text{var}}(z)), 1)\} \]

Since \( \mu_{\text{var}}(z), \mu_{\text{threshold}}(z) \) are fuzzy-numbers, it is obvious that the result will be

\((1 - \mu_{\text{threshold}}(z) + \mu_{\text{var}}(z)), \text{if } (1 - \mu_{\text{threshold}}(z) + \mu_{\text{var}}(z)) \text{ is named as } a, \)

which is a Uni-intvl,

\[ \mu_{\text{var}}(z) \geq \mu_{\text{threshold}}(z) \equiv a \]

\(Fuzzy\_edge\_point(F, x, y) = uni_{\text{min}}(uni_{\text{max}}(t_1, t_2)), a)\) can be written. If \(t_1 \mathcal{\vee} t_2 \equiv \max(t_1, t_2) = b\), then

\(Fuzzy\_edge\_point(F, x, y) = a \mathcal{\wedge} b \equiv \min(a, b)\) will be a number in \([0, 1]\), where \(F: \text{Tri}\_fuzzy\_image, x, y: \text{Integer}\) and \(\delta \in N_z\). After all the parts of the definition of fuzzy_edge_point is defined as shown in figure 7.27.

```
{\text{op}\ Fuzzy\_edge\_point : \text{Tri}\_fuzzy\_image \times \text{Integer} \times \\
\text{Integer} \rightarrow \text{Uni\_intvl} \}
{\text{axiom} \ Fuzzy\_edge\_point\_def \ is}
{\text{fa}(F: \text{Tri}\_fuzzy\_image, x: \text{Integer}, y: \text{Integer})}
Fuzzy\_edge\_point(F, x, y) =
(\text{uni}_{\text{min}}(\text{uni}_{\text{max}}(\text{fuzzy}\_\text{zero}\_\text{crossing}\_1(F, x, y), \text{fuzzy}\_\text{zero}\_\text{crossing}\_2(F, x, y)), \text{fequal}(\text{flap}(F, x, y), \text{tri}\_\text{fuzzify}(0, \delta)))})
```

Figure 7.27: Op \(Fuzzy\_edge\_point\) declaration and definition

**Theorem**

In order to show that reasoning about the impact of the uncertainty of input information can be specified formally in every aspect of the fusion process, a theorem is implemented on the edge detection algorithm, which is:
\[ \delta_1 \geq \delta_2 \Rightarrow \text{Fuzzy\_edge\_point}_1 \geq \text{Fuzzy\_edge\_point}_2 \]

all \(x, y \in \text{Integer}, I \in \text{Image}, \) and \(\delta_1, \delta_2 \in \text{Nz}\). Where \(\delta_1\) and \(\delta_2\) are two different values chosen to fuzzify the input data and represent the uncertainty levels of the input information, and \(\text{Fuzzy\_edge\_point}_1\) and \(\text{Fuzzy\_edge\_point}_2\) are the generated uncertainty values for deriving the results of \(\text{Fuzzy\_edge\_point}(x, y)\) for two different fuzzified images.

```
theorem fuzzyEdge is
  (fa(x: Integer, y: Integer, I: Image, \delta_1: Nz, \delta_2: Nz)
   (\delta_2 \leq \delta_1) =>
   (\text{Fuzzy\_edge\_point}(\text{tri\_fuzzify\_image}(I, \delta_2), x, y))
   \leq
   (\text{Fuzzy\_edge\_point}(\text{tri\_fuzzify\_image}(I, \delta_1), x, y)))
```

Figure 7.28: Theorem fuzzyEdge

**7.2.10 Proving in SNARK and Isabelle**

As mentioned section 3.4, SNARK is an automated theorem prover. In SNARK, the user can make changes in execution process only by using options. With these capabilities, Theorem fuzzyEdge could not be proved by SNARK, although, some options were active. Theorem proving experiments with the fuzzyEdge theorem can be seen in Figure 7.29.

However, Isabelle is a powerful interactive theorem prover, the proof requires extensive knowledge and experience in logical calculus and Isabelle. In order to get Isabelle to prove theorem fuzzyEdge, all specifications are translated to Isabelle’s language. Some basic theorems were proved. Isabelle proof terms embedded specifications can be seen in figures between 7.31-7.37. In order to prove these theorems, apply (sim add: ) method is used.
As an example, Isabelle theory files for monotocity are shown in figures 7.37, 7.38, and 7.39:

Theorem fuzzyEdge could not be proved. Figure 7.37 shows first subgoal of theorem fuzzyEdge. In this subgoal, Fuzzy_edge_point and tri_fuzzify_image definitions are known, so it can be simplified by using these definitions.

Isabelle apply(simp add: axiom1 axiom2) rule are applied to first subgoal. Axioms: Fuzzy_edge_point_def, tri_fuzzify_image_subtype_constr, tri_fuzzify_image_def, fuzzy_zero_crossing_1_def, fuzzy_zero_crossing_2_def, and tri_fuzzify_def are used as shown in figure 7.37

After simplification apply induction rule is applied to delta1 and delta2. Then two more simplifications are applied. Lastly, apply auto rule is applied.
**THEOREM** complement is
\[ f(a(f: \text{Fuzzy set Nat}, x: \text{Nat})) \] fuzzy_complement(fuzzy_complement(f)) x = f x

**Proof** Isa
by (auto simp add: between_zero_one_p_def fuzzy_complement_def abc)
end-proof

Figure 7.31: Proof of theorem *Complement*

**THEOREM** commutativity is
\[ t\text{-norm}(f1, f2)(x) \] t_norm(f1, f2)(x) = t_norm(f2, f1)(x)

**Proof** Isa
by (simp add: between_zero_one_p_def t_norm_def min_def)
end-proof

Figure 7.32: Proof of theorem *Commutativity*

After these processes, there existed one more subgoal. Since it is about a hundred lines long, it was not put here but can be found in the appendix.
CHAPTER 7. AN EXAMPLE: EDGE DETECTION ALGORITHM

Theorem \textit{assosiativity} is
\[
\text{fa}(f_1: \text{Fuzzy\_set\ Nat}, f_2: \text{Fuzzy\_set\ Nat}, f_3: \text{Fuzzy\_set\ Nat}, x: \text{Nat})
\]
\[
t\text{norm}(f_1, t\text{\_norm}(f_2, f_3))(x) = t\text{\_norm}(t\text{\_norm}(f_1, f_2), f_3)(x)
\]

Proof Isa
by (simp add: between\_zero\_one\_p\_def t\_norm\_def min\_def)
end-proof

Figure 7.33: Proof of theorem \textit{Assosiativity}

Theorem \textit{monotonicity} is
\[
\text{fa}(f_1: \text{Fuzzy\_set\ Nat}, f_2: \text{Fuzzy\_set\ Nat}, f_3: \text{Fuzzy\_set\ Nat}, x: \text{Nat})
\]
\[
f_2(x) \leq f_3(x) \Rightarrow t\text{\_norm}(f_1, f_2)(x) \leq t\text{\_norm}(f_1, f_3)(x)
\]

Proof Isa
by (simp add: between\_zero\_one\_p\_def t\_norm\_def min\_def)
end-proof

Figure 7.34: Proof of theorem \textit{Monotonicity}

Theorem \textit{theol} is
\[
\text{fa}(e: \text{Nat}, f: \text{Fuzzy\_number}, x: \text{Nat})
\]
\[
fuzzify(10, \delta)(100) = 100
\]

Proof Isa:
by (simp add: fuzzify\_def)
end-proof

Figure 7.35: Proof of theorem \textit{Fuzzify}
**CHAPTER 7. AN EXAMPLE: EDGE DETECTION ALGORITHM**

80

```isar
category theorem theo2 is
fun fa (u1:Uni_intvl, u2: Uni_intvl)
  { uni_min(10, 2) = 2

proof Isa theo2_Obligation_subtype:
by (simp add: between_zero_one_p_def)
end-proof

proof Isa theo2_Obligation_subtype0:
by (simp add: between_zero_one_p_def)
end-proof

proof Isa theo2:
by (simp add: between_zero_one_p_def uni_min_def)
end-proof

done
```

Figure 7.36: Proof of theorem \( Uni\_min \)

```isar
category theorem monotonicity:
  "\<bracket>Fun\_PR between_zero_one_p (\langle\_\_\rangle nat \Rightarrow Uni\_intvl); Fun\_PR between_zero_one_p (\langle\_\_\rangle nat \Rightarrow Uni\_intvl);
  Fun\_PR between_zero_one_p (\langle\_\_\rangle nat \Rightarrow Uni\_intvl);
  \langle\_\_\rangle nat \Rightarrow Uni\_intvl)
  \langle\_\_\rangle nat \Rightarrow Uni\_intvl)
  \langle\_\_\rangle nat \Rightarrow Uni\_intvl)
  \langle\_\_\rangle nat \Rightarrow Uni\_intvl)

  apply (simp add: between_zero_one_p_def t_norm_def min_def)
```

done

Figure 7.37: Subgoal for Monotonicity.
CHAPTER 7. AN EXAMPLE: EDGE DETECTION ALGORITHM

Figure 7.38: Theorem \textit{Monotonicity} proved in step 1 in Isabelle.

Figure 7.39: Theorem \textit{Monotonicity}.

Figure 7.40: Theorem \textit{fuzzyEdge} proving step 1 in Isabelle.
CHAPTER 7. AN EXAMPLE: EDGE DETECTION ALGORITHM

Figure 7.41: Theorem `fuzzyEdge` proving step 2 in Isabelle.

```isar
proof (cases)
  1. \{ 0 < \delta1; \delta2 \leq \delta1; \}
  \begin{align*}
    & \text{apply \ (induct_tac \ \delta1)} \\
    & \text{apply \ (induct_tac \ \delta2)} \\
    & \text{apply \ (simp)} \\
    & \text{apply \ (simp)} \\
    & \text{apply \ (auto)}
  \end{align*}
```

Figure 7.42: Other steps in Isabelle.
Chapter 8

CONCLUSION

8.1 Conclusion

In the course of this work, we attempted to learn Formal Method tools starting from minimal knowledge about these tools. Since formal method tools are complex and not very well documented, it took a long time to learn how to use them. In addition to Specware, the SNARK theorem prover (that comes bundled with Specware) and the Isabelle theorem prover, were the formal method tools used in this thesis.

A verification process is consists of two main parts: specification and theorem proving. Specware is used to build specifications and SNARK and Isabelle are used to prove theorems. In this work, we found Specware is a powerful specification tool. Using Specware we were able to represent very complex computational processes and the logic behind their processing algorithms. Specware language (Metaslang) supports higher order logic and thus we were able to represent operations that are performed not only on various data types, but also on algorithms.

However, an insufficiency of examples and documentation is the biggest problem with Specware. For a long time, it was attempted to get SNARK to prove theorems, but it could prove nothing more than simple theorems. In
this process we found some bugs in SNARK. We came to this conclusion after a number of interactions with the developers of Specware and SNARK.

As a result, we changed the theorem prover to Isabelle. In contrast to SNARK, Isabelle is an interactive theorem prover. But Isabelle comes with an overwhelming load of documentation. Therefore, it is hard to find a logical starting point. Although Isabelle can prove some theorems automatically, for some more complex theorems it is possible to use tactics and other options by means of adding them into specifications. This feature is very useful in proving such theorems.

XEmacs is the software development environment for Specware. It integrates Specware and Isabelle in the same environment allowing the user to work with both tools within the same environment. XEmacs proved to be a very handy tool in this kind of work.

Metaslang is the specification language of Specware. It is not an imperative programming language. It is a declarative logic based language. Metaslang can be viewed as a language to express mathematics. Thus coding in Metaslang is actually the same as deriving formulas in mathematics.

In this project, first, an information system was specified abstractly then more complex specifications were built incrementally. Initially, the specifications were non-constructive, i.e., they were based on existential specifications. In other words, a logical formula was stating that a solution exists, without actually pointing to a specific solution. Such specifications cannot be easily translated into code. So the use of such specifications is somewhat limited. These kinds of specifications can be used for verifying that a specification is consistent, but not that it is correct.

To address this limitation, Specware provides constructs that allow for representing constructive specifications. One of the constructs that have to be used to develop such specifications is the construct of lambda form. It is part of
Lambda Calculus. In essence, Lambda Calculus allows to express and manipulate functions. Since a large class of algorithms are just functions, Lambda Calculus provides good basis for specifying such algorithms. Thus lambda forms were used extensively in developing the constructive specifications in this work.

The main goal of this work was to investigate the use of formal method tools in the software development process. It is a well known fact in software engineering that the cost of modification of software grows exponentially with the stage in the software development process. The cost of removing an error in a software system after its deployment is thousands of times more expensive than the cost of eliminating the cause of such an error in the specification phase. Thus from the economical point of view, it pays to invest in the initial phases, in particular in the specification phase, in order to avoid much higher cost in the final phases. The goal of using formal method tools in software development is just this.

Toward this goal, we have developed an library of specifications for capturing the processing of uncertainty (fuzzy information). Then we developed a specification of a fuzzy information processing system. We formulated a number of theorems to test the specification of this system. In the course of this work we were able to verify some of the features of the system. Some of the theorems were not proved. This might be due to two reasons - either our specification was not refined to a sufficient level of detail, or our analysis tools were not powerful enough. Or simply we did devote a sufficient amount of effort to this verification process. Whatever happens to be the case, it is clear that we were able to stress test the specification. Although we we did not pursue this task in this work, Specware can also generate code from a specification.
In conclusion, through our work in this thesis fuzzy set theory which represents uncertainty of categorization was specified formally with Metaslang in Specware. As an example, fuzzy information processing specifications were applied to an edge detection algorithm and to a theorem based on this specification. Although they are theoretically possible and some fundamental theorems were proved, Isabelle and SNARK failed to prove some theorems. Overall, we saw that it is possible to verify systems before they are built and it was found that Specware and Isabelle can be great formal method tools to build specifications and to verify them. We believe that our work will be useful for future Specware and Isabelle users and can be considered as a useful approach for information fusion system formalization and verification.

8.2 Future Works

- Proving of the Model Using Isabelle
  Some theorems of model, such as theorem fuzzyEdge, can be proved by using Isabelle after getting extensive knowledge and experience in logical calculus and Isabelle. Isabelle has lots of capabilities that have not been used in this thesis.

- Refinement and Code Generation
  We spent all of our effort to prove theorems and to write specifications constructively. However, we could build some concrete specifications; it is needed to refine the specifications further by considering entire specifications. Specware has the capability of code generation from specifications. So it is possible to generate executable Lisp code from refined specifications.

- Different Fuzzy Rules
Even though we implemented three fuzzification methods - *triangular*, *trapozoidal*, *gaussian* - the work was focused only on the triangular fuzzification method. As a future work, other fuzzification methods can be applied to edge detection algorithms. Furthermore, fuzzy reasoning specifications can be defined by using different methods other than Lukasiewicz.
Bibliography


[5] Scott A. DeLoach and Mieczyslaw M. Kokar, "Category Theory Approach to Fusion of Wavelet-Based Features".


### Appendix A

## Crisp To Fuzzy Mapping Table

<table>
<thead>
<tr>
<th>CRISP</th>
<th>FUZZY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td>Fuzzy Number (FN)</td>
</tr>
<tr>
<td>True, False</td>
<td>[0,1]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>+</th>
<th>$+: FN \times FN \rightarrow FN$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$(A+B)(z) = \sup_{z=x+y} \min[A(x), B(y)], \forall z \in \mathbb{R}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>-</th>
<th>$-: FN \times FN \rightarrow FN$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$(A-B)(z) = \sup_{z=x-y} \min[A(x), B(y)], \forall z \in \mathbb{R}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>·</th>
<th>$\cdot: FN \times FN \rightarrow FN$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$(A\cdot B)(z) = \sup_{z=x\cdot y} \min[A(x), B(y)], \forall z \in \mathbb{R}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>÷</th>
<th>$\div: FN \times FN \rightarrow FN$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$(A\div B)(z) = \sup_{z=x\div y} \min[A(x), B(y)], \forall z \in \mathbb{R}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>min</th>
<th>$\text{fuzzy} - \min: FN \times FN \rightarrow FN$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\text{fuzzy} - \min(A, B)(z) = \sup_{z=\min(x,y)} \min[A(x), B(y)], \forall z \in \mathbb{R}$</td>
</tr>
</tbody>
</table>
### APPENDIX A. CRISP TO FUZZY MAPPING TABLE

<table>
<thead>
<tr>
<th>$\max$</th>
<th>$fuzzy - max : FN \times FN \rightarrow FN$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$fuzzy - \max(A, B)(z) = \sup_{z = \max(x, y)} \min[A(x), B(y)], \forall z \in \mathbb{R}$</td>
</tr>
<tr>
<td>$\Rightarrow$</td>
<td>$\Rightarrow : [0, 1] \times [0, 1] \rightarrow [0, 1]$</td>
</tr>
<tr>
<td></td>
<td>1: $a \Rightarrow b = (1, 1 - a + b)$</td>
</tr>
<tr>
<td></td>
<td>2: $a \Rightarrow b = \begin{cases} 1, &amp; \text{if } a \leq b \ b &amp; \text{other} \end{cases}$</td>
</tr>
<tr>
<td>$\Leftarrow$</td>
<td>$\Leftarrow : [0, 1] \times [0, 1] \rightarrow [0, 1]$</td>
</tr>
<tr>
<td></td>
<td>1: $a \Leftarrow b = 1 -</td>
</tr>
<tr>
<td></td>
<td>2: $a \Leftarrow b = \begin{cases} 1, &amp; \text{if } a = b \ \min(a, b) &amp; \text{other} \end{cases}$</td>
</tr>
<tr>
<td>$\overset{\equiv}{=} : FN \times FN \rightarrow [0, 1]$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1: $\mu \overset{\equiv}{=} v = \inf {1 -</td>
</tr>
<tr>
<td></td>
<td>1: $\mu \overset{\equiv}{=} v = \inf {g(\mu(x), v(x))</td>
</tr>
<tr>
<td></td>
<td>where $g(a, b) = \begin{cases} 1, &amp; \text{if } a = b \ b &amp; \text{other} \end{cases}$</td>
</tr>
<tr>
<td>$\overset{\geq}{\geq} : FN \times FN \rightarrow [0, 1]$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$\mu \overset{\geq}{\geq} v = \inf {\min(1 - v(x) + \mu(x), 1)</td>
</tr>
<tr>
<td>$\overset{\leq}{\leq} : FN \times FN \rightarrow [0, 1]$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$\mu \overset{\leq}{\leq} v = \inf {\min(1 - \mu(x) + v(x), 1)</td>
</tr>
</tbody>
</table>
Appendix B

Formal Information Fusion

Library Specs 1

%%% Formal Information Fusion Library Specs
%%% Fuzzy Set specification
%%% Fuzzy Set is type: Fuzzy_set a = a -> Uni_intvl

%%% UNI_INTVL specification
UNI_INTVL = spec
  import Nat
    op between_zero_one?(x:Nat): Boolean =
      x < 100 && 0 <= x

%%% Uni_intvl is a type defined between 0 and 1.
  type Uni_intvl = (Nat | between_zero_one?)
endspec

%%% FUZZY_SET specification
FUZZY_SET = spec
  import /Library/General/FiniteSet, UNI_INTVL

%%% Definition of Fuzzy_set type
  type Fuzzy_set a = a -> Uni_intvl

%%% Declarations of Fuzzy Complement, t_norm, and t_conorm operations
  op [a] fuzzy_complement : Fuzzy_set a -> Fuzzy_set a
  op [a] t_norm : Fuzzy_set a * Fuzzy_set a -> Fuzzy_set a
  op [a] t_conorm : Fuzzy_set a * Fuzzy_set a -> Fuzzy_set a

%%% Definitions of Fuzzy Complement, t_norm, and t_conorm.
  def [a] fuzzy_complement(f1) = fn d -> 100 - f1(d)
  def [a] t_norm(f1, f2) = fn d -> min(f1(d), f2(d))
  def [a] t_conorm (f1, f2) = fn d -> max(f1(d), f2(d))
%% Definitions of min and max
op min (x1: Nat, x2: Nat): Nat =  
  if x1 <= x2 then x1 else x2  
op max (x1: Nat, x2: Nat): Nat =  
  if x1 <= x2 then x2 else x1  

%% Declarations of t_norm_min, t_conorm_min, t_norm_Luka  
%% t_conorm_Luka, t_norm_prod, t_conorm_prod, alpha_cut operations  
op [a] t_norm_min : Fuzzy_set a * Fuzzy_set a -> Fuzzy_set a  
op [a] t_conorm_min : Fuzzy_set a * Fuzzy_set a -> Fuzzy_set a  
op [a] t_norm_Luka : Fuzzy_set a * Fuzzy_set a -> Fuzzy_set a  
op [a] t_conorm_Luka : Fuzzy_set a * Fuzzy_set a -> Fuzzy_set a  
op [a] t_norm_prod : Fuzzy_set a * Fuzzy_set a -> Fuzzy_set a  
op [a] t_conorm_prod : Fuzzy_set a * Fuzzy_set a -> Fuzzy_set a  
op [a] alpha_cut : Fuzzy_set a * Uni_intvl -> FiniteSet a  

%% Definitions of t_norm_min, t_conorm_min, t_norm_Luka  
%% t_conorm_Luka, t_norm_prod, t_conorm_prod, alpha_cut  
def [a] t_norm_min (f1, f2) = fn d -> min(f1(d), f2(d))  
def [a] t_conorm_min (f1, f2) = fn d -> max(f1(d), f2(d))  
def [a] t_norm_Luka(f1, f2) = fn d -> max(0, f1(d) + f2(d) - 100)  
def [a] t_conorm_Luka (f1, f2) = fn d -> min(f1(d) + f2(d), 100)  
def [a] t_norm_prod (f1, f2) = fn d -> f1(d) * f2(d)  
def [a] t_conorm_prod (f1, f2) = fn d -> (f1(d) + f2(d)) - (f1(d) * f2(d))  
def [a] alpha_cut (f, a) = fn d -> a <= f(d)  
def [a] height(f: Fuzzy_set a): Uni_intvl =  
  the(h) (fa(x: a) f x <= h) && (ex(x: a) f x = h)  
endspec  

%% Formal Information Fusion Library Specs  
%% Fuzzy number specification  
%% Fuzzy_number is imported from Fuzzy_set  
FUZZY_NUMBER = spec  
  import FUZZY_SET  
%% Definitions of Fuzzy_number and Set_of_Nat  
type Fuzzy_number = Fuzzy_set Nat  
type Set_of_Nat = FiniteSet Nat  
%% Fuzzy number is normal when height(f) has full membership degree  
axiom normality is  
  fa(f:Fuzzy_number)  
  height(f) = 100  
%% Fuzzy number is convex  
axiom convexity is  
  fa(f:Fuzzy_number, x1: Nat, x2: Nat, lamd: Uni_intvl)  
  λ x1, x2. (100 - lamd) * x2 >= min(f(x1), f(x2))  
endspec
%% FUZZY_ARITHM
%% In this part fuzzy arithmetic operations are specified

FUZZY_ARITHM = spec
  import FUZZY_NUMBER

  op fuzzy_add : Fuzzy_number * Fuzzy_number -> Fuzzy_number
  op fuzzy_sub : Fuzzy_number * Fuzzy_number -> Fuzzy_number
  op fuzzy_mult: Fuzzy_number * Fuzzy_number -> Fuzzy_number
  op fuzzy_div : Fuzzy_number * Fuzzy_number -> Fuzzy_number
  op fuzzy_min : Fuzzy_number * Fuzzy_number -> Fuzzy_number
  op fuzzy_max : Fuzzy_number * Fuzzy_number -> Fuzzy_number

%% Definition of Fuzzy Addition
axiom fuzzy_add_def is
  fa(f1:Fuzzy_number, f2:Fuzzy_number, x:Nat, y:Nat, z:Nat, a:Uni_intvl)
  fuzzy_add(f1, f2)(z) = a <=> z=x+y => (f1(x) <= a || f2(y) <= a) &&
    (ex(f1:Fuzzy_number, f2:Fuzzy_number, x:Nat, y:Nat, z:Nat, a:Uni_intvl)
      z=x+y && ((f1(x) = a && a <= f2(y)) || (f2(y) = a && a <= f1(x)))

%% Definition of Fuzzy Substraction
axiom fuzzy_sub_def is
  fa(f1:Fuzzy_number, f2:Fuzzy_number, x:Nat, y:Nat, z:Nat, a:Uni_intvl)
  fuzzy_sub(f1, f2)(z) = a <=> z=x-y => (f1(x) <= a || f2(y) <= a) &&
    (ex(f1:Fuzzy_number, f2:Fuzzy_number, x:Nat, y:Nat, z:Nat, a:Uni_intvl)
      z=x-y && ((f1(x) = a && a <= f2(y)) || (f2(y) = a && a <= f1(x)))

%% Definition of Fuzzy Multiplication
axiom fuzzy_mult_def is
  fa(f1:Fuzzy_number, f2:Fuzzy_number, x:Nat, y:Nat, z:Nat, a:Uni_intvl)
  fuzzy_mult(f1, f2)(z) = a <=> z=x*y => (f1(x) <= a || f2(y) <= a) &&
    (ex(f1:Fuzzy_number, f2:Fuzzy_number, x:Nat, y:Nat, z:Nat, a:Uni_intvl)
      z=x*y && ((f1(x) = a && a <= f2(y)) || (f2(y) = a && a <= f1(x)))

%% Definition of Fuzzy Division
axiom fuzzy_div_def is
  fa(f1:Fuzzy_number, f2:Fuzzy_number, x:Nat, y:Nat, z:Nat, a:Uni_intvl)
  fuzzy_div(f1, f2)(z) = a <=> x=z*y => (f1(x) <= a || f2(y) <= a) &&
    (ex(f1:Fuzzy_number, f2:Fuzzy_number, x:Nat, y:Nat, z:Nat, a:Uni_intvl)
      x=z*y && ((f1(x) = a && a <= f2(y)) || (f2(y) = a && a <= f1(x)))

%% Definition of Fuzzy Minimum
axiom fuzzy_min_def is
  fa(f1:Fuzzy_number, f2:Fuzzy_number, x:Nat, y:Nat, z:Nat, a:Uni_intvl)
  fuzzy_min(f1, f2)(z) = a <=> z= min(x,y) => (f1(x) <= a || f2(y) <= a) &&
    (ex(f1:Fuzzy_number, f2:Fuzzy_number, x:Nat, y:Nat, z:Nat, a:Uni_intvl)
      z= min(x,y) && ((f1(x) = a && a <= f2(y)) || (f2(y) = a && a <= f1(x)))

%% Definition of Fuzzy Maximum
axiom fuzzy_max_def is
  fa(f1:Fuzzy_number, f2:Fuzzy_number, x:Nat, y:Nat, z:Nat, a:Uni_intvl)
  fuzzy_max(f1, f2)(z) = a <=> z= max(x,y) => (f1(x) <= a || f2(y) <= a) &&
    (ex(f1:Fuzzy_number, f2:Fuzzy_number, x:Nat, y:Nat, z:Nat, a:Uni_intvl)
      z= max(x,y) && ((f1(x) = a && a <= f2(y)) || (f2(y) = a && a <= f1(x)))
%% Formal Information Fusion Library Specs
%% FUZZY_REASONING
%% Fuzzy reasoning operations are defined.

FUZZY_REASONING = spec
import FUZZY_ARITHM

op fequal : Fuzzy_number * Fuzzy_number -> Uni_intvl
op fgeq : Fuzzy_number * Fuzzy_number -> Uni_intvl
op fleq : Fuzzy_number * Fuzzy_number -> Uni_intvl
op inf : Nat -> Uni_intvl

%%% Definition of Fuzzy equal
axiom fequal_def is
fa(f1:Fuzzy_number, f2:Fuzzy_number, x:Nat)
fequal(f1, f2) = inf (100 - (abs(f1(x) - f2(x))))

%%% Definition of Fuzzy greater than and equal
axiom fgeq_def is
fa(f1:Fuzzy_number, f2:Fuzzy_number, x:Nat)
fgeq(f1, f2) = inf (min ((100 - f1(x) + f2(x)),100))

%%% Definition of Fuzzy less than and equal
axiom fleq_def is
fa(f1:Fuzzy_number, f2:Fuzzy_number, x:Nat)
fleq(f1, f2) = inf (min ((100 - f2(x) + f1(x)),100))

endspec

%% Formal Information Fusion Library Specs
%% IMAGE
%% In this part image is defined.

IMAGE = spec
type Image = Integer * Integer -> Nat

endspec

%% Formal Information Fusion Library Specs
%% FUZZIFICATION
%% Fuzzification spec is defined.

FUZZIFICATION = spec
import IMAGE, FUZZY_REASONING
type Nz = {i:Nat | i~0}

endspec
%% Formal Information Fusion Library Specs  
%% TRI FUZZIFY  
%% Triangular fuzzification is defined.

TRI FUZZIFY = spec
   import FUZZIFICATION

type Tri_fuzzy_image = Integer * Integer -> Fuzzy_number

op uni_min : Uni_intvl * Uni_intvl -> Uni_intvl
op uni_max : Uni_intvl * Uni_intvl -> Uni_intvl
op tri_fuzzify : Nat * Nz -> Fuzzy_number
op tri_fuzzify_2 : Nat -> Fuzzy_number
op tri_fuzzify_image : Image * Nz -> Tri_fuzzy_image
op fuzzify : Nat * Nz -> Fuzzy_number
op fuzzify_1 : Nat * Nz -> Fuzzy_number

%% Uncertainty level is given as a const
%% delta is the uncertainty level of the fuzzification
op delta : Nz

def uni_min(u1,u2) = if u1 <= u2 then u1 else u2
def uni_max(u1,u2) = if u2 <= u1 then u2 else u1

%% tri_fuzzify is to generate a triangular membership
%% function for each crisp value, define an Around label.
axiom tri_fuzzify_def is
   fa(e: Nat, f: Fuzzy_number)
   (tri_fuzzify(e, delta) = f <=> (fa(x: Nat)
      (x < (e - delta) || (e + delta) < x =>
         f(x) = 0 || (e - delta) <= x && x <= e =>
         f(x) = div((x - e + delta), (delta)) || e <= x && x <= (e + delta) =>
         f(x) = div((e - x + delta), (delta))))))

%% tri_fuzzify_2 deals with the situation when
%% uncertainty level is zero
axiom tri_fuzzify_2_def is
   fa(e: Nat, f: Fuzzy_number)
   (tri_fuzzify_2(e) = f <=> (fa(x: Nat)
      (x < e || e < x =>
         f(x) = 0 || x <= e && e <= x =>
         f(x) = 1))))

%% Definition of image fuzzification
axiom tri_fuzzify_image_def is
   fa(I: Image, F:Tri_fuzzy_image)
   (tri_fuzzify_image(I, delta) = F <=>
      (fa(x: Integer, y: Integer)
         (F(x,y) = tri_fuzzify(I(x, y), delta))))

%% Definitions of fuzzify and fuzzify_1
axiom fuzzify_def is
   fa(e: Nat, f: Fuzzy_number, x: Nat)
   (fuzzify(e, delta)(x) = tri_fuzzify(e, delta)(x) <=>
     x < e || fuzzify(e, delta)(x) = 100)
e <= x)

axiom fuzzify_1_def is
fa(e: Nat, f: Fuzzy_number, x: Nat)
(fuzzify(e, delta)(x) = tri_fuzzify(e, delta)(x) <=>
  e < x || fuzzify(e, delta)(x) = 100 <=>
  x <= e)
endspec

%% Formal Information Fusion Library Specs
%% TRA FUZZIFY
%% Trapezoidal fuzzification is defined.

TRA FUZZIFY = spec
import FUZZIFICATION
type Tra_fuzzy_image = Integer * Integer -> Fuzzy_number
op tra_fuzzify : Nat * Nz * Nz -> Fuzzy_number
op tra_fuzzify_2 : Nat -> Fuzzy_number
op tra_fuzzify_image : Image * Nz * Nz -> Tra_fuzzy_image

%% Uncertainty level is given as two consns
%% delta1 + delta2 is the uncertainty level of the fuzzification
op delta1: Nz
op delta2: Nz

%% op tra_fuzzify is to generate a trapezoidal membership
%% function for each crisp value, define an Around label.
axiom tri_fuzzify_def is
fa(e: Nat, f: Fuzzy_number)
(tra_fuzzify(e, delta1, delta2) = f <=>
  fa(x: Nat)
    (x<(e - (delta1+delta2)) || (e + (delta1+delta2))<x <=>
      f(x)=0 || (e - delta1)<=x && x<=(e + delta1) <=>
      f(x)=100 || (e - (delta1+delta2))<x && x<=(e - delta1) <=>
      f(x)=(x-(e - (delta1+delta2)))* inv(delta2) || (e+delta1)<=x &&
      x<=(e+(delta1+delta2)) <=>
      f(x)=(e+(delta1+delta2)-x)* inv(delta2))))

%% op tra_fuzzify_2 deals with the situation when
%% uncertainty level is zero
axiom tra_fuzzify_2_def is
fa(e: Nat, f: Fuzzy_number)
(tra_fuzzify_2(e) = f <=>
  fa(x: Nat)
    (x<e || e<x <=>
      f(x)=0 || x<=e && e=x <=>
      f(x)=100)))

%% definition of fuzzification of Image
axiom tra_fuzzify_image_def is
fa(I: Image, F:Tra_fuzzy_image)
(tra_fuzzify_image(I, delta1, delta2) = F <=>
  (fa(x: Integer, y: Integer)
   (F(x,y) = tra_fuzzify(I(x, y), delta1, delta2)))
endspec

%% Formal Information Fusion Library Specs
%% GAUSS_FUZZIFY
%% Gaussian fuzzification is defined.

GAUSS_FUZZIFY = spec
  import FUZZIFICATION
  type Gauss_fuzzy_image = Integer * Integer -> Fuzzy_number
  op gauss_fuzzify : Nat * Nz * Nz -> Fuzzy_number
  op gauss_fuzzify_2 : Nat -> Fuzzy_number
  op gauss_fuzzify_image : Image * Nz * Nz -> Gauss_fuzzy_image
  op exp : Nat -> Nat
  op square : Nat -> Nat

%% Uncertainty level is given as two consts
  op sigma : Nz
  op mu : Nz

%% op gauss_fuzzify is to generate a Gaussian membership
%% function for each crisp value, define an Around label.
  axiom gauss_fuzzify_def is
    fa(e: Nat, f: Fuzzy_number)
     (gauss_fuzzify(e, sigma, mu) = f <=>
      (fa(x: Nat)
       f(x) = exp(-square((e-mu)*inv(sigma)))))

%% op gauss_fuzzify_2 deals with the situation when
%% uncertainty level is zero
  axiom gauss_fuzzify_2_def is
    fa(e: Nat, f: Fuzzy_number)
     (gauss_fuzzify_2(e) = f <=> (fa(x: Nat)
        (x<e || e<x <=>
         f(x)=zero || x<=e && e<=x <=>
         f(x)=one)))

%% definition of fuzzification of Image
  axiom gauss_fuzzify_image_def is
    fa(I: Image, F: Gauss_fuzzy_image)
     (gauss_fuzzify_image(I, sigma, mu) = F <=>
      (fa(x:Integer, y:Integer)
       (F(x,y) = gauss_fuzzify(I(x,y), sigma, mu))))
endspec
%% Defuzzification is defined.

```
DEFUZZIFICATION = spec
  import FUZZY_NUMBER

%% type Intvl is a set of natural numbers
  type Intvl = Set_of_Nat

  op defuzzify_1 : Fuzzy_number -> Nat
  op inf : Intvl -> Nat
  op sup : Intvl -> Nat
  op height_intvl : Fuzzy_number -> Intvl
  op div : Nat * Nat -> Nat

%% defuzzify(F) = (inf(M) + sup(M))/2
%% M=an interval [z] s.t. F(z) = height(F) = one
  def defuzzify_1(F) =
    div((inf(height_intvl(F))+ sup(height_intvl(F))),(1 + 1))

axiom inf_def is
  fa(I: Intvl, a: Nat, x: Nat)
    (inf(I) = a <=>
      (x in? I) => a <= x)

axiom sup_def is
  fa(I: Intvl, a: Nat, x: Nat)
    (sup(I) = a <=>
      (x in? I) => x <= a)

axiom height_intvl_def is
  fa(F: Fuzzy_number, I: Intvl, x: Nat)
    (height_intvl(F) = I <=>
      F(x) = 100 <=> (x in? I))
```

endspec

%% Formal Information Fusion Library Specs
%% DEFUZZIFICATION 2
%% Defuzzification is defined.

```
DEFUZZIFICATION_2 = spec
  import FUZZY_NUMBER

  type Intvl = Set_of_Nat

  op defuzzify_2 : Fuzzy_number -> Nat
  op inf : Intvl -> Nat
  op sup : Intvl -> Nat
  op alpha_intvl : Fuzzy_number -> Intvl
  op div : Nat * Nat -> Nat

  op alpha : Uni_intvl

%% defuzzify_2(F) = (inf(M) + sup(M))/2
```
%% M = an interval [z] s.t. F(z) = alpha_cut of F

\[ \text{def defuzzify_2}(F) = \frac{\text{div}(\inf(\text{alpha_intvl}(F)) + \sup(\text{alpha_intvl}(F))), (1 + 1))}{1} \]

axiom inf_def is
fa(I: Intvl, a: Nat, x: Nat)
(\(\inf(I) = a \iff (x \in? I) \Rightarrow a \leq x\))

axiom sup_def is
fa(I: Intvl, a: Nat, x: Nat)
(\(\sup(I) = a \iff (x \in? I) \Rightarrow x \leq a\))

axiom alpha_intvl_def is
fa(F: Fuzzy_number, I: Intvl, x: Nat)
(\(\text{alpha_intvl}(F) = I \iff F(x) = \alpha \iff (x \in? I)\))

endspec

%% Formal Information Fusion Library Specs

%% EDGE_POINT

%% Edge_point is defined.

EDGE_POINT = spec
import IMAGE

%% Laplacian_based algorithm is used to derive edge point

op edge_point? : Image * Integer * Integer -> Boolean
op lap : Image * Integer * Integer -> Nat
op var : Image * Integer * Integer -> Nat
op ysum: Image * Integer * Integer -> Nat
op elmt: Image * Integer * Integer -> Nat
op m : Image * Integer * Integer -> Nat
op sub : Image * Integer * Integer -> Nat
op zero_crossing_1 : Image * Integer * Integer -> Boolean
op zero_crossing_2 : Image * Integer * Integer -> Boolean
op thrd : Nat

type Edge_point = (Image * Integer * Integer | edge_point?)

def lap(I, x, y) = (((I((x+1), y)+I((x-1), y)) + (I((y+1), x)+I((y-1), x))) - 
\(\text{((I(x,y)+I(x,y) + I(x,y)+I(x,y))))\))

def sub(I, x, y) = (((I(x, (y-(1+1))) + I(x, (y-1))) + (I(x, y) + I(x, (y+1)))) + 
\(\text{I(x, (y+(1+1))))\))

def m(I, x, y) = 
\(\text{((\text{sub}(I, (x-(i+1)), y) + \text{sub}(I, (x-1), y)) +}\)
(sub(I, x, y) + sub(I, (x+1), y)) + sub(I, (x+(1+1)), y))

def elmt(I, x, y) =
((I(x, y) - m(I, x, y))*(I(x, y) - m(I, x, y)))

def ysum(I, x, y) =
((((elmt(I, x, (y-(1+1)))) + elmt(I, x, (y-1))) +
  (elmt(I, x, y) + elmt(I, x, (y+1)))) + elmt(I, x, (y+(1+1))))

%% definition of local variance
def var(I, x, y) =
(((ysum(I, (x-(1+1)), y) + ysum(I, (x-1), y)) +
  (ysum(I, x, y) + ysum(I, (x+1), y))) + ysum(I, (x+(1+1)), y))

%% definition of zero_crossing
axiom zero_crossing_1_def is
fa(I: Image, x: Integer, y: Integer)
(zero_crossing_1(I, x, y) = ((lap(I,x,y) < 0) &&
  (0 < lap(I,x,(y+1))) ||
  (0 < lap(I,(x+1), y))))

axiom zero_crossing_2_def is
fa(I: Image, x: Integer, y: Integer)
(zero_crossing_2(I, x, y) = ((0 < lap(I, x, y)) &&
  (lap(I, x, (y+1)) < 0) ||
  (lap(I, (x+1), y) < 0)))

%% definition of edge_point
axiom edge_point_def is
fa(I: Image, x: Integer, y: Integer)
(edge_point?(I, x, y) <=>
  (zero_crossing_1(I,x,y) || zero_crossing_2(I,x,y))
  && (thrd < var(I, x, y)))
endspec

%% Formal Information Fusion Library Specs
%% EDGE
%% Edge is defined.
EDGE = spec
import EDGE_POINT
type Edge = (Image*Integer*Integer*Integer*Integer | edge?)
axiom edge?_def is
fa(I: Image, x1: Integer, y1: Integer, x2: Integer, y2: Integer)
(edge?(I, x1, y1, x2, y2) <=>
  (edge_point?(I, x1, y1) && edge_point?(I, x2, y2) &&
   ((y2 = y1 && (fa(x: Integer) (x <= x2 && x1 < x =>
     edge_point?(I, x, y1))) ||
    (x2 = x1 && (fa(y: Integer) (y <= y2 && y1 < y =>
      edge_point?(I, x1, y)))) ||
    (fa(x: Integer, y: Integer)
(y < y2 && y1 <= y && x < x2 && x1 <= x =>
  edge_point?(I, (x+1), (y+1))))))
endspec

%% Formal Information Fusion Library Specs
%% FUZZY_EDGE
%% how to derive fuzzy edge points
%% If (zero_crossing_1(I, x, y),zero_crossing_2(I,x,y)==1
%% and var(I, x, y) >= thrd
%% then (x, y) is an Edge_point.(crisp algorithm)
%% Fuzzy logic:
%% If zero_crossing_1(I, x, y)and zero_crossing_2(I, x, y) are true and
%% fvar(I, x, y) is fuzzy greater than thrd
%% then (x, y) is a Fuzzy_edge_point.

FUZZY_EDGE = spec
  import TRI_FUZZIFY
  type ONE_SORT
    op flap : Tri_fuzzy_image * Integer * Integer -> Fuzzy_number
    op fvar : Tri_fuzzy_image * Integer * Integer -> Fuzzy_number
    op fysum : Tri_fuzzy_image * Integer * Integer -> Fuzzy_number
    op felmt : Tri_fuzzy_image * Integer * Integer -> Fuzzy_number
    op fm : Tri_fuzzy_image * Integer * Integer -> Fuzzy_number
    op fsub : Tri_fuzzy_image * Integer * Integer -> Fuzzy_number
    op Fuzzy_edge_point : Tri_fuzzy_image * Integer * Integer -> Uni_intvl
    op Tri_fuzzy_image : Integer * Integer -> Fuzzy_number
    op fuzzy_zero_crossing_1 : Tri_fuzzy_image * Integer * Integer -> Uni_intvl
    op fuzzy_zero_crossing_2 : Tri_fuzzy_image * Integer * Integer -> Uni_intvl
    op thrd: Nat
  type Fuzzy_edge_point = Tri_fuzzy_image * Integer * Integer -> Uni_intvl

%% definition of fuzzy Laplacian
  def flap(F, x, y) =
    fuzzy_sub (fuzzy_add (fuzzy_add (F((x+1), y),
                                F((x-1), y)),
              fuzzy_add (F(x, (y+1)),
                         F(x, (y-1)))),
             fuzzy_add (fuzzy_add (F(x,y), F(x,y)),
                        fuzzy_add (F(x,y), F(x,y))))

  def fsub(F, x, y) =
    fuzzy_add (fuzzy_add (fuzzy_add (fsub(F, (x-(1+1)), y),
                                  F(x, (y-(1+1)))),
                     fuzzy_add (F(x, y),
                                F(x, (y-1)))),
              fuzzy_add (F(x, y),
                         F(x, (y+1))))

  def fm(F, x, y) =
    fuzzy_add (fuzzy_add (fuzzy_add (fsub(F, (x-(1+1)), y),
                                    fsub(F, (x-1), y)),
                       fuzzy_add (fuzzy_add (F((x+1), y),
                                       F((x-1), y))),
                      fuzzy_add (F((x+1), y),
                                 F((x-1), y))))
fuzzy_add (fsub(F, x, y),
    fsub(F, (x+1), y)),
    fsub(F, (x+(1+1)), y))

def felmt(F, x, y) =
    fuzzy_mult (fuzzy_sub (F(x, y), F(x, y)),
    fuzzy_sub (F(x, y), F(x, y)))

def fysum(F, x, y) =
    fuzzy_add (fuzzy_add (fuzzy_add (felmt(F, x, (y-(1+1))),
    felmt(F, x, (y-1))),
    fuzzy_add (felmt(F, x, y),
    felmt(F, x, (y+1))",
    felmt(F, x, (y+(1+1)))))

%% definition of fuzzy local variance
def fvar(F, x, y) =
    fuzzy_add (fuzzy_add (fuzzy_add (fysum(F, (x-(1+1)), y),
    fysum(F, (x-1), y)),
    fysum(F, x, y),
    fysum(F, (x+1), y)),
    fysum(F, (x+(1+1)), y))

%% fuzzy_zero_crossing
axiom fuzzy_zero_crossing_1_def is
    fa(F: Tri_fuzzy_image, x: Integer, y: Integer)
    (fuzzy_zero_crossing_1(F, x, y) =
    (uni_min (fleq (flap(F, x, y), tri_fuzzify(0, delta)),
    (uni_max (fgeq (flap(F, x, (y+1)), tri_fuzzify(0, delta)),
    fleq (flap(F, (x+1), y), tri_fuzzify(0, delta))))))

axiom fuzzy_zero_crossing_2_def is
    fa(F: Tri_fuzzy_image, x: Integer, y: Integer)
    (fuzzy_zero_crossing_2(F, x, y) =
    (uni_min (fgeq (flap(F, x, y), tri_fuzzify(0, delta)),
    (uni_max (fleq (flap(F, x, (y+1)), tri_fuzzify(0, delta)),
    fleq (flap(F, (x+1), y), tri_fuzzify(0, delta))))))

%% definition of Fuzzy edge point
def Fuzzy_edge_point(F, x, y) =
    (uni_min (uni_max (fuzzy_zero_crossing_1(F, x, y),
    fuzzy_zero_crossing_2(F, x, y)),
    fequal (flap(F, x, y), tri_fuzzify(0, delta))))

%% Theorem fuzzyEdge
theorem fuzzyEdge is
    (fa(x: Integer, y: Integer, I: Image, delta1: Nz, delta2: Nz)
    (delta2 <= delta1) =>
    (Fuzzy_edge_point(tri_fuzzify_image(I, delta2), x, y)) <=
    (Fuzzy_edge_point(tri_fuzzify_image(I, delta1), x, y)))
endspec

p1 = prove fuzzyEdge in FUZZY_EDGE
}
Appendix C

Formal Information Fusion
Library Specs 2

Specifications are transformed to concrete specifications and added theorems and Isabelle proof terms. While some parts stayed same some parts changed.

This part shows changed parts of specifications.

```plaintext
%% Formal Information Fusion Library Specs
%% Fuzzy Set specification
%% Fuzzy Set is type: Fuzzy_set a = a -> Uni_intvl

%% UNI_INTVL specification
UNI_INTVL = spec
%   import Nat
    op between_zero_one?(x:Nat): Boolean = x < 100 && 0 <= x

%% Uni_intvl is a type defined between 0 and 1.
   type Uni_intvl = (Nat | between_zero_one?)
endspec

%% FUZZY_SET specification
FUZZY_SET = spec
   import /Library/General/FiniteSet, UNI_INTVL

%% Definition of Fuzzy_set type
   type Fuzzy_set a = a -> Uni_intvl

%% Declarations of Basic Fuzzy Operations
   op [a] fuzzy_complement : Fuzzy_set a -> Fuzzy_set a
   op [a] t_norm : Fuzzy_set a * Fuzzy_set a -> Fuzzy_set a
   op [a] t_conorm : Fuzzy_set a * Fuzzy_set a -> Fuzzy_set a
   op [a] t_norm_min : Fuzzy_set a * Fuzzy_set a -> Fuzzy_set a
   op [a] t_conorm_min : Fuzzy_set a * Fuzzy_set a -> Fuzzy_set a
   op [a] t_norm_Luka : Fuzzy_set a * Fuzzy_set a -> Fuzzy_set a
   op [a] t_conorm_Luka : Fuzzy_set a * Fuzzy_set a -> Fuzzy_set a
   op [a] t_norm_prod : Fuzzy_set a * Fuzzy_set a -> Fuzzy_set a
   op [a] t_conorm_prod : Fuzzy_set a * Fuzzy_set a -> Fuzzy_set a
```

104
% Definitions of Basic Fuzzy Operations

def [a] fuzzy_complement(f1) d = 100 - f1(d)
def [a] t_norm(f1, f2) d = min(f1(d), f2(d))
def [a] t_conorm (f1, f2) d = max(f1(d), f2(d))
def [a] t_norm_min (f1, f2) d = min(f1(d), f2(d))
def [a] t_conorm_min (f1, f2) d = max(f1(d), f2(d))
def [a] t_norm_Luka(f1, f2) d = max(0, f1(d) + f2(d) - 100)
def [a] t_conorm_Luka (f1, f2) d = min(f1(d) + f2(d), 100)
def [a] t_norm_prod (f1, f2) d = f1(d) * f2(d)
def [a] t_conorm_prod (f1, f2) d = (f1(d) + f2(d)) - (f1(d) * f2(d))
def [a] alpha_cut (f, a) d = (a <= f(d))
def [a] height(f: Fuzzy_set Nat): Uni_intvl = the(h) (fa(x: Nat) f x <= h) && (ex(x: Nat) f x = h)

% Theorems and Isabelle proof terms

axiom abc is
fa(a: Nat, b: Nat)
a - (a - b) = b

% Fuzzy complement of fuzzy complement of a fuzzy set equals to itself.

theorem complement is
fa(f: Fuzzy_set Nat, x: Nat)
fuzzy_complement(fuzzy_complement(f)) x = f x

proof Isa
by (auto simp add: between_zero_one_p_def fuzzy_complement_def abc)
end-proof

% Fuzzy Set intersection/union provides commutativity.

theorem commutativity is
fa(f1: Fuzzy_set Nat, f2: Fuzzy_set Nat, x: Nat)
t_norm(f1, f2)(x) = t_norm(f2, f1)(x)

proof Isa
by (simp add: between_zero_one_p_def t_norm_def min_def)
end-proof

% Fuzzy Set intersection/union provides assosiativity.

theorem assosiativity is
t_norm(f1, t_norm(f2, f3))(x) = t_norm(t_norm(f1, f2), f3)(x)

proof Isa
by (simp add: between_zero_one_p_def t_norm_def min_def)
end-proof

% Fuzzy Set intersection/union provides monotonicity.

theorem monotonicity is
f2(x) <= f3(x) => t_norm(f1, f2)(x) <= t_norm(f1, f3)(x)
proof Isa
  by (simp add: between_zero_one_p_def t_norm_def min_def)
end-proof
endspec

%% Formal Information Fusion Library Specs
%% Fuzzy number specification
%% Fuzzy-number is imported from Fuzzy-set

FUZZY_NUMBER = spec
  import FUZZY_SET

%% Definitions of Fuzzy_number and Set_of_Real
  type Fuzzy_number = Fuzzy_set Nat
  type Set_of_Nat = FiniteSet Nat

%% Fuzzy number is normal when height(f) has full membership degree
  axiom normality is
    fa(f:Fuzzy_number)
    height(f) = 100

%% Fuzzy number is convex
  axiom convexity is
    fa(f:Fuzzy_number, x1:Nat, x2:Nat, lamd: Uni_intvl)
    f(lamd * x1) + ((100 - lamd) * x2) >= min(f(x1), f(x2))
endspec

%% Formal Information Fusion Library Specs
%% FUZZY-ARITHM
%% In this part fuzzy arithmetic operations are specified

FUZZY_ARITHM = spec
  import FUZZY_NUMBER

  op isMinIn (r:Nat, sr: Set Nat) infixl 20 : Bool =
    r in? sr && (fa(r1) r1 in? sr => r <= r1)

  op hasMin? (sr: Set Nat) : Bool = (ex(r) r isMinIn sr)

  op minIn (sr: Set Nat | hasMin? sr) : Nat = the(r) r isMinIn sr

  op rangeCC (lo:Nat, hi:Nat | lo <= hi) : Set Nat=
    fn r -> lo <= r && r <= hi

  op rangeOO (lo:Nat, hi:Nat | lo < hi) : Set Nat=
    fn r -> lo < r && r < hi

  op rangeCO (lo:Nat, hi:Nat | lo < hi) : Set Nat=
    fn r -> lo <= r && r < hi
app\x00e4ndix c. formal information fusion library specs

\textit{op rangeOC (lo: Nat, hi: Nat | lo < hi)} : Set Nat =
\begin{align*}
&fn r \to lo < r \&\& r \leq hi \\
\end{align*}

type Range = \{rs : Set Nat | ex(lo: Nat, hi: Nat) \\
&lo <= hi \&\& rs = rangeCC (lo, hi) \|\| \\
&lo < hi \&\& rs = rangeOD (lo, hi) \|\| \\
&lo < hi \&\& rs = rangeCO (lo, hi) \|\| \\
&lo < hi \&\& rs = rangeOC (lo, hi)\}

\textit{op sup (rng: Range)} : Nat =
\begin{align*}
&minIn (fn hi \to (fa(r) r \in? rng => hi >= r)) \\
\end{align*}

\begin{align*}
&\text{sup(map (fn (x,y) \to min(f1 x, f2 y)) (fn (x,y) \to x + y = z))} \\
\end{align*}

\begin{align*}
&\text{sup(map (fn (x,y) \to min(f1 x, f2 y)) (fn (x,y) \to x - y = z))} \\
\end{align*}

\begin{align*}
&\text{sup(map (fn (x,y) \to min(f1 x, f2 y)) (fn (x,y) \to x * y = z))} \\
\end{align*}

\begin{align*}
&\text{sup(map (fn (x,y) \to min(f1 x, f2 y)) (fn (x,y) \to z * y = x))} \\
\end{align*}

\begin{align*}
&\text{sup(map (fn (x,y) \to min(f1 x, f2 y)) (fn (x,y) \to min(x, y) = z))} \\
\end{align*}

\begin{align*}
&\text{sup(map (fn (x,y) \to min(f1 x, f2 y)) (fn (x,y) \to max(x, y) = z))} \\
\end{align*}

endspec

%%% Formal Information Fusion Library Specs
%%% FUZZY-REASONING
%%% Fuzzy reasoning operations are defined.

FUZZY\textunderscore REASONING = spec
import FUZZY\textunderscore ARITHM

%%% Definition of Fuzzy equal
\textit{op fequal (f1: Fuzzy_number, f2: Fuzzy_number)} : Uni_intvl =
\begin{align*}
&\text{(fa(x: Nat) (inf (100 - (abs(f1(x) - f2(x))))))} \\
\end{align*}

%%% Definition of Fuzzy greater than and equal
\textit{op fgeq (f1: Fuzzy_number, f2: Fuzzy_number)} : Uni_intvl =
\begin{align*}
&\text{(fa(x: Nat) inf (min ((100 + f1(x) - f2(x)), 100)))} \\
\end{align*}

%%% Definition of Fuzzy less than and equal
\textit{op fleq (f1: Fuzzy_number, f2: Fuzzy_number)} : Uni_intvl =
\begin{align*}
&\text{(fa(x: Nat) inf (min ((100 + f2(x) - f1(x)), 100)))} \\
\end{align*}

endspec
%% Formal Information Fusion Library Specs
%% IMAGE
%% In this part image is defined.

IMAGE = spec
  type Image = Integer * Integer -> Nat
endspec

%% Formal Information Fusion Library Specs
%% FUZZIFICATION
%% Fuzzification spec is defined.

FUZZIFICATION = spec
  import IMAGE, FUZZY_REASONING
  type Nz = {i:Nat | i!=0}
endspec

%% Formal Information Fusion Library Specs
%% TRI-FUZZIFY
%% Triangular fuzzification is defined.

TRI_FUZZIFY = spec
  import FUZZIFICATION
    type Tri_fuzzy_image = Integer * Integer -> Fuzzy_number
    op uni_min : Uni_intvl * Uni_intvl -> Uni_intvl
    op uni_max : Uni_intvl * Uni_intvl -> Uni_intvl
    op tri_fuzzify : Nat * Nz -> Fuzzy_number
    op tri_fuzzify_2 : Nat -> Fuzzy_number
    op tri_fuzzify_image : Image * Nz -> Tri_fuzzy_image
    op fuzzify : Nat * Nz -> Fuzzy_number
    op fuzzify_1 : Nat * Nz -> Fuzzy_number

%% Uncertainty level is given as a const
%% delta is the uncertainty level of the fuzzification
  op delta : Nz
  def uni_min(u1,u2) = if u1 <= u2 then u1 else u2
  def uni_max(u1,u2) = if u2 <= u1 then u2 else u1

%% op tri-fuzzify is to generate a triangular membership
%% function for each crisp value, define an Around label.
  axiom tri_fuzzify_def is
    fa(e: Nat, f: Fuzzy_number)
    (tri_fuzzify(e, delta) = f =>
      (fa(x: Nat)
        if ((e - delta) <= x && x <= e)
        then f(x) = div((x - e + delta), (delta))
        else if (e <= x && x <= (e + delta))
        then f(x) = div((e - x + delta), (delta))
    )
else f(x) = 0)))

%% op tri-fuzzify-2 deals with the situation when
%% uncertainty level is zero
axiom tri_fuzzify_2_def is
fa(e: Nat, f: Fuzzy_number)
(tri_fuzzify_2(e) = f =>
(fa(x: Nat)
  if (e <= x &x <= e)
    then f(x) = 100
    else f(x) = 0))

%% Definition of image fuzzification
axiom tri_fuzzify_image_def is
fa(I: Image, F:Tri_fuzzy_image)
(tri_fuzzify_image(I, delta) = F =>
(fa(x: Integer, y: Integer)
  (F(x,y) = tri_fuzzify(I(x, y), delta))))

%% Definitions of fuzzify and fuzzify-l
axiom fuzzify_def is
fa(e: Nat, f: Fuzzy_number, x: Nat)
(fuzzify(e, delta)(x) =
  (if (x < e)
    then tri_fuzzify(e, delta)(x)
    else 100))

axiom fuzzify_1_def is
fa(e: Nat, f: Fuzzy_number, x: Nat)
(fuzzify(e, delta)(x) =
  (if (e < x)
    then tri_fuzzify(e, delta)(x)
    else 100))

%% Theorems and Isabelle proof terms
theorem theo1 is
fa(e: Nat, f: Fuzzy_number, x: Nat)
fuzzify(10, delta)(100) = 100
proof Isa theo2_Obligation_subtype:
by (simp add: fuzzify_def)
end-proof

theorem theo2 is
fa(u1:Uni_intvl, u2: Uni_intvl)
uni_min(10, 2) = 2

%% Obligation subtypes are created by Isabelle and needed to prove too.
proof Isa theo2_Obligation_subtype:
by (simp add: between_zero_one_p_def)
end-proof

proof Isa theo2_Obligation_subtype0:
by (simp add: between_zero_one_p_def)
end-proof
proof Isa theo2:
by (simp add: between_zero_one_p_def uni_min_def)
end-proof

endspec

%% Formal Information Fusion Library Specs
%% DEFUZZIFICATION 1
%% Defuzzification is defined.
DEFUZZIFICATION = spec
  import FUZZY_NUMBER, FUZZY_ARITHM

%% type Intvl is a set of natural numbers
  type Intvl = Set_of_Nat
  op defuzzify_1 : Fuzzy_number -> Nat
  op inf : Intvl -> Nat
  op height_intvl : Fuzzy_number -> Intvl
  op div : Nat * Nat -> Nat

%% defuzzify(F) = (inf(M) + sup(M))/2
%% M=an interval [z] s.t. F(z) = height(F) = one
  def defuzzify_1(F) =
    div((inf(height_intvl(F))+ sup(height_intvl(F))),(1 + 1))

  axiom inf_def is
    fa(I: Intvl, a: Nat, x: Nat)
    (inf(I) = a <=>
      (x in? I) => a <= x)

  %% We do not need to define sup again here.
  %% sup is called from FUZZY_ARITHM
  axiom height_intvl_def is
    fa(F: Fuzzy_number, I: Intvl, x: Nat)
    (height_intvl(F) = I <=>
      F(x) = 100 <=> (x in? I))

endspec

%% Formal Information Fusion Library Specs
%% DEFUZZIFICATION 2
%% Defuzzification is defined.
DEFUZZIFICATION_2 = spec
  import FUZZY_NUMBER, FUZZY_ARITHM

  type Intvl = Set_of_Nat
  op defuzzify_2 : Fuzzy_number -> Nat
op inf : Intvl -> Nat
op alpha_intvl : Fuzzy_number -> Intvl
op div : Nat * Nat -> Nat

op alpha : Uni_intvl

%% defuzzify-2(F) = (inf(M) + sup(M))/2
%% M=an interval [z] s.t. F(z) = alpha-cut of F
def defuzzify_2(F) =
  div((inf(alpha_intvl(F)) + sup(alpha_intvl(F))),(1 + 1))

axiom inf_def is
fa(I: Intvl, a: Nat, x: Nat)
  (inf(I) = a <=>
   (x in? I) => a <= x)

%% sup is called from FUZZY_ARITHM

axiom alpha_intvl_def is
fa(F: Fuzzy_number, I: Intvl, x: Nat)
  (alpha_intvl(F) = I <=>
   F(x) = alpha <=> (x in? I))

endspec
Appendix D

Subgoal for Theorem FuzzyEdge Which Generated by Isabelle

proof (prove): step 1

goal (1 subgoal):
1. [| 0 < delta2; delta2 <= delta1 |]
   ==> uni_min
      (uni_max
       (uni_min
        (fleq (%z. sup__c (Set__map
                     (%(xa, ya).
                     FuzzyEdgeDetection2_FUZZY_SET.min
                    (sup__c (Set__map
                     (%(xa, ya).
                     FuzzyEdgeDetection2_FUZZY_SET.min
                    (sup__c (Set__map
                     (%(xa, ya).
                     FuzzyEdgeDetection2_FUZZY_SET.min
                    (tri_fuzzify_image (I, delta2) (x + 1, y) xa,
                    tri_fuzzify_image (I, delta2) (x - 1, y) ya))
                    (%(x, y). x + y = xa)),
                   sup__c (Set__map
                    (%(xa, ya).
                    FuzzyEdgeDetection2_FUZZY_SET.min
                    (tri_fuzzify_image (I, delta2) (x, y + 1) xa,
                    tri_fuzzify_image (I, delta2) (x, y - 1) ya))
                    (%(x, y). x + y = ya)))))
                    (%(x, y). x + y = xa)),
                   sup__c (Set__map
                    (%(xa, ya).
                    FuzzyEdgeDetection2_FUZZY_SET.min
                    (tri_fuzzify_image (I, delta2) (x, y) xa,
                    tri_fuzzify_image (I, delta2) (x, y) ya))
                    (%(x, y). x + y = xa)),
                   sup__c (Set__map
                    (%(xa, ya).
                    FuzzyEdgeDetection2_FUZZY_SET.min
                    (tri_fuzzify_image (I, delta2) (x, y) xa,)
tri_fuzzify_image (I, delta2) (x, y) ya))
(%(x, y). x + y = ya)))))
(%(x, y). x + y = ya)))))
(%(x, y). int x - int y = int z)),
  tri_fuzzify (0, delta)),
  uni_max
  (fgeq (%z. sup__c (Set__map
    (%(xa, ya).
      FuzzyEdgeDetection2_FUZZY_SET.min
    (sup__c (Set__map
      (%(xa, ya).
        FuzzyEdgeDetection2_FUZZY_SET.min
      (tri_fuzzify_image (I, delta2) (x + 1, y + 1) xa,
        tri_fuzzify_image (I, delta2) (x - 1, y + 1) ya))
    %(x, y). x + y = xa)),
     sup__c (Set__map
    %(xa, ya).
      FuzzyEdgeDetection2_FUZZY_SET.min
    (tri_fuzzify_image (I, delta2) (x, 2 + y) xa,
        tri_fuzzify_image (I, delta2) (x, y) ya))
    %(x, y). x + y = ya))))
  (%(x, y). x + y = xa)),
  sup__c (Set__map
  %(xa, ya).
    FuzzyEdgeDetection2_FUZZY_SET.min
  (tri_fuzzify_image (I, delta2) (2 + x, y) xa,
      tri_fuzzify_image (I, delta2) (x, y) ya))
  %(x, y). x + y = ya)))))
  (%(x, y). int x - int y = int z)),
    tri_fuzzify (0, delta)),
    fgeq (%z. sup__c (Set__map
      (%(xa, ya).
        FuzzyEdgeDetection2_FUZZY_SET.min
      (sup__c (Set__map
        %(xa, ya).
          FuzzyEdgeDetection2_FUZZY_SET.min
        (tri_fuzzify_image (I, delta2) (2 + x, y) xa,
            tri_fuzzify_image (I, delta2) (x, y) ya))
        %(x, y). x + y = xa)),
         sup__c (Set__map
        %(xa, ya).
          FuzzyEdgeDetection2_FUZZY_SET.min
        (tri_fuzzify_image (I, delta2) (2 + x, y) xa,
            tri_fuzzify_image (I, delta2) (x, y) ya))
        %(x, y). x + y = xa)),
         sup__c (Set__map
        %(xa, ya).
          FuzzyEdgeDetection2_FUZZY_SET.min
        (tri_fuzzify_image (I, delta2) (2 + x, y) xa,
            tri_fuzzify_image (I, delta2) (x, y) ya))
        %(x, y). x + y = xa)),
         sup__c (Set__map
        %(xa, ya).
          FuzzyEdgeDetection2_FUZZY_SET.min
        (tri_fuzzify_image (I, delta2) (2 + x, y) xa,
            tri_fuzzify_image (I, delta2) (x, y) ya))
        %(x, y). x + y = xa)),
         sup__c (Set__map
        %(xa, ya).
          FuzzyEdgeDetection2_FUZZY_SET.min
        (tri_fuzzify_image (I, delta2) (2 + x, y) xa,
            tri_fuzzify_image (I, delta2) (x, y) ya))
        %(x, y). x + y = xa)),
         sup__c (Set__map
        %(xa, ya).
          FuzzyEdgeDetection2_FUZZY_SET.min
        (tri_fuzzify_image (I, delta2) (2 + x, y) xa,
            tri_fuzzify_image (I, delta2) (x, y) ya))
        %(x, y). x + y = xa)),
         sup__c (Set__map
        %(xa, ya).
          FuzzyEdgeDetection2_FUZZY_SET.min
        (tri_fuzzify_image (I, delta2) (2 + x, y) xa,
            tri_fuzzify_image (I, delta2) (x, y) ya))
        %(x, y). x + y = xa)),
         sup__c (Set__map
        %(xa, ya).
          FuzzyEdgeDetection2_FUZZY_SET.min
APPENDIX D. SUBGOAL FOR THEOREM FUZZYEDGE WHICH GENERATED BY ISABELLE

(%(xa, ya).
    FuzzyEdgeDetection2_FUZZY_SET.min
    (tri_fuzzify_image (I, delta2) (x + 1, y + 1) xa,
     tri_fuzzify_image (I, delta2) (x + 1, y - 1) ya))
%(x, y). x + y = ya)))
%(x, y). x + y = xa)),
    sup_c (Set_map
     %(xa, ya).
     FuzzyEdgeDetection2_FUZZY_SET.min
     (sup_c (Set_map
      %(xa, ya).
      FuzzyEdgeDetection2_FUZZY_SET.min
      (tri_fuzzify_image (I, delta2) (x + 1, y) xa,
       tri_fuzzify_image (I, delta2) (x + 1, y) ya))
      %(x, y). x + y = ya)))
     %(x, y). x + y = xa)),
      sup_c (Set_map
       %(xa, ya).
       FuzzyEdgeDetection2_FUZZY_SET.min
       (tri_fuzzify_image (I, delta2) (x, y + 1) xa,
        tri_fuzzify_image (I, delta2) (x, y - 1) ya))
      %(x, y). x + y = ya)))),
    unim
    (fgeq (%z. sup_c (Set_map
     %(xa, ya).
     FuzzyEdgeDetection2_FUZZY_SET.min
     (sup_c (Set_map
      %(xa, ya).
      FuzzyEdgeDetection2_FUZZY_SET.min
      (tri_fuzzify_image (I, delta2) (x + 1, y) xa,
       tri_fuzzify_image (I, delta2) (x + 1, y) ya))
      %(x, y). x + y = ya)))),
     %(x, y). x + y = xa)),
    sup_c (Set_map
     %(xa, ya).
     FuzzyEdgeDetection2_FUZZY_SET.min
     (tri_fuzzify_image (I, delta2) (x, y + 1) xa,
      tri_fuzzify_image (I, delta2) (x, y - 1) ya))
     %(x, y). x + y = ya)))),
    sup_c (Set_map
     %(xa, ya).
     FuzzyEdgeDetection2_FUZZY_SET.min
     (sup_c (Set_map
      %(xa, ya).
      FuzzyEdgeDetection2_FUZZY_SET.min
      (tri_fuzzify_image (I, delta2) (x, y) xa,
       tri_fuzzify_image (I, delta2) (x, y) ya))
      %(x, y). x + y = xa)),
    sup_c (Set_map
     %(xa, ya).
     FuzzyEdgeDetection2_FUZZY_SET.min
     (sup_c (Set_map
      %(xa, ya).
      FuzzyEdgeDetection2_FUZZY_SET.min
      (tri_fuzzify_image (I, delta2) (x, y) xa,
       tri_fuzzify_image (I, delta2) (x, y) ya))
      %(x, y). x + y = ya))))
    %(x, y). x + y = xa)),
    sup_c (Set_map
     %(xa, ya).
     FuzzyEdgeDetection2_FUZZY_SET.min
     (sup_c (Set_map
      %(xa, ya).
      FuzzyEdgeDetection2_FUZZY_SET.min
      (tri_fuzzify_image (I, delta2) (x, y) xa,
       tri_fuzzify_image (I, delta2) (x, y) ya))
      %(x, y). x + y = xa)),
    sup_c (Set_map
     %(xa, ya).
     FuzzyEdgeDetection2_FUZZY_SET.min
     (sup_c (Set_map
      %(xa, ya).
      FuzzyEdgeDetection2_FUZZY_SET.min
      (tri_fuzzify_image (I, delta2) (x, y) xa,
       tri_fuzzify_image (I, delta2) (x, y) ya))
      %(x, y). x + y = xa)),
(tri_fuzzify_image (I, delta2) (x, y) xa, 
 tri_fuzzify_image (I, delta2) (x, y) ya)
(%(x, y). x + y = ya)))
  (%(x, y). x + y = ya)))
(%(x, y). int x - int y = int z),
  tri_fuzzify (0, delta)),
  uni_max
  (fleq (%z. sup__c (Set__map
  (%(xa, ya). 
    FuzzyEdgeDetection2_FUZZY_SET.min
    sup__c (Set__map
    (xa, ya).
    FuzzyEdgeDetection2_FUZZY_SET.min
    sup__c (Set__map
    (xa, ya).
    FuzzyEdgeDetection2_FUZZY_SET.min
    (tri_fuzzify_image (I, delta2) (x + 1, y + 1) xa, 
    tri_fuzzify_image (I, delta2) (x - 1, y + 1) ya))
(%(x, y). x + y = xa)),
  sup__c (Set__map
  (%(xa, ya).
  FuzzyEdgeDetection2_FUZZY_SET.min
  (tri_fuzzify_image (I, delta2) (x, 2 + y) xa, 
  tri_fuzzify_image (I, delta2) (x, y) ya))
(%(x, y). x + y = ya)))
  (%(x, y). x + y = xa)),
  sup__c (Set__map
  (%(xa, ya).
  FuzzyEdgeDetection2_FUZZY_SET.min
  (tri_fuzzify_image (I, delta2) (2 + x, y) xa, 
  tri_fuzzify_image (I, delta2) (x, y) ya))
(%(x, y). x + y = ya)))))
  (%(x, y). x + y = ya)))
  (%(x, y). int x - int y = int z),
  tri_fuzzify (0, delta)),
  fleq (%z. sup__c (Set__map
  (%(xa, ya).
  FuzzyEdgeDetection2_FUZZY_SET.min
  (sup__c (Set__map
  FuzzyEdgeDetection2_FUZZY_SET.min
  (sup__c (Set__map
  FuzzyEdgeDetection2_FUZZY_SET.min
  (tri_fuzzify_image (I, delta2) (2 + x, y) xa, 
  tri_fuzzify_image (I, delta2) (x, y) ya))
(%(x, y). x + y = xa)),

sup_c (Set_map
  (%(xa, ya).
   FuzzyEdgeDetection2_FUZZY_SET.min
   (tri_fuzzify_image (I, delta2) (x + 1, y + 1) xa,
    tri_fuzzify_image (I, delta2) (x + 1, y - 1) ya))
  (%(x, y). x + y = ya)))
  (%(x, y). x + y = xa)),
 sup_c (Set_map
  (%(xa, ya).
   FuzzyEdgeDetection2_FUZZY_SET.min
  (sup_c (Set_map
    (%(xa, ya).
     FuzzyEdgeDetection2_FUZZY_SET.min
     (tri_fuzzify_image (I, delta2) (x + 1, y) xa,
      tri_fuzzify_image (I, delta2) (x + 1, y) ya))
    (%(x, y). x + y = xa)),
 sup_c (Set_map
    (%(xa, y).
     FuzzyEdgeDetection2_FUZZY_SET.min
    (tr_fuzzify_image (I, delta2) (x + 1, y) xa,
     tri_fuzzify_image (I, delta2) (x + 1, y) ya))
    (%(x, y). x + y = ya)))
    (%(x, y). x + y = ya)))
    (%(x, y). int x - int y = int z),
     tri_fuzzify (0, delta))))),
FuzzyEdgeDetection2_FUZZY_REASONING.fequal
  (%z. sup_c (Set_map
    (%(xa, ya).
     FuzzyEdgeDetection2_FUZZY_SET.min
    (sup_c (Set_map
      (%(xa, y).
       FuzzyEdgeDetection2_FUZZY_SET.min
      (tri_fuzzify_image (I, delta2) (x + 1, y) xa,
       tri_fuzzify_image (I, delta2) (x - 1, y) ya))
      (%(x, y). x + y = xa)),
 sup_c (Set_map
      (%(xa, ya).
       FuzzyEdgeDetection2_FUZZY_SET.min
      (tri_fuzzify_image (I, delta2) (x, y + 1) xa,
       tri_fuzzify_image (I, delta2) (x, y - 1) ya))
      (%(x, y). x + y = ya)))
      (%(x, y). x + y = xa)),
 sup_c (Set_map
      (%(xa, ya).
       FuzzyEdgeDetection2_FUZZY_SET.min
      (tri_fuzzify_image (I, delta2) (x, y + 1) xa,
       tri_fuzzify_image (I, delta2) (x, y - 1) ya))
      (%(x, y). x + y = ya)))
      (%(x, y). x + y = xa)),
 sup_c (Set_map
      (%(xa, ya).
       FuzzyEdgeDetection2_FUZZY_SET.min
      (tri_fuzzify_image (I, delta2) (x, y) xa,
       tri_fuzzify_image (I, delta2) (x, y) ya))
      (%(x, y). x + y = xa)),
 sup_c (Set_map
      (%(xa, ya).
FuzzyEdgeDetection2_FUZZY_SET.min
(tri_fuzzify_image (I, delta2) (x, y) xa,
 tri_fuzzify_image (I, delta2) (x, y) ya)
(%(x, y). x + y = ya)))
  (%(x, y). x + y = ya)))
  (%(x, y). int x - int y = int z),
  tri_fuzzify (0, delta)))
<= uni_min
  (uni_max
     (uni_min
       (fleq (%z. sup__c (Set__map
         (%(xa, ya).
           FuzzyEdgeDetection2_FUZZY_SET.min
           (sup__c (Set__map
             (%(xa, ya).
               FuzzyEdgeDetection2_FUZZY_SET.min
               (tri_fuzzify_image (I, delta1) (x + 1, y) xa,
                tri_fuzzify_image (I, delta1) (x - 1, y) ya))
             (%(x, y). x + y = xa)),
             sup__c (Set__map
             (xa, ya).
             FuzzyEdgeDetection2_FUZZY_SET.min
             (tri_fuzzify_image (I, delta1) (x, y + 1) xa,
              tri_fuzzify_image (I, delta1) (x, y - 1) ya))
             (%(x, y). x + y = ya))),
             sup__c (Set__map
             (xa, ya).
             FuzzyEdgeDetection2_FUZZY_SET.min
             (tri_fuzzify_image (I, delta1) (x, y) xa,
              tri_fuzzify_image (I, delta1) (x, y) ya))
             (%(x, y). x + y = xa)),
             sup__c (Set__map
             (xa, ya).
             FuzzyEdgeDetection2_FUZZY_SET.min
             (tri_fuzzify_image (I, delta1) (x, y) xa,
              tri_fuzzify_image (I, delta1) (x, y) ya))
             (%(x, y). x + y = ya)))
             (%(x, y). x + y = ya)))
             (%(x, y). int x - int y = int z)),
             tri_fuzzify (0, delta))),
     ni_max
     (fgeq (%z. sup__c (Set__map
       (%(xa, ya).
       FuzzyEdgeDetection2_FUZZY_SET.min
       (sup__c (Set__map
         (%(xa, ya).
         FuzzyEdgeDetection2_FUZZY_SET.min
         (sup__c (Set__map
           (Set__map}
APPENDIX D. SUBGOAL FOR THEOREM FUZZYEDGE WHICH GENERATED BY ISABELLE

\%(xa, ya).
  FuzzyEdgeDetection2_FUZZY_SET.min
  (tri_fuzzify_image (I, delta1) (x + 1, y + 1) xa,
   tri_fuzzify_image (I, delta1) (x - 1, y + 1) ya)
\%(x, y). x + y = xa),
  sup__c
(Set__map
\%(xa, ya).
  FuzzyEdgeDetection2_FUZZY_SET.min
  (tri_fuzzify_image (I, delta1) (x, 2 + y) xa,
   tri_fuzzify_image (I, delta1) (x, y) ya)
\%(x, y). x + y = ya),
  sup__c (Set__map
\%(xa, ya).
  FuzzyEdgeDetection2_FUZZY_SET.min (sup__c
(Set__map
\%(xa, ya).
  FuzzyEdgeDetection2_FUZZY_SET.min
  (tri_fuzzify_image (I, delta1) (x, y + 1) xa,
   tri_fuzzify_image (I, delta1) (x, y + 1) ya)
\%(x, y). x + y = ya),
  sup__c
(Set__map
\%(xa, ya).
  FuzzyEdgeDetection2_FUZZY_SET.min
  (tri_fuzzify_image (I, delta1) (2 + x, y) xa,
   tri_fuzzify_image (I, delta1) (x, y) ya)
\%(x, y). x + y = xa),
  sup__c
(Set__map
\%(xa, ya).
  FuzzyEdgeDetection2_FUZZY_SET.min
  (tri_fuzzify_image (I, delta1) (x + 1, y + 1) xa,
   tri_fuzzify_image (I, delta1) (x + 1, y - 1) ya)
\%(x, y). x + y = ya),
  sup__c (Set__map
\%(xa, ya).
  FuzzyEdgeDetection2_FUZZY_SET.min (sup__c
(Set__map
\%(xa, ya).
APPENDIX D. SUBGOAL FOR THEOREM FUZZYEDGE WHICH GENERATED BY ISABELLE

FuzzyEdgeDetection2_FUZZY_SET.min
(tri_fuzzify_image (I, delta1) (x + 1, y) xa,
tri_fuzzify_image (I, delta1) (x + 1, y) ya)
(\%(x, y). x + y = xa),
sup_c
(Set_map
(\%(xa, ya). FuzzyEdgeDetection2_FUZZY_SET.min
(tri_fuzzify_image (I, delta1) (x + 1, y) xa,
tri_fuzzify_image (I, delta1) (x + 1, y) ya))
(\%(x, y). x + y = ya)))
(\%(x, y). int x - int y = int z)),
tri_fuzzify (0, delta))),
uni_min
(fgeq (%z. sup_c (Set_map
(\%(xa, ya). FuzzyEdgeDetection2_FUZZY_SET.min
sup_c (Set_map
(\%(xa, ya). FuzzyEdgeDetection2_FUZZY_SET.min
(tri_fuzzify_image (I, delta1) (x + 1, y) xa,
tri_fuzzify_image (I, delta1) (x + 1, y) ya))
(\%(x, y). x + y = xa)),
sup_c (Set_map
(\%(xa, ya). FuzzyEdgeDetection2_FUZZY_SET.min
(tri_fuzzify_image (I, delta1) (x + 1, y) xa,
tri_fuzzify_image (I, delta1) (x + 1, y) ya))
(\%(x, y). x + y = ya)))
(\%(x, y). x + y = ya))
(\%(x, y). int x - int y = int z)),
up_c (Set_map
(\%(xa, ya).
FuzzyEdgeDetection2_FUZZY_SET.min
(\%(xa, ya).
FuzzyEdgeDetection2_FUZZY_SET.min
(tri_fuzzify_image (I, delta1) (x, y) xa,
tri_fuzzify_image (I, delta1) (x, y) ya))
(\%(x, y). x + y = xa),
sup_c (Set_map
(\%(xa, ya).
FuzzyEdgeDetection2_FUZZY_SET.min
(tri_fuzzify_image (I, delta1) (x, y) xa,
tri_fuzzify_image (I, delta1) (x, y) ya))
(\%(x, y). x + y = ya)))
(\%(x, y). x + y = ya))
(\%(x, y). int x - int y = int z)),
tri_fuzzify (0, delta)),
uni_max
(fleq (%z. sup_c (Set_map
(\%(xa, ya).
FuzzyEdgeDetection2_FUZZY_SET.min
(sup\_c (Set\_map
  (\%(xa, ya).
   FuzzyEdgeDetection2\_FUZZY\_SET.min
   (sup\_c
    (Set\_map
     (\%(xa, ya).
      FuzzyEdgeDetection2\_FUZZY\_SET.min
      (tri\_fuzzify\_image (I, delta1) (x + 1, y + 1) xa,
       tri\_fuzzify\_image (I, delta1) (x - 1, y + 1) ya))
    (\%(x, y). x + y = xa)),
     sup\_c (Set\_map
      (\%(xa, ya).
       FuzzyEdgeDetection2\_FUZZY\_SET.min
       (sup\_c
        (Set\_map
         (\%(xa, ya).
          FuzzyEdgeDetection2\_FUZZY\_SET.min
          (tri\_fuzzify\_image (I, delta1) (x, 2 + y) xa,
           tri\_fuzzify\_image (I, delta1) (x, y) ya))
         (\%(x, y). x + y = xa)),
          sup\_c (Set\_map
           (\%(xa, ya).
            FuzzyEdgeDetection2\_FUZZY\_SET.min
            (sup\_c
             (Set\_map
              (\%(xa, ya).
               FuzzyEdgeDetection2\_FUZZY\_SET.min
               (tri\_fuzzify\_image (I, delta1) (x, y + 1) xa,
                tri\_fuzzify\_image (I, delta1) (x, y + 1) ya))
              (\%(x, y). x + y = xa)),
               sup\_c (Set\_map
                (\%(xa, ya).
                 FuzzyEdgeDetection2\_FUZZY\_SET.min
                 (sup\_c
                  (Set\_map
                   (\%(xa, ya).
                    FuzzyEdgeDetection2\_FUZZY\_SET.min
                    (tri\_fuzzify\_image (I, delta1) (x, y + 1) xa,
                     tri\_fuzzify\_image (I, delta1) (x, y + 1) ya))
                    (\%(x, y). x + y = xa)),
                    sup\_c (Set\_map
                     (\%(xa, ya).
                      FuzzyEdgeDetection2\_FUZZY\_SET.min
                      (sup\_c
                       (Set\_map
                        (\%(xa, ya).
                         FuzzyEdgeDetection2\_FUZZY\_SET.min
                         (tri\_fuzzify\_image (I, delta1) (2 + x, y) xa,
                          tri\_fuzzify\_image (I, delta1) (x, y) ya))
                         (\%(x, y). x + y = xa)),
                         sup\_c (Set\_map
                          (\%(xa, ya).
                           FuzzyEdgeDetection2\_FUZZY\_SET.min
                           (sup\_c
                            (Set\_map
                             (\%(xa, ya).
                              FuzzyEdgeDetection2\_FUZZY\_SET.min
                              (tri\_fuzzify\_image (I, delta1) (x + 1, y + 1) xa,
\[ \text{tri} \_ \text{fuzzify} \_ \text{image} \ (I, \ \text{delta1}) \ (x + 1, y - 1) \ (\text{ya}) \) \\
\%(x, y). \ x + y = \text{ya})\))
\%(x, y). \ x + y = \text{xa}),
\text{sup}_\_c \ (\text{Set}_\_\text{map}
\%(\text{xa}, \ \text{ya}).
\text{FuzzyEdgeDetection2}_\_\text{FUZZY}_\_\text{SET}.\text{min}
\text{sup}_\_c
\text{Set}_\_\text{map}
\%(\text{xa}, \ \text{ya}).
\text{FuzzyEdgeDetection2}_\_\text{FUZZY}_\_\text{SET}.\text{min}
\text{sup}_\_c
\text{Set}_\_\text{map}
\%(\text{xa}, \ \text{ya}).
\text{FuzzyEdgeDetection2}_\_\text{FUZZY}_\_\text{SET}.\text{min}
\text{sup}_\_c
\text{Set}_\_\text{map}
\%(\text{xa}, \ \text{ya}).
\text{FuzzyEdgeDetection2}_\_\text{FUZZY}_\_\text{SET}.\text{min}
\text{sup}_\_c
\text{Set}_\_\text{map}
\%(\text{xa}, \ \text{ya}).
\text{FuzzyEdgeDetection2}_\_\text{FUZZY}_\_\text{SET}.\text{min}
\text{sup}_\_c
\text{Set}_\_\text{map}
\%(\text{xa}, \ \text{ya}).
\text{FuzzyEdgeDetection2}_\_\text{FUZZY}_\_\text{SET}.\text{min}
\text{sup}_\_c
\text{Set}_\_\text{map}
\%(\text{xa}, \ \text{ya}).
\text{FuzzyEdgeDetection2}_\_\text{FUZZY}_\_\text{SET}.\text{min}
\text{sup}_\_c
\text{Set}_\_\text{map}
\%(\text{xa}, \ \text{ya}).
\text{FuzzyEdgeDetection2}_\_\text{FUZZY}_\_\text{SET}.\text{min}
\text{sup}_\_c
\text{Set}_\_\text{map}
\%(\text{xa}, \ \text{ya}).
\text{FuzzyEdgeDetection2}_\_\text{FUZZY}_\_\text{SET}.\text{min}
\text{sup}_\_c
\text{Set}_\_\text{map}
\%(\text{xa}, \ \text{ya}).
\text{FuzzyEdgeDetection2}_\_\text{FUZZY}_\_\text{SET}.\text{min}
\text{sup}_\_c
\text{Set}_\_\text{map}
\%(\text{xa}, \ \text{ya}).
\text{FuzzyEdgeDetection2}_\_\text{FUZZY}_\_\text{SET}.\text{min}
\text{sup}_\_c
\text{Set}_\_\text{map}
\%(\text{xa}, \ \text{ya}).
\text{FuzzyEdgeDetection2}_\_\text{FUZZY}_\_\text{SET}.\text{min}
\text{sup}_\_c
\text{Set}_\_\text{map}
\%(\text{xa}, \ \text{ya}).
\text{FuzzyEdgeDetection2}_\_\text{FUZZY}_\_\text{SET}.\text{min}
\text{sup}_\_c
\text{Set}_\_\text{map}
\%(\text{xa}, \ \text{ya}).
\text{FuzzyEdgeDetection2}_\_\text{FUZZY}_\_\text{SET}.\text{min}
\text{sup}_\_c
\text{Set}_\_\text{map}
\%(\text{xa}, \ \text{ya}).\]
APPENDIX D. SUBGOAL FOR THEOREM FUZZYEDGE WHICH GENERATED BY ISABELLE

\begin{verbatim}
tri_fuzzify_image (I, delta1) (x, y) ya))
  (\%(x, y). x + y = ya))
  (\%(x, y). x + y = ya))
  (\%(x, y). int x - int y = int z),
  tri_fuzzify (0, delta))
\end{verbatim}