Fuzzy Concepts and Formal Methods:
A Sample Specification for a Fuzzy Expert System

Chris Matthews
Department of Information Technology,
School of Business and Technology,
La Trobe University, P.O. Box 199 Bendigo 3552, Victoria, Australia.
Phone: +61 3 54447350
Fax: +61 3 54447998
email: c.matthews@ironbark.bendigo.latrobe.edu.au

Abstract - A fuzzy logic toolkit has been developed for the formal specification language Z. It permits the incorporation of fuzzy concepts into the language while retaining the precision of any Z specification. The toolkit provides the necessary operators, measures and modifiers for the definition and manipulation of fuzzy sets and relations. This paper illustrates how the toolkit can be used to specify a simple fuzzy expert system. The focus is on the specification of the rule base and the operations necessary for fuzzy inferencing. In particular the example illustrates the use of the fuzzy cartesian product and fuzzy set truncation operators and offers a generic definition for a centroid defuzzification function.

I. Introduction

Formal methods are a set of mathematical based techniques that allow the development of a complete, precise and correct specification for system behaviour and properties. They have been described as "methods that use formulas" [5]. A formula can be text (or even a diagram) that is formed by applying a set of explicit rules to a series of pre-defined symbols. The syntax of such a specification language can be checked by a computer. In this context any programming language can be seen as being a "formal" method. However unlike many of the more abstract specification languages, a programming language is executable and is designed with only limited attention being paid to the communication of the meaning of the specification to a human being. It is only through the execution of the program on a computer that the meaning becomes evident. If we use a programming language as the only formal specification tool in system development then errors or omissions in the specification may only become apparent at run-time. Programming languages focus attention on how a set of requirements are to be implemented, rather than what is required. The use of more abstract languages allows us to model and predict system behaviour, and to identify errors and omissions prior to implementation — "The whole point of formal methods is to be able to predict what a program will do without running any code — in fact, without writing any" [5].

One commonly used specification language is Z [13], [10]. It is a model-based formal specification language based on typed set theory and first order predicate calculus. System properties are modelled using set-theoretic concepts and the necessary pre- and post-conditions for each system operation, or change of state, are explicitly stated. The basic building block in any Z specification is the schema. This is a named grouping of components that can be referred to throughout the specification. Schemas provide structure for a specification and describe system operations and permissible system states. The schema calculus allows more complex components of a specification to be built from a set of previously defined and simpler schemas. Z is a powerful analytical tool that facilitates system understanding through the development of a series of unambiguous, verifiable mathematical models [5], [11]. Various levels of abstraction are possible. These may vary from a statement of requirements (to be used as a basis for communication and validation) to a more concrete software design document. Unlike some of the more informal graphical methods such as dataflow diagrams, Z is not open to differing interpretations, but instead allows the designer to prove through rigorous mathematical reasoning, the properties of a specification [12]. Z is a well established specification language with many case studies and real world applications appearing in the literature [4], [11]. There are range of automated tools available such as type checking software which allow specifications to be checked for type consistency.

II. The Fuzzy Logic Toolkit

Although formal approaches have been most commonly used in software specification and design, it has been argued that they can and should be applied to all stages of the systems
development lifecycle, including requirements determination [1], [16]. However there is a body of literature suggesting that there are some system problems not naturally understood in precise or exact terms. Instead they may be characterised by imprecision and uncertainty, and any models that we build to represent them need to take this into account (for example see [2], [3]and [17]). Fuzziness or naturally occurring imprecision is one major source of uncertainty in such systems, particularly those that are designed to support human decision making or judgement [6], [18]. If Z is being used in the specification of such systems then the specifier would need to define their own strategy for representing inherently fuzzy concepts.

A fuzzy logic toolkit for Z has been developed to provide a 'standard' approach by providing a mechanism for representing and manipulating fuzzy sets [7], [8]. The toolkit provides a notation that incorporates fuzzy set ideas within the language itself while at the same time retaining the precision of any Z model. It contains definitions for the operators, measures and modifiers necessary for the manipulation of fuzzy sets and relations. It also contains a set of laws that describe the behaviour of the toolkit definitions including ones that establish an isomorphism between the toolkit definitions and the corresponding definitions in the default Z mathematical toolkit when applied to crisp or boolean sets. A series of examples have been generated to illustrate how the toolkit can be used to model problem domains where it is more natural to think of the extent or the degree to which something occurs, or where it is more natural to describe concepts approximately and linguistically rather than in a precise or numeric way [9].

This paper presents a further example, namely a sample specification for a simple fuzzy expert system. Unlike those presented previously [9], which focussed on the toolkit definitions for fuzzy measures such as the degree of equality between fuzzy sets or relations, this example illustrates how the toolkit can be used to specify a fuzzy rule base and provide the operators necessary for fuzzy inferencing. The paper is organised as follows: Section III briefly introduces the example. The sample specification is developed and discussed in section IV. Section V discusses some issues concerned with defuzzification of the output set. The paper concludes with an appendix containing the relevant toolkit definitions used in the specification.

III. The Example

The example is a stock market advisory system based on one presented in [15]. It is a typical example of a simple two input, single output fuzzy model. The system provides advice on whether to buy or sell stock based on the current share price and trading volume. The specification that follows focusses on the fuzzy rule base and the fuzzy sets defined on the input and output variables. It should be noted that the knowledge contained in the rule base together with the parameters that define the fuzzy sets used by these rules could be acquired from a domain expert or perhaps be learnt from a set of previous decisions or examples. The focus here is on the use of the fuzzy logic toolkit in the specification of systems of this type rather than on issues concerned with knowledge acquisition and elicitation.

It is assumed that reader has a rudimentary knowledge of the Z language and notation. The reader is directed to the reference list (in particular [5], [11] and [13]) and the Z archive (accessed from www.afm.sbu.ac.uk/z/) for more detail if required. The specification that follows has been type-checked using the type checking software, ZTC2.

IV. The Specification

The concept of a type is fundamental to Z. Each expression in a specification is associated with a unique type. Basic types (or given sets) are declared explicitly at the beginning of the specification. For the stock market advisory system two basic types, share price and trading volume can be declared as follows:

\[
\text{[sharePrice, tradingVolume]}
\]

Later in the specification fuzzy sets will be defined on each and a system operation can be provided to allow these sets to be constructed.

The output variable, decision can be defined as a subset of the set of integers. It represents the number of shares to be bought or sold. The set of integers, \( \mathbb{Z} \), is a built-in type within the language so there is no need to declare it explicitly. A possible definition for decision is

\[
\text{decision} : \mathbb{Z}
\]

The two input variables (of type sharePrice and tradingVolume respectively) and the output variable (of type decision) are treated as linguistic variables whose values will be modelled as fuzzy sets. For simplicity there are three values for each variable. Those for the output variable will remain constant throughout the specification and can be defined as fuzzy sets on the reference set decision as follows:

\[
\text{buy} : \mathcal{F}_{\text{decision}} \\
\text{hold} : \mathcal{F}_{\text{decision}} \\
\text{sell} : \mathcal{F}_{\text{decision}}
\]

\(^2\)ZTC: A Type Checker for Z Notation, Version 2.01, May 1995 (Xiaoping Jia, Division of Software Engineering, School of Computer Science, Telecommunication, and Information Sciences, DePaul University, Chicago, Illinois, USA).
The generic symbol $F$ is defined in the toolkit and models a fuzzy set as a function from some reference set to the real number interval [0,1] (see appendix). The parameters for these could be defined as part of the axiomatic definition (see figure 1 for some sample set parameters). It should be noted that these sets will be discrete due to the discrete nature of the reference set decision and that for a defuzzification technique to be applied the final output set should be a finite fuzzy set.

![Fuzzy sets for decision](image)

**Fig. 1.** Possible fuzzy set parameters for the outputs in the stock market advisory system.

The schema ruleBase defines the fuzzy rule base for the system.

```
ruleBase

increasingPrice : F sharePrice
stablePrice : F sharePrice
decreasingPrice : F sharePrice
lightVolume : F tradingVolume
moderateVolume : F tradingVolume
heavyVolume : F tradingVolume
buyShares : sharePrice x tradingVolume
sellShares : sharePrice x tradingVolume
holdShares : sharePrice x tradingVolume

buyShares = ((stablePrice x lightVolume) 
                or (increasingPrice x moderateVolume)) 
                or (increasingPrice x heavyVolume)
holdShares = ((stablePrice x heavyVolume) 
                or (stablePrice x moderateVolume)) 
                or ((increasingPrice x lightVolume) 
                or (decreasingPrice x lightVolume))
sellShares = (decreasingPrice x moderateVolume) 
                or (decreasingPrice x heavyVolume)
```

In this example there are nine (9) fuzzy rules. Three of these lead to a decision to buy shares, four to a decision to hold shares and two lead to a decision to sell shares. The fuzzy relations buyShares, holdShares and sellShares (of type $F(sharePrice \times tradingVolume)$) are formed by the fuzzy set union of a series of fuzzy cartesian products between the fuzzy sets defined on the basic types, sharePrice and tradingVolume. Each fuzzy cartesian product corresponds to a fuzzy rule antecedent in the rule base. A generic symbol for a fuzzy relation $F$, the fuzzy cartesian product ($\times$) and the fuzzy union (or) operators are all defined in the toolkit (see appendix). The rule base can be read as follows:

- IF share price is stable and trading volume is light
  THEN decision is buy
- IF share price is increasing and trading volume is moderate
  THEN decision is buy
- IF share price is increasing and trading volume is heavy
  THEN decision is buy
- IF share price is stable and trading volume is heavy
  THEN decision is hold

etc

In the initial state all fuzzy sets and fuzzy relations, declared in the schema ruleBase are empty. The initial system is

```
INIT
ruleBase

increasingPrice = empty
stablePrice = empty
decreasingPrice = empty
lightVolume = empty
moderateVolume = empty
heavyVolume = empty
```

From the toolkit definitions and laws it can be shown that in the initial state the following holds.

- buyShares = empty
- holdShares = empty
- sellShares = empty

The specification now provides operational schema that allow the fuzzy sets defined on the basic types sharePrice and tradingVolume to be built. For example the schema build SharePrice takes three inputs msp, mjp, mdp of type $M$ and one, sp, of type sharePrice to allow the membership values of the three fuzzy sets for each reference set element to be defined. A similar schema, buildTradingVolume could be written to allow the three fuzzy sets lightVolume, moderateVolume and heavyVolume to be built.

---

3 In $Z$ the character $\uparrow$ is used to denote an input observation. The decoration $'$ is used to denote the value after some operation has been carried out.
The schema \( \Delta \text{ruleBase} \) is included in \( \text{buildSharePrice} \). It is defined as

\[
\Delta \text{ruleBase} \\
\text{ruleBase} \\
\text{ruleBase}'
\]

The operational schema, \( \text{advise} \), takes a share price and a trading volume as an input and builds the output fuzzy set, \( \text{decisionSet} \).

\[
\text{advise} \\
\Xi \text{ruleBase} \\
\sp ? : \text{sharePrice} \\
\tv ? : \text{tradingVolume} \\
\text{decisionSet} : \mathcal{F} \text{decision}
\]

\[
\text{decisionSet} = (\text{buyShares}(\sp ? \rightarrow \tv ?) \supseteq \text{buy}) \\
\text{or} (\text{sellShares}(\sp ? \rightarrow \tv ?) \supseteq \text{sell}) \\
\text{or} (\text{holdShares}(\sp ? \rightarrow \tv ?) \supseteq \text{hold})
\]

The schema uses the fuzzy set truncation operator, \( \lfloor \rfloor \), to truncate each of the output fuzzy sets and forms the decision set by evaluating the fuzzy set union of each of the truncated sets. The toolkit also contains a generic definition for a fuzzy set scaling operator \( \langle \cdot \rangle \) which preserves the shape of the fuzzy set. Truncation is being used only as an example. The schema \( \Xi \text{ruleBase} \) is included in \( \text{advise} \) and is defined as follows

\[
\Xi \text{ruleBase} \\
\text{ruleBase} \\
\text{ruleBase}'
\]

\[
\text{increasingPrice} = \text{increasingPrice}' \\
\text{stablePrice} = \text{stablePrice}' \\
\text{decreasingPrice} = \text{decreasingPrice}' \\
\text{lightVolume} = \text{lightVolume}' \\
\text{moderateVolume} = \text{moderateVolume}' \\
\text{heavyVolume} = \text{heavyVolume}'
\]

When \( \text{advise} \) is applied in the initial state it can be shown that \( \text{decisionSet} \) is empty i.e.

\[
\text{buyShares}(\sp ? \rightarrow \tv ?) = 0 \land \\
\text{sellShares}(\sp ? \rightarrow \tv ?) = 0 \land \\
\text{holdShares}(\sp ? \rightarrow \tv ?) = 0 \\
\Rightarrow \\
\text{buyShares}(\sp ? \rightarrow \tv ?) \supseteq \text{buy} = \text{empty} \land \\
\text{sellShares}(\sp ? \rightarrow \tv ?) \supseteq \text{buy} = \text{empty} \land \\
\text{holdShares}(\sp ? \rightarrow \tv ?) \supseteq \text{buy} = \text{empty} \\
[\forall \text{fun} : \mathcal{T} \rightarrow \mathcal{M} \bullet 0 \supseteq \text{fun} = \text{empty} ] \\
\Rightarrow \text{decisionSet} = \text{empty} \\
[ \text{empty or (empty or empty)} = \text{empty} ]
\]

It should be noted that the toolkit defines the basic operations on fuzzy sets and relations in terms of the T-norm and T-conorm operators, \( \min \) and \( \max \). If the specifier wishes to use alternative definitions such as algebraic product and algebraic sum, then the relevant toolkit definitions would need to be redefined in the preamble of the specification and the proofs of all relevant toolkit laws re-checked.

V. Defuzzification

To complete the specification and to provide advice on the number of shares to be traded, the output set \( \text{decisionSet} \) should be defuzzified. Currently the toolkit does not contain any definitions for a defuzzification function so it would need to be defined within the specification itself. In a practical sense it only has meaning for fuzzy sets defined on a closed interval finite numeric reference set. To ensure that the defuzzification function will applied to a finite fuzzy set the reference set, \( \text{decision} \) should be constrained to a closed interval of integers. For example the definition given earlier could be replaced by

\[
\text{decision} == \{ z : \mathbb{Z} \ | -300 \leq z \leq 300 \}
\]

The following would appear in the preamble of the specification (before introducing the state and operational schemas) and defines a defuzzification function, \( \text{centroid} \). The functions \( \text{wcounter} \) and \( \text{wcount} \) sum the product of each reference set element and membership value pair. They are only defined for finite fuzzy sets on the reference set \( \mathcal{R} \). The function \( \text{centroid} \) evaluates the weighted mean. For an empty fuzzy set, the weighted mean is zero.

\[
\text{wcounter} : (\mathcal{R} \rightarrow \mathcal{M}) \rightarrow \mathcal{R} \\
\forall \text{fset} : (\mathcal{R} \rightarrow \mathcal{M}) \bullet (\forall r : \text{dom fset} \bullet \text{wcounter (fset)} = \langle (\text{fset}(r) \times r) + \text{wcounter} (\{r \in \text{fset}\}) \rangle \land \text{wcounter} \emptyset = 0
\]

\[
\text{wcount} : \text{finite } \mathcal{R} \rightarrow \mathcal{R} \\
\forall \text{fset} : \text{finite } \mathcal{R} \bullet \text{wcount (fset)} = \text{wcounter} (\mathcal{E S}(\text{fset}))
\]
\[ \text{centroid} : (\text{finite } \mathcal{F} \mathcal{T}) \to \mathbb{R} \]
\[ \forall \text{fset} : \text{finite } \mathcal{F} \mathcal{T} \cdot \]
\[ (\text{fset} = \text{empty}) \Rightarrow (\text{centroid}(\text{fset}) = 0) \land \]
\[ (\text{fset} \neq \text{empty}) \Rightarrow \]
\[ (\text{centroid}(\text{fset}) = \text{wcount}(\text{fset})/\text{count}(\text{fset})) \]

Finally the schema, advise, can be modified to include the de-

fuzzification of the decision set. The schema now contains

an output observation representing the number of shares to be

traded. The function div is used to return the integer part of

centroid(decisionSet) and is defined for the set of real num-

bers in [14].

\[
\text{advise} \]
\[
\exists \text{ruleBase}
\]
\[
\text{sp? : sharePrice}
\]
\[
\text{to? : tradingVolume}
\]
\[
\text{ansl : decision}
\]
\[
\text{decisionSet} : \mathcal{F} \text{decision}
\]
\[
\text{decisionSet} = (\text{buyShares}(\text{sp?} \mapsto \text{to?}) \supset \text{buy})
\]
\[
or (\text{sellShares}(\text{sp?} \mapsto \text{to?}) \supset \text{sell})
\]
\[
or (\text{holdShares}(\text{sp?} \mapsto \text{to?}) \supset \text{hold})
\]
\[
\text{ansl} = (\text{centroid}((\text{decisionSet})) \div 1
\]

When advise is applied in the initial state the advice will be
to trade no shares i.e.

\[ (\text{centroid}(\text{empty}) = 0) \land (0 \div 1 = 0) \Rightarrow \text{ansl} = 0 \]

VI. Conclusion

The example presented here illustrates the use of the fuzzy
logic toolkit in the specification of simple fuzzy expert
system. The fuzzy rule base is modelled as a series of fuzzy
relations formed between the two basic types introduced at
the beginning of the specification. The toolkit definitions for
fuzzy set union and the fuzzy cartesian product operators
are used to build these relations. An initial state, where
all fuzzy sets and relations in the rule base schema are
empty, is defined. Operational schema to build the rule base
fuzzy sets and the final decision set are also defined. It is
shown that these schema can be applied to the initial state
and that when the decision set is a finite fuzzy set, a de-
fuzzification function can be applied to deliver a scalar output.

Appendix: Relevant Toolkit Definitions

It assumed that the set of real numbers, \( \mathbb{R} \), is known and that
all arithmetic operators and relations are defined for \( \mathbb{R} \). It is
also assumed that min and max have been defined for the set
of real numbers [10], [14].

\[ \mathcal{M} = = \{ r : \mathbb{R} \mid 0 \leq r \leq 1 \} \]

\[ \mathcal{F} \mathcal{T} = T \to \mathcal{M} \]

\[ \text{finite } \mathcal{F} \mathcal{T} = \{ F : \mathcal{F} \mathcal{T} \mid F \supset \{ 0 \} \in \mathcal{F}(T \times \mathcal{M}) \} \]

\[ X \triangleright Y = = \mathcal{F}(X \times Y) \]

\[ \mathcal{E} \mathcal{S} : (\mathcal{F} \mathcal{T}) \to (\mathcal{F} \mathcal{T}) \]

\[ \forall \text{fun} : \mathcal{F} \mathcal{T} \cdot \mathcal{E} \mathcal{S}(\text{fun}) = \text{fun} \supset \{ 0 \} \]

\[ \text{total} : (\mathcal{F} \mathcal{T}) \to (T \to \mathcal{M}) \]

\[ \forall \text{fun} : (\mathcal{F} \mathcal{T}); \ t : T \cdot \]
\[ t \in \text{dom fun} \Leftrightarrow (\text{total fun})(t) = \text{fun}(t) \]
\[ t \notin \text{dom fun} \Leftrightarrow (\text{total fun})(t) = 0 \]

\[ X \times Y = = (FX \times FY) \to (X \triangleright Y) \]

\[ \forall x : X; \ y : Y; \ zset : FX; \ yset : FY \cdot \]
\[ z \triangleright (zset \times yset)(x \mapsto y) = \min \{(\text{total zset})(x), (\text{total yset})(y)\} \]

\[ \text{fun1}, \text{fun2} : \mathcal{F} \mathcal{T} \cdot \]
\[ \text{dom}((\text{fun1} \cup \text{fun2})) = \text{dom fun1} \cup \text{dom fun2} \land \]
\[ \forall t : \text{dom (fun1 or fun2)} \cdot \]
\[ ((\text{fun1} \lor \text{fun2})(t)) = \max \{(\text{total fun1})(t), (\text{total fun2})(t)\} \]

\[ \text{empty} : \mathcal{F} \mathcal{T} \]

\[ \forall t : T \cdot \text{empty}(t) = 0 \]

\[ \triangleright = : (\mathcal{M} \times \mathcal{F} \mathcal{T}) \to \mathcal{F} \mathcal{T} \]

\[ \forall m : \mathcal{M}; \ \text{fun} : \mathcal{F} \mathcal{T} \cdot \]
\[ (\forall t : \text{dom fun} \cdot (m \triangleright \text{fun})(t) = \min (m, \text{fun}(t))) \]
\[ T \]

counter : \((T \rightarrow \mathcal{M}) \rightarrow \mathbb{R}\)

\[
\forall \text{fun} : (T \rightarrow \mathcal{M}) \bullet
\]

\[
(\text{fun} \neq \emptyset) \Rightarrow (\forall t : \text{dom fun} \bullet
\]

\[
\text{counter} (\text{fun}) = \text{fun}(t) + \text{counter} (\{t\} \ast \text{fun}) \land
\]

\[
(\text{fun} = \emptyset) \Rightarrow (\text{counter} (\text{fun}) = 0)
\]

\[ T \]

count : \((\text{finite } T) \rightarrow \mathbb{R}\)

\[
\forall \text{fun} : \text{finite } T \bullet
\]

\[
\text{count} (\text{fun}) = \text{counter} (\mathcal{E}S (\text{fun}))
\]

References


