Chapter 7
Fuzzy Logic

7.1 Introduction

This chapter is probably the densest one in this book and at the same time the more demanding from the reader. It deals with logic and then with a superset of it, fuzzy logic, so it covers a lot of material that will need to be digested with calm. Even so, and for reasons of space, it only scratches the surface of this branch of mathematics. It must be taken into account that a deep revision or a complete introduction to logic usually needs an entire book, so we have opted to show a concise and clear set of concepts that we hope will satisfy the audience. For example, for fuzzy implication we expose the Lukasiewicz operator as the most classic one in fuzzy logic (Kundu and Chen 1994), but the reader should be aware that there is much more fuzzy logic theory out there that can be easily found in theoretical books.

The material already covered in Chaps. 5 and 6 allow us to start the chapter with a brief introduction to propositional logic that quickly leads us to fuzzy logic, paying attention to the logical operators in fuzzy compound propositions for conjunction, disjunction, negation and implication. Later, fuzzy hedges are exposed, exploring at the same time some of their most interesting properties. Then, the fuzzy compound expression of the type "if x is *A* and y is *B* then z is *C*" paves the way to Fuzzy Rule Based Systems, FRBS.

Aside describing the concept of knowledge database and the classic architecture that makes up a FRBS, we discuss some defuzzification methods, discovering Singletons at the same time as a useful tool for creating practical fuzzy logic based applications. Along the chapter new and powerful FuzzyLisp functions will allow us to develop some real fuzzy logic models at the end of the chapter.

L. Argüelles Méndez, *A Practical Introduction to Fuzzy Logic using LISP*,
Studies in Fuzziness and Soft Computing 327,
DOI 10.1007/978-3-319-23186-0_7

7.2 The Beginning of Logic

More than 2000 years ago, in a European region bathed by the east Mediterranean waters a country, or better said, a civilization, was flourishing. Greece not only consolidated the study of mathematics and astronomy, but brought to history the development of democracy and then other astonishing consequences of human reasoning such as philosophy and within it, the study of logic. Although the controversy about whether logic is a part of philosophy or merely an instrument for progressing in philosophy continued for centuries, the truth is that even today logic is taught in the universities as a curriculum for philosophy studies. Maybe surprisingly, the study of philosophy and then logic is the reason behind the interest of many actual philosophers on fuzzy logic.

As it is well known, the most important philosopher from antique Greece was Aristotle (384BC–322BC). Maybe the most influential philosopher ever, Aristotle was the founder of logic, a structured pattern of thinking that makes it possible to arrive to new knowledge from previous established information. Central to Aristotle's Logic is his theory of syllogisms, from the Greek term *sullogismos* (deduction). In his work titled "Prior Analytics", Aristotle describes what a syllogism is: "A syllogism is speech (*logos*) in which, certain things having been supposed, something different from those supposed results of necessity because of their being so". In this sentence, the terms "things supposed" mean what we call a premise, while the terms "results of necessity" are what we understand as a conclusion. Translating it into English, we can say that a syllogism is a logic construction where after some premises are given we can reach a conclusion, or a logic construction that transform some given premises into a conclusion. As an example, the following syllogism is one of the most famous ones:

- all men are mortal
- all Athenians are men

$\rightarrow$ all Athenians are mortal

In this syllogism we have two premises: "all men are mortal" and "all Athenians are men". The syllogism produces a conclusion: "all Athenians are mortal". As homage to syllogisms, we can express these premises in Lisp simply typing the following at the Lisp prompt: *(setq c1 '(men mortal)), (setq c2 '(athenians men)).* For obtaining the conclusion we can write a simple Lisp function for dealing with this naïve type of syllogisms. The function, named *(syllogism)* is shown in Code 7-1:

```
;code 7-1
(define (syllogism c1 c2)
   (if (= (first c1) (last c2))
     (list (first c2) (last c1))
     nil
   )
)
```

For trying this function we only need to type the following at the Lisp prompt:

```
> (syllogism '(men mortal) '(athenians men))
: (athenians mortal)
```

This type of syllogism is only one from the 24 valid types of syllogisms. The reader can find all of them in a good book of Logic and it would not be a hard task to translate them all to Lisp functions. In fact, such a Lisp project should be appealing to students of logic, simply because it would be a good thing for the study of reasoning itself (McFerran 2014; Priest 2001).

7.3 Modern Bivalent Logic

After the brilliant period of philosophy in Greece, no true revolution happened in the history of logic until the beginning of the nineteenth century, when the method of proof used in mathematics was applied to the concepts of logic. Moreover, Aristotelian logic was unable to satisfy mathematical creations at all because many arguments used for developing logical mathematical constructions are based on formulations of the type "if… then…", and these formulations were absent in Aristotelian logic. This section of the chapter deals with propositional logic, an important branch of modern formal logic that is a solid foundation in classical texts of artificial intelligence (e.g.: Russell and Norvig 2009).

Propositional logic is interested in logic relationships that are based on the study of propositions, and then in the creation of other propositions by means of logical operators. The main question for propositional logic is thus the following one: What can we do with propositions and with combining them by using logical operators? In this question there are two key words: propositions and logical operators.

A proposition is natural language (Aristotle would name it again *logos*), expressing an assertion that can be either true or false, hence the term "bivalent", but not true and false at the same time". Simple propositions take the following form:

x is P

here, x is called the subject of the proposition, and P is called the predicate of the proposition, usually expressing a property or feature of x. The following are valid propositions:

Tigers are mammals
The Montblanc is the highest mountain in the Alps
France is a European country
The "Jupiter" Symphony was written by Beethoven

The three first propositions are true, while the last one is false (Mozart would hate us if stated otherwise!).

When x is any subject belonging to an universe of discourse X, then a predicate P converts itself automatically into a function defined on X, forming a different proposition for every subject x. We shall represent by P(x) to this type of functions, named propositional functions. From different values x_1, x_2, …, x_n we can obtain different propositions $p(x_1)$, $p(x_2)$, …, $p(x_n)$. Any of them can be also expressed by:

$$x_i \text{ is } P$$

and, as already stated, anyone of them can be true or false until the variable itself is instanced. As an example, let X be the universe of discourse of some European countries X = {France, Spain, Germany} and P a predicate meaning "x is a Country bathed by the Mediterranean Sea". Then, the propositional function P(x) produces the following propositions:

$p(x_1)$ = "France is a Country bathed by the Mediterranean Sea"
$p(x_2)$ = "Spain is a Country bathed by the Mediterranean Sea"
$p(x_3)$ = "Germany is a Country bathed by the Mediterranean Sea"

Propositions $p(x_1)$ and $p(x_2)$ have a truth value equal to true, while proposition $p(x_3)$ has a truth value equal to false. Note again that the propositional function P(x) = "x is a Country bathed by the Mediterranean Sea" on its own is neither false nor true, and only when we substitute x by a member from the Universe of discourse the resulting proposition becomes true or false.

Things become even more interesting when two or more simple propositions are combined, forming what is known as a compound proposition. In this case, the truth-value of a compound proposition results from the truth-values of every simple proposition and the way they are connected, that is, from the type of logical connective, or logical operator, used for establishing the link between simple propositions. The main logical operators are the following ones:

Conjunction: Given two propositions p, q, we call their conjunction to the compound proposition "p and q", denoting them by the logical expression "$p \wedge q$". This compound proposition is true if and only if both p and q propositions are true. As an example, the following compound proposition is true: "Japan is a country and Kyoto is a Japanese city". The following one is false: "The sun is the brighter sky object in the solar system and it revolves around Earth", as for example, Ptolemy affirmed.

Disjunction: Given two prepositions p, q, we call their disjunction to the compound proposition "p or q", denoting them by the logical expression "$p \vee q$". This compound proposition is true when at least one of both propositions, p or q, is true, or, obviously, when both are true. For being false it needs that both propositions are false. An example of true compound proposition linked by a disjunction is: "four is bigger than two or eight is a prime number" (it does not matter than eight is not a prime number, it suffices with the true proposition "four is bigger than two". On the other hand: "Pi is an integer or a circle has two centres" is a false compound proposition.

Negation: From any proposition p we shall call "negation of p" to the proposition "not p", denoting it by the expression "¬p". The proposition ¬p will be true when p is false and will be false when p is true. As an example, after stating the proposition p = "the Sun is a star", then ¬p, that is, "the Sun is not a star", is false.

Implication: Also known as "conditional proposition", it is expressed by the sentence "p implies q", and it is usually denoted by the expression "p → q". When p → q, it means that the conditional proposition is true except when the proposition p is true and q is false. In other words, **if p (also named the "antecedent" in implication) is true, then q (the consequent) must be also true for satisfying p → q being true**. As an example, the implication "if I breath then I produce CO_2" is true. At the other extreme, please note that we have lots of expressions in common language where both p and q are false, for example: "if you are the Queen of Saba then I am Solomon". These types of implications, where both the antecedent and consequent are false, are always true, because the logical structure of the conditional is satisfied.

If we define the truth value of a proposition by means of using the number 1 for expressing true and using 0 for expressing false, we can express formally these logical operators in the following way:

- Conjunction: $p \wedge q = \min(p,q)$ (7-1)
- Disjunction: $p \vee q = \max(p,q)$ (7-2)
- Negation: $\neg p = 1 - p$ (7-3)
- Implication: $p \rightarrow q = \min(1, 1 + q - p)$ (7-4)

And, hereafter, we can construct a truth table, an arranged table of values that enumerates all the possible truth-values combinations of p_i propositions. Table 7.1 shows the truth table for conjunction, disjunction, negation and implication from two propositions p, q. In this table we represent false by the number 0 and true by the number 1.

Translating Table 7.1 into Lisp is easy because Lisp (and all traditional programming languages) use standard bivalent logic, including the functionality of "and", "or" and "not", as we have learnt in Part I of this book. For completing the architecture of Table 7.1 we only need to create a simple function named *(implication)*, as shown in Code 7-2:

Table 7.1 Truth table for conjunction, disjunction, negation and implication

p	q	p ∧ q	p ∨ q	¬p	¬q	p → q
1	1	1	1	0	0	1
1	0	0	1	0	1	0
0	1	0	1	1	0	1
0	0	0	0	1	1	1

False = 0, True = 1

Table 7.2 A translation of Table 7.1 into Lisp

p	*q*	*(and p q)*	*(or p q)*	*(not p)*	*(not q)*	*(implication p q)*
(setq p true)	*(setq q true)*	*True*	*True*	*Nil*	*Nil*	*True*
(setq p true)	*(setq q nil)*	*Nil*	*True*	*Nil*	*True*	*Nil*
(setq p nil)	*(setq q true)*	*Nil*	*True*	*True*	*Nil*	*True*
(setq p nil)	*(setq q nil)*	*Nil*	*Nil*	*True*	*True*	*True*

```
;code 7-2
 (define (implication p q)
    (if (and p (not q))
          nil
          true
    )
)
```

Typing the following Lisp expressions row by row and from left to right at the Lisp prompt we obtain the complete contents of Table 7.2.

Now it is time to make an important assertion. Maybe one of the most important ones in this chapter. A sentence that will help the reader to give a quantum leap towards the understanding of fuzzy logic:

There is an isomorphism between the logical operators "and", "or", "not", "implication" and the set operations "intersection", "union", "complement" and "inclusion".

In practical terms this means that all the material exposed in Sect. 5.2 about the classic theory of sets is of entire application in the field of logic after the simple substitution of the concept of membership or not membership of an element *x* to a set A and the concept of true or false when referring to a proposition p.

7.4 Fuzzy Logic

Let us now consider the proposition p = "John is old" when John is 55 years old. Under a traditional point of view in logic, in order to know if the proposition p is true or false, we need to define the predicate of p, that is, a definition of an old person. Arbitrarily, let us assume that an old person is a person older than 50 years old. Taking into account this definition, then proposition p is true. If we reflect about this, we immediately realize the isomorphism enunciated some lines above: When we say "John is old", we implicitly declare that John belongs to the set of old persons, defined for example as the set of all persons older than 50 years old. That is, since John belongs to the set of old persons, then proposition p is true.

However, and as we discussed extensively in Chap. 5, in fuzzy sets theory the membership of an element *x* to a fuzzy set *S* is a question of degree, expressed in the closed interval [0,1]. In fact, the isomorphism continues between the membership degree of an element x to a fuzzy set A and the truth degree of a predicate and then of a proposition P, thus conceptually filling the gap between fuzzy sets and fuzzy logic (Klir and Yuan 1995). This line of reasoning gives birth to the concept of fuzzy proposition whose definition results from introducing some important touches in the definition of a traditional proposition:

A fuzzy proposition is natural language declaring an assertion that has implicitly associated a truth-value expressed by a real number in the closed interval [0,1]. Simple fuzzy propositions have the following structure:

$$x \,\text{is}\, P$$

The reader should note that the subject x of a fuzzy proposition P is usually not fuzzy. What distinguishes a classic proposition from a fuzzy proposition is the characterization of the predicate P. The fuzziness of P is what generates fuzzy propositions. Hereafter we shall refer to fuzzy propositions writing them in italics: e.g.: *p, q, r*.

These are sound paragraphs, but these ideas are what really allow us to transit from classical logic to fuzzy logic. Fortunately we already have learnt lots of concepts, definitions, and Lisp constructions on fuzzy sets, and all this material is of immediate application in fuzzy logic. We can even say that we started, without saying it so, to speak about fuzzy logic from Sect. 5.3 of this book.

In order to evaluate a simple fuzzy proposition p, we need to know the definition of its predicate, which will be generally given by the definition of a fuzzy set. Let, for example P, be a fuzzy set representing "old persons" expressed in FuzzyLisp by the construction *(setq P '(old 50 90 90 90))*. For calculating the truth-value of the proposition p = "John is old" when John is 55 years old, we can simply type at the Lisp prompt:

```
> (last (fl-set-membership? P 55))
: 0.125
```

So the resulting truth-value of proposition p is 0.125.

In this moment, and more from an aesthetic or even also semantic point of view than a real disadvantage, you can argue that the name of the function *(fl-set-membership?)* does not seem appropriate for evaluating fuzzy propositions. Thanks to Lisp this is no problem. We can type the following expression at the Lisp prompt:

```
> (setq fl-truth-value? fl-set-membership?)
: (lambda ....) (output omitted)
```

and now we can write:

```
> (last (fl-truth-value? P 55))
: 0.125
```

what we have made is to create an alias for the FuzzyLisp function *(fl-set-membership?)*. If you wish so, you can create new aliases from all the existing FuzzyLisp functions for your own practical applications. The rest of this book will not use aliases. By the way, I would like to seize the opportunity to say that if English is not your mother tongue you can translate all the FuzzyLisp functions to your language. If you are, for example, a German reader, you could rename *(fl-set-membership?)* to:

> (setq fl-Wahrheitswert? fl-set-membership?)
: (lambda ....) (output omitted)

and then again:

> (last (fl-Wahrheitswert? P 55))
: 0.125

After translating all the FuzzyLisp functions, you could put all these definitions into a single file named, for example, fl-deutsch.lsp and then, for creating your fuzzy models you would only need to include the following two lines of code at the beginning of your application:

```
(load "fuzzylisp.lsp")
(load "fl-deutsch.lsp")
```

And hereafter you can call all the functions in your own language. By the way, all the original names of the FuzzyLisp functions as described in Appendix II will be always available at your fingertips despite the translation.

7.5 Logical Connectives in Fuzzy Propositions

In the same way we have defined logical connectives in order to form compound propositions in classical logic, we can now extend these ideas to compound fuzzy propositions. Let us review them:

The Conjunction of two fuzzy propositions p, q, represented by $p \wedge q$, is the result of the minimum truth-value, Tv, of both p and q. Expressed formally:

$$Tv(p \wedge q) = min(Tv(p), Tv(q)) = min(\mu_A(x), \mu_B(y)) \qquad (7\text{-}5)$$

where A and B are fuzzy sets representing the predicates associated to the fuzzy propositions p and q, respectively, while x and y are feature values associated to their respective subjects. As an example, we can have the following two fuzzy propositions: p = "John is old" and q = "Eva is young". As in the previous example, let us define a fuzzy set P for representing the concept of old persons by means of the Lisp expression (setq P '(old 50 90 90 90)) and then a fuzzy set Q representing the concept of a young person by means of the Lisp expression (setq Q '(young 0 0

15 30)). If John is 55 years old, that is, x = 55 and Eva is 18 years old, that is, y = 18, we shall have:

(last (fl-set-membership? P 55)) → *0.125*
(last (fl-set-membership? Q 18)) → *0.8*

That is, the fuzzy proposition *p* has a truth-value 0.125 and the fuzzy proposition *q* has a truth-value 0.8. Now using expression (7-5) we arrive to the conclusion that the compound fuzzy proposition "John is old and Eva is young" has a truth-value equal to 0.125.

Writing a NewLisp function for obtaining the truth value of a compound fuzzy proposition with a logical connective "and" is straightforward, as shown in Code 7-3:

```
;code 7-3
(define (fl-truth-value-p-and-q? P Q x y, a b)
   (setq a (last (fl-set-membership? P x)))
   (setq b (last (fl-set-membership? Q y)))
   (min a b)
)
```

Trying the function confirms our previous result:

> (fl-truth-value-p-and-q? P Q 55 18)
: 0.125

In a similar way, we define the Disjunction of two fuzzy propositions *p*, *q*, represented by *p* ∨ *q*, as the result of the maximum truth-value of both *p* and *q*. Expressed formally:

$$Tv(p \vee q) = max(Tv(p), Tv(q)) = max(\mu_A(x), \mu_B(y)) \tag{7-6}$$

where again *A* and *B* are fuzzy sets representing the predicates associated to the fuzzy propositions *p* and *q*, respectively, while *x* and *y* are feature values associated to their respective subjects. Using the same example from some lines above, the fuzzy proposition "John is old or Eva is young" will have a truth value *Tv(p* ∨ *q) = 0.8*. Let us prove it by creating the function *(fl-truth-value-p-or-q)*, as shown in Code 7-4:

```
; Code 7-4
(define (fl-truth-value-p-or-q? P Q x y, a b)
(setq a (last (fl-set-membership? P x)))
(setq b (last (fl-set-membership? Q y)))
(max a b)
)
```

Testing the function with the same supplied data *P*: *(old 50 90 90 90)*, *Q*: *(young 0 0 15 30))*, x = 55 years old and y = 18 years old, we obtain:

> (fl-truth-value-p-or-q? P Q 55 18)
: 0.8

The Negation of a fuzzy proposition p, denoted by $\neg p$, is the result of subtracting the truth-value of p from one. Formally:

$$Tv(\neg p) = 1 - Tv(p) = 1 - (\mu_A(x)) \qquad (7\text{-}7)$$

where A is the fuzzy set representing the predicate associated to the fuzzy proposition p and x is the feature value associated to its subject. From all the logical connectives used on fuzzy propositions, this is the easier one to implement in Lisp, as shown in Code 7-5:

```
;Code 7-5
;returns the truth value of the negation of fuzzy
proposition P
(define (fl-truth-value-negation-p? P x)
   (sub 1.0 (last (fl-set-membership? P x)))
)
```

For testing the function, we shall calculate the truth-value of the following propositions: "John is not old", "Eva is not young", using the same definitions of fuzzy sets P and Q as above. For John we shall have x = 55 and for Eva we shall have x = 18 years old. Then

> (fl-truth-value-negation-p? P 55)
: 0.875

> (fl-truth-value-negation-p? Q 18)
: 0.2

As expected, the expression that returns the truth-value of a fuzzy conditional proposition or fuzzy implication $p \rightarrow q$ is the most complex one from the four logical connectives from this section:

$$\mathrm{Tv}(p \rightarrow q) = min(1{,}1 + Tv(q) - Tv(p)) = min(1{,}1 + \mu_B(y) - \mu_A(x)) \qquad (7\text{-}8)$$

As usually, A and B are fuzzy sets representing the predicates associated to fuzzy propositions p and q, respectively, while x and y are the feature values associated to their respective subjects. Code 7-6 shows the code for calculating fuzzy conditional propositions:

```
;Code 7-6
(define (fl-truth-value-fuzzy-implication-p-q? P Q x y, a b)
   (setq a (last (fl-set-membership? P x)))
   (setq b (last (fl-set-membership? Q y)))
   (min 1.0 (sub (add 1.0 b) a))
)
```

Now, let us suppose Laura (10) is a daughter of John (55). For calculating the truth value of the fuzzy conditional proposition "if John is old then Laura is young", we only need to type:

```
> (fl-truth-value-fuzzy-implication-p-q? P Q 55 10)
: 1
```

The only way to get a zero truth value from a fuzzy implication is when the truth value of the fuzzy proposition *p* is exactly 1.0 and the truth value of the fuzzy proposition *q* is exactly 0.0, as it happens with classical implication. In our example, given the definitions of fuzzy sets *P* and *Q*, then for any pair x, y included into the supports of their respective fuzzy sets this only happens if x = 90 years old (John) and y = 30 years old (Laura). Expressed into words: "if John is old then Laura is young" means, for these values x, y that the proposition "John is old" has a truth value equal to 1.0, but Laura, from the definition of "being young" in the fuzzy set Q grows and grows until she is not anymore a young person, hence the proposition "Laura is young" ultimately reaches a truth value equal to 0.0 resulting in a zero truth value for the fuzzy implication $p \rightarrow q$. Testing it:

```
> (fl-truth-value-fuzzy-implication-p-q? P Q 90 30)
: 0
```

Table 7.3 shows several truth values as a result of $p \rightarrow q$ for the fuzzy conditional implication "if John is old then Laura is young" when different values x, y are used as the feature values associated to their respective subjects, John and Laura. In the first row several values x_i are given, while the first column represents several y_i values. The shaded cells in the table tell us that when Laura grows and approaches the right limit of the support of fuzzy set *Q*, then the truth value of the fuzzy implication "if John is old then Laura is young" decreases until becoming null. Once again we must remark that John being 90 and Laura being 30 years old is entirely possible (in this instance, Laura would have been conceived when John was 60), but the fuzzy implication has a null truth value because the definitions of fuzzy sets P and specially Q, where Laura at 30 is not anymore a young person:

It is interesting to note that the conjunction/disjunction of two fuzzy propositions, and also the fuzzy implication, are fuzzy relations. For example, the aforementioned conjunction "John is old and Eva is young" is an instance of the general fuzzy compound proposition "x is *A* and y is *B*" that ultimately produces a truth value (isomorphic to membership degree) in the closed interval [0,1]. Remembering the expression [5-24] for defining a fuzzy relation:

Table 7.3 A truth table for different combinations of values x, y in a fuzzy implication $p \rightarrow q$

$P \rightarrow q$	50	60	70	80	90
0	1	1	1	1	1
10	1	1	1	1	1
20	1	1	1	0.91667	0.66667
30	1	0.75	0.5	0.25	0

$$R = \{((x, y), \mu_R(x, y)) \,|(x, y) \in AxB, \mu_R(x, y) \in [0, 1]\}$$

We can see that the general compound fuzzy proposition “x is *A* and y is *B*” is a mapping from the pair x,y to the closed interval [0,1] where *A* and *B* are the fuzzy sets representing the predicates associated to the individual fuzzy propositions “x is *A*”, “y is *B*”, respectively. In the case of the logical connective conjunction, the mapping is given by the expression:

$$\mu_R(x,y) = \min(\mu_A(x), \mu_B(y))$$

The same line of reasoning is valid for disjunction and fuzzy implication in fuzzy compound propositions.

Another important consideration in fuzzy propositions arises when we ask ourselves about the nature of the fuzzy sets associated to them. Do fuzzy propositions need associated fuzzy sets with a continuous characteristic function? Absolutely not. In fact we can use also fuzzy sets with discrete characteristic functions, so we can use both FuzzyLisp Standard or Discrete Set Representations, FLSSR, FLDSR for building fuzzy propositions. As an example, let us again observe the fuzzy proposition *p* = “John is old”, this time defining a FLDSR for *P*. Let us type at the Lisp prompt:

> (setq P (fl-discretize ‘(old 50 90 90 90) 4))
: (old (50 0) (60 0.25) (70 0.5) (80 0.75) (90 1) (90 1) (90 1) (90 1) (90 1))

Now, for calculating the truth vale of p we only need to type:

> (last (fl-dset-membership? P 55))
: 0.125

The same is also valid for compound fuzzy propositions. In this case we need slight modifications to the functions *(fl-truth-value-p-and-q?)*, *(fl-truth-value-p-or-q?)*, *(fl-truth-value-negation-p?)* and *(fl-truth-value-fuzzy-implication-p-q?)* as shown in Code 7-3b, Code 7-4b, Code 7-5b and Code 7-6b, respectively:

```
; Code 7-3b
(define (fl-dtruth-value-p-and-q? dP dQ x y, a b)
   (setq a (last (fl-dset-membership? dP x)))
   (setq b (last (fl-dset-membership? dQ y)))
   (min a b)
)

; Code 7-4b
(define (fl-dtruth-value-p-or-q? dP dQ x y, a b)
   (setq a (last (fl-dset-membership? dP x)))
   (setq b (last (fl-dset-membership? dQ y)))
   (max a b)
)
```

```
;Code 7-5b
(define (fl-dtruth-value-negation-p? dP x)
   (sub 1.0 (last (fl-dset-membership? dP x)))
)

;Code 7-6b
(define (fl-dtruth-value-fuzzy-implication-p-q? dP dQ x y,
a b)
   (setq a (last (fl-dset-membership? dP x)))
   (setq b (last (fl-dset-membership? dQ y)))
   (min 1.0 (sub (add 1.0 b) a))
)
```

Let us test them quickly after defining a fuzzy set *Q* with a discrete characteristic function for representing "young people":

> (setq Q (fl-discretize '(young 0 0 15 30) 4))
:(young (0 1) (0 1) (0 1) (0 1) (0 1) (3.75 1) (7.5 1) (11.25 1) (15 1) (18.75 0.75) (22.5 0.5) (26.25 0.25) (30 0))

Then, considering again the fuzzy propositions *p* = "John is old" and *q* = "Eva is young" when John is 55 years old and Eva is 18 we have:

- "John is old and Eva is young":
> (fl-dtruth-value-p-and-q? P Q 55 18)
: 0.125

- "John is old or Eva is young":
> (fl-dtruth-value-p-or-q? P Q 55 18)
: 0.8

- "Eva is not young"
> (fl-dtruth-value-negation-p? Q 18)
: 0.2

Finally, for the proposition *q* = "Laura is young", when Laura (ten years old), being a daughter of John, we have:

"if John is old then Laura is young"
> (fl-dtruth-value-fuzzy-implication-p-q? P Q 55 10)
: 1

Another example (Bojadziev and Bojadziev 1999) will help to understand even better the fuzzy implication $p \rightarrow q$, in this case using fuzzy sets with discrete characteristic functions. Let *P* and *Q* be fuzzy sets formed respectively by the following Lisp expressions:

(setq P '(high-score (0 0) (20 0.2) (40 0.5) (60 0.8) (80 0.9) (100 1)))
(setq Q '(good-credit (0 0) (20 0.2) (40 0.4) (60 0.7) (80 1) (100 1)))

Where *P* represents the concept of “high score” applied to a bank customer and *Q* represents a certain type of measure about the quality of a hypothetical loan. We can form the fuzzy propositions *p* = “x is high score” and *q* = “y is good credit” and then the fuzzy propositional function “if x is high score then y is good credit”. When, for example, x = 85 and y = 75 we can type at the Lisp prompt:

```
> (fl-dtruth-value-fuzzy-implication-p-q? P Q 85 75)
: 1
```

As the reader can appreciate, there is no practical difference between using a FLSSR or a FLDSR for evaluating truth-values from fuzzy propositions. When the related fuzzy sets have a triangular or trapezoidal membership function results are the same. When other continuous functions are discretized for representing membership functions the differences are minimal.

7.6 Fuzzy Hedges

Fuzzy hedges are linguistic modifiers applied to fuzzy predicates. This short definition soon reveals its consequences because if we say they are applied to fuzzy predicates it immediately means they are also applied to fuzzy sets related to fuzzy predicates. Then it is easy to follow that they will affect fuzzy propositions and then their truth-values, too. In general, a fuzzy hedge H can be represented by an unary operator on the closed interval [0,1] in such a way that a fuzzy proposition of the type “*x* is *P*” converts in:

$$x \,\text{is}\, \mathrm{H}P$$

If *A* is a fuzzy set associated to a fuzzy predicate *P*, the following expression shows how *A* is modified by H for every element x (subject) in *A*:

$$\mathrm{H}A = \mathrm{H}(\mu_A(x)) \tag{7-9}$$

The most used fuzzy hedges are the linguistic modifiers H_1, “very” and H_2, “fairly”, defined respectively by the following expressions:

$$\mathrm{H}_1 : \mathrm{H}_1(\mu_A(x)) = (\mu_A(x))^2 \tag{7-10}$$

$$\mathrm{H}_2 : \mathrm{H}_2(\mu_A(x)) = (\mu_A(x))^{1/2} \tag{7-11}$$

As an example, if we take the fuzzy proposition *p*: “Joe is old”, with a truth value Tv(*p*) = 0.8 that is, if x = Joe has a 0.8 membership degree to the fuzzy set *A* representing old People, then the fuzzy expression *q*: “Joe is very old” will have a truth value:

$$\mathrm{Tv}(q) = H_1(\mu_A(x)) = (\mu_A(x))^2 = 0.64$$

On the other hand, the fuzzy proposition r: "John is fairly old" will have a truth value: $\mathrm{Tv}(r) = \mathrm{H}_2(\mu_A(x)) = (\mu_A(x))^{1/2} = 0.89$. From the intrinsic nature of the functions $y = x^2$ and $y = x^{1/2}$ it is easy to follow that the linguistic hedge H_1, "very", decreases the membership degree of an element x to a fuzzy set A when $\mu_A(x) \neq 1.0$ and ultimately, the truth value of an associated fuzzy proposition p. Conversely, the linguistic hedge H_2, "fairly", increases the membership degree in fuzzy sets and then in any associated fuzzy proposition when $\mu_A(x) \neq 1.0$. In fuzzy logic theory, we say a linguistic hedge is a Strong modifier when $\mathrm{H}(\mu_A(x)) < \mu_A(x)$ and a Weak modifier when $\mathrm{H}(\mu_A(x)) > \mu_A(x)$. Hence, the fuzzy hedge "very" is a strong modifier, while the fuzzy hedge "fairly" is a weak modifier.

FuzzyLisp has a function named *(fl-dset-hedge)* that applies a fuzzy hedge to a discrete fuzzy set. Again from the very nature of the functions $y = x^2$ and $y = x^{1/2}$ the resulting fuzzy set after applying a linguistic modifier on it can not be generally represented by a triangular or trapezoidal characteristic function, so the function always returns the transformed fuzzy set with a FLDSR. This feature invites to design the function in such a way that the input fuzzy set has also a FuzzyLisp Discrete Set Representation. Code 7-7 shows this function:

```
;Code 7-7
(define (fl-dset-hedge dset hedge, i n list-out sublist)
   (setq list-out (first dset));we conserve the set's name
   (setq i 1 n (length dset))
   (while (< i n)
      (setq sublist (nth i dset))
      (case hedge
        (FAIRLY
           (setq list-out (cons
             (list (first sublist) (sqrt (last sublist)))
                          list-out))
        );end FAIRLY
        (VERY
           (setq list-out (cons
             (list (first sublist) (pow (last sublist)))
                          list-out))
        );end VERY
     );end case
    (++ i)
  );while end
  (reverse list-out)
)
```

(fl-dset-hedge) takes two arguments: The first one is a fuzzy set with a FLDSR and the second one is a Lisp symbol representing the desired fuzzy hedge to apply

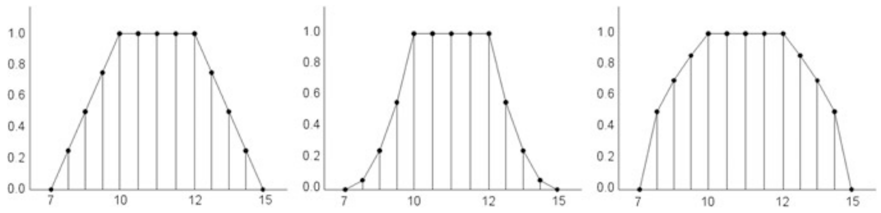

Fig. 7.1 From *left* to *right* a fuzzy set *A* and the fuzzy sets "*VeryA*" and "*FairlyA*"

on it, either FAIRLY or VERY. If our initial fuzzy set has a FLSSR representation we can first make a call to the function *(fl-discretize)* and then use the resulting discretized fuzzy set. Let us see an example of the linguistic modifiers "very" and "fairly" applied to the fuzzy set *(A1 7 10 12 15)*:

> (fl-dset-hedge (fl-discretize '(A1 7 10 12 15) 4) 'VERY)
: (A1 (7 0) (7.75 0.0625) (8.5 0.25) (9.25 0.5625) (10 1) (10.5 1) (11 1) (11.5 1) (12 1) (12.75 0.5625) (13.5 0.25) (14.25 0.0625) (15 0))

> (fl-dset-hedge (fl-discretize '(A1 7 10 12 15) 4) 'FAIRLY)
: (A1 (7 0) (7.75 0.5) (8.5 0.7071067812) (9.25 0.8660254038) (10 1) (10.5 1) (11 1)
(11.5 1) (12 1) (12.75 0.8660254038) (13.5 0.7071067812) (14.25 0.5) (15 0))

it suffices to numerically compare the results from these two functions calls to *(fl-dset-hedge)* to realize the impact of both fuzzy hedges on the fuzzy set defined by the Lisp expression *(A1 7 10 12 15)*, but a graphical representation shows it even better, as can be easily seen in Fig. 7.1.

Needless to say, we can experiment also with fuzzy sets that require from scratch of a discrete characteristic function, as it is the case with a bell shaped function. As we already know, we start defining the function f in Lisp:

> (setq f '(div (add 1.0 (cos (mul 2.0 pi (sub x 2.0)))) 2.0))
: (div (add 1 (cos (mul 2 pi (sub x 2)))) 2)

now we discretize it:

> (setq dBell (fl-discretize-fx 'Bell f 10 1.5 2.5))
: (Bell (1.5 0) (1.6 0.09549150283) (1.7 0.3454915028) (1.8 0.6545084972) (1.9 0.9045084972) (2 1) (2.1 0.9045084972) (2.2 0.6545084972) (2.3 0.3454915028) (2.4 0.09549150283) (2.5 0))

and then we can apply a linguistic modifier on it. In this case, "very":

> (fl-dset-hedge dBell 'VERY)
: (Bell (1.5 0) (1.6 0.009118627113) (1.7 0.1193643785) (1.8 0.4283813729) (1.9 0.8181356215) (2 1) (2.1 0.8181356215) (2.2 0.4283813729) (2.3 0.1193643785) (2.4 0.009118627113) (2.5 0))

Something very interesting happens when we reiteratively apply a fuzzy hedge on a fuzzy set *A*. Let us define, for example, a fuzzy number "close to seven" expressed in Lisp as: *(about-seven 6.5 7.0 7.0 7.5)*:

> (setq A '(about-seven 6.5 7.0 7.0 7.5))
: (about-seven 6.5 7 7 7.5)

First we discretize this fuzzy number defined by a triangular membership function. We use an adequate resolution in order to better appreciate the expected effects of reiteration:

> (setq dA (fl-discretize A 10))
: (about-seven (6.5 0) (6.55 0.1) (6.6 0.2) (6.65 0.3) (6.7 0.4) (6.75 0.5) (6.8 0.6) (6.85 0.7) (6.9 0.8) (6.95 0.9) (7 1) (7.05 0.9) (7.1 0.8) (7.15 0.7) (7.2 0.6) (7.25 0.5) (7.3 0.4) (7.35 0.3) (7.4 0.2) (7.45 0.1) (7.5 0))

applying the fuzzy hedge "Very" we obtain $H_1(\mu_{dA}(x))$:

> (setq VERY-7 (fl-dset-hedge dA 'VERY))
: (about-seven (6.5 0) (6.55 0.01) (6.6 0.04) (6.65 0.09) (6.7 0.16) (6.75 0.25) (6.8 0.36) (6.85 0.49) (6.9 0.64) (6.95 0.81) (7 1) (7.05 0.81) (7.1 0.64) (7.15 0.49) (7.2 0.36) (7.25 0.25) (7.3 0.16) (7.35 0.09) (7.4 0.04) (7.45 0.01) (7.5 0))

now we apply it again, obtaining $H_1(H_1(\mu_{dA}(x)))$:

> (setq VERY-VERY-7 (fl-dset-hedge VERY-7 'VERY))
: (about-seven (6.5 0) (6.55 0.0001) (6.6 0.0016) (6.65 0.0081) (6.7 0.0256) (6.75 0.0625) (6.8 0.1296) (6.85 0.2401) (6.9 0.4096) (6.95 0.6561) (7 1) (7.05 0.6561) (7.1 0.4096) (7.15 0.2401) (7.2 0.1296) (7.25 0.0625) (7.3 0.0256) (7.35 0.0081) (7.4 0.0016) (7.45 0.0001) (7.5 0))

the fuzzy set "*very-very-very-dA*", $H_1(H_1(H_1(\mu_{dA}(x))))$ *is again:*

> (setq VERY-VERY-VERY-7 (fl-dset-hedge VERY-VERY-7 'VERY))
: (about-seven (6.5 0) (6.55 1e-08) (6.6 2.56e-06) (6.65 6.561e-05) (6.7 0.00065536) (6.75 0.00390625) (6.8 0.01679616) (6.85 0.05764801) (6.9 0.16777216) (6.95 0.43046721) (7 1) (7.05 0.43046721) (7.1 0.16777216) (7.15 0.05764801) (7.2 0.01679616) (7.25 0.00390625) (7.3 0.00065536) (7.35 6.561e-05) (7.4 2.56e-06) (7.45 1e-08) (7.5 0))

and after only four iterations, we obtain $H_1(H_1(H_1(H_1(\mu_{dA}(x)))))$, that is, the fuzzy set *"very-very-very-very-dA":*

> (setq VERY-VERY-VERY-VERY-7 (fl-dset-hedge VERY-VERY-VERY-7 'VERY))
: (about-seven (6.5 0) (6.55 1e-16) (6.6 6.5536e-12) (6.65 4.3046721e-09) (6.7 4.294967296e-07) (6.75 1.525878906e-05) (6.8 0.0002821109907)
(6.85 0.003323293057) (6.9 0.02814749767) (6.95 0.1853020189) (7 1)

(7.05 0.1853020189) (7.1 0.02814749767) (7.15 0.003323293057) (7.2 0.0002821109907) (7.25 1.525878906e-05) (7.3 4.294967296e-07)
(7.35 4.3046721e-09) (7.4 6.5536e-12) (7.45 1e-16) (7.5 0))

Figure 7.2 shows the comparison of the original fuzzy set *dA* and the result of applying the fuzzy hedge "very" reiteratively at four iterations.

The simple observation of Fig. 7.2 explains the following corollary: When n tends to infinity, the n-iteration of the hedge VERY on a fuzzy number *A* produces another special fuzzy set A' whose nucleus and support is established exactly at x_0, where its membership degree equals one:

$$VERY^n(A) \rightarrow (x_0, \mu(x_0) = 1.0)$$

In a similar way, when n tends to infinity, the n-iteration of the hedge VERY on a fuzzy interval *I*, expressed by a trapezoidal membership function, produces another fuzzy interval [a, b], where a = x_2 and b = x_3, as suggested in Fig. 7.3.

Interestingly, the concept of fuzzy hedges is not entirely new for us. In the previous chapter, Sect. 6.7.2, we created the function *(fl-expand-contract-set)* that allows us to expand or to contract the support and nucleus of a fuzzy set by a real number k. The resulting fuzzy set remains centered on its original position. As an example, for the fuzzy interval represented by the Lisp expression *(a 2 3 5 6)*, we would have, for k = 2 and k = 0.5, the following results, respectively:

> (fl-expand-contract-set '(a 2 3 5 6) 2.0)
: (a 0 2 6 8)

> (fl-expand-contract-set '(a 2 3 5 6) 0.5)
: (a 3 3.5 4.5 5)

The comparison between the resulting fuzzy sets can be appreciated in Fig. 7.4. Let us see what happens when k = 0, shown at right in the same figure:

> (fl-expand-contract-set '(a 2 3 5 6) 0)
: (a 4 4 4 4)

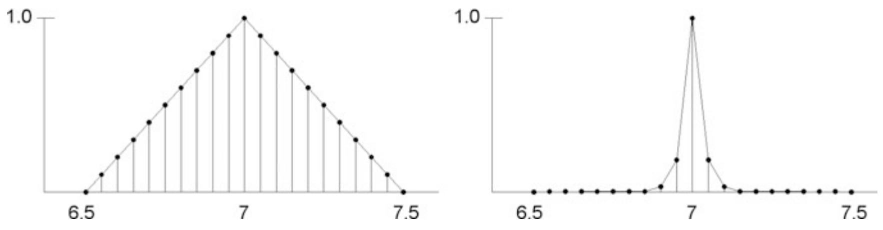

Fig. 7.2 A fuzzy set *dA* and the resulting fuzzy set after applying four times the fuzzy hedge "very" on it

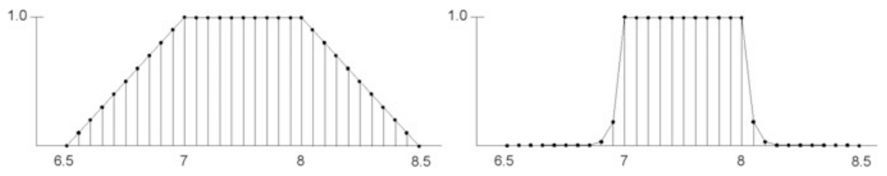

Fig. 7.3 A fuzzy interval *dI* and the resulting fuzzy set after applying four times the fuzzy hedge "very" on it

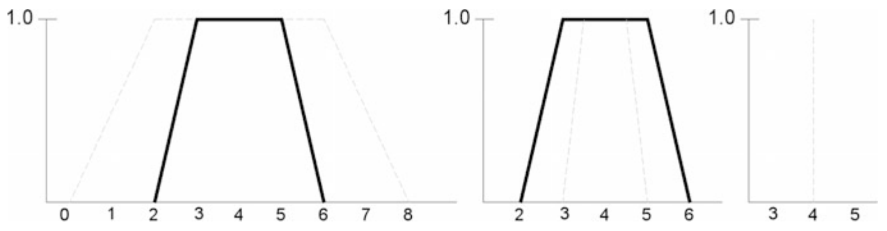

Fig. 7.4 A comparison between a fuzzy interval and some transformations after applying the FuzzyLisp function *(fl-expand-contract-set)* on it. Transformed fuzzy sets are shown in *dashed lines*

As can be seen, the result of using a value k = 0 as the second parameter in the FuzzyLisp function *(fl-expand-contract-set)* is the equivalent to apply the n-iteration of the hedge VERY on a fuzzy set. These peculiar fuzzy sets where their nucleus and support are established at a unique point x_0 on the real axis such as *(x_0, $\mu(x_0) = 1.0$)* have an important and special role to play in fuzzy logic, and we shall dedicate enough space to them later in this chapter.

Speaking rigorously, the function *(fl-expand-contract-set)* is not the same thing as using a pure fuzzy hedge, but it also concentrates or spreads-out a fuzzy set, and in some fuzzy modeling circumstances can be a valid alternative to the standard "very" and "fairly". Another valid strategy to represent the meaning of these fuzzy hedges is to conveniently use the FuzzyLisp function *(fl-fuzzy-shift)*. For example, if we represent the fuzzy set *Old* by the triangular membership function *(setq Old '(age-old 75 80 80 85))*, we can define "very old" by means of the following function call:

```
> (setq Very-old (fl-fuzzy-shift Old 5))
: (age-old 80 85 85 90)
```

And now, for keeping the semantic FuzzyLisp structure:

```
> (setq (nth 0 Very-old) 'age-very-old)
: age-very-old
```

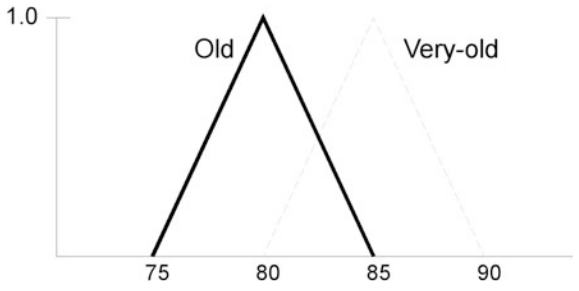

Fig. 7.5 The emerging of a linguistic variable from the use of the fuzzy set Old and a shift of it towards *right*, obtaining the fuzzy set Very-old

and then, evaluating the resulting symbol: *Very-old* → *(age-very-old 80 85 85 90)*

As can be seen in Fig. 7.5 this strategy ultimately invites ourselves to understand a linguistic variable as the result of starting with a seminal fuzzy set and then add other fuzzy sets to the linguistic variable by means of applying several shifts towards left and/or right on that set. The reader should note the huge existing flexibility in designing linguistic variables as we started to appreciate in the previous chapter.

7.7 Fuzzy Systems from Fuzzy Propositions

Exactly at 20:17:40 UTC on July 20th, 1969, Neil Armstrong landed the Eagle on the Moon. He had to land the Lunar Module semi-automatically because the resulting landing target stored on the onboard computer was not well suited for a good descent after visual inspection. Buzz Aldrin was providing him a continuous readout of altitude and velocity data from the instruments on the console and finally, the commandant of the Apollo XI mission got a successfully touchdown when less than 30 s of fuel remained available (Kranz 2009). We can guess that for getting a smooth descent he used his enormous experience as a test pilot and also the fruits of an extensive training at NASA's simulators. Even without being aware, lots of fuzzy compound propositions were inside his brain. The following one was probably one of them:

"If altitude is low and descent speed is high then thrust must be strong"

This type of fuzzy propositions can be generalized with the following expression:

$$(p \wedge q) \rightarrow r \tag{7-12}$$

Or, expressed more clearly:

$$\text{If x is } A \text{ and y is } B \text{ then z is } C \tag{7-13}$$

Until now, all the fuzzy propositions analyzed so far in this chapter, both the fuzzy sets representing predicates and the feature values associated to their respective subjects are perfectly determined, that is, they have known parameters and values that allow to calculate the truth value of a given proposition. Now we introduce a new situation where the feature value z in the consequent of expression (7-13) is unknown. This is new with respect to all the previous fuzzy compound propositions exposed so far in this chapter, but opens a door to new techniques in fuzzy logic that deserve to be explored since expression (7-13) is perhaps one of the most useful and practical ones in the theory. As an example, and again in Armstrong's mind: Given an altitude x and a descent speed y from the readouts, what must be the value z for thrust engine? Let us do a formal analysis of expressions (7-12) and (7-13) first.

Since all the parameters in the antecedent x, y, *A*, *B* from expression (7-13) are known, having into account (7-5) we can write:

$$\begin{gathered} Tv(p \wedge q) = min(\mu_A(x), \mu_B(y)) \\ Tv((p \wedge q) \rightarrow r) = min(1, 1 + \mu_C(z) - min(\mu_A(x), \mu_B(y))) \end{gathered} \tag{7-14}$$

Since z is unknown, then $\mu_C(z)$ is unknown, too, so Eq. (7-14) is impossible to solve. However, if we assume that the truth-value of the antecedent is the same as the truth-value of the consequent in (7-13), we can write:

$$\mu_C(z) = min(\mu_A(x), \mu_B(y)) \tag{7-15}$$

and substituting in (7-14):

$$Tv((p \wedge q) \rightarrow r) = min\,(1, 1 + \mu_C(z) - \mu_C(z))) \tag{7-16}$$

and this simplifies to:

$$Tv((p \wedge q) \rightarrow r) = 1 \tag{7-17}$$

This is a pivotal result for practical uses of fuzzy logic: When we assume that the truth value of the antecedent is the same as the truth value of the consequent in compound fuzzy propositions of the type expressed in (7-13) then the truth-value of $(p \wedge q) \rightarrow r$ is always one, that is, the fuzzy implication $(p \wedge q) \rightarrow r$ always holds. This should be not new for us, since we already got a hint about this point in Sect. 7.3 when we stated that if the antecedent is true in an implication then the consequent must be also true for satisfying the implication.

Expression (7-15) guarantees that we can obtain the membership degree of z to the fuzzy set *C*, but we still do not know the crisp value z in (7-13). Following our previous example, Armstrong still wouldn't know the thrust engine to apply for landing the Eagle. Let us see what can be done with the material learnt so far.

For representing "low altitude" in landing the Eagle, we can create, for example, a triangular fuzzy set *H* by means of the Lisp expression *(setq H '(low-altitude 30 100 100 170))*, where the magnitude is expressed in meters. In the same way, "high descent speed", in m/s, can be described with a triangular fuzzy set *dH* with the expression *(setq dH '(high-descent-speed 100 200 200 300))*. The whole thrust available in the Eagle propulsion system can be expressed by means of a range from 0.0 to 100.0 percent, so we can formulate "strong thrust" with the fuzzy set *sT* in the following way: *(setq sT '(strong-thrust 50 75 75 100))*. Now, giving some readouts h and s representing altitude and descending speed, respectively, expression (7-13) converts to:

If h is *H* and s is *dH* then z is *sT*

When readouts from the instruments on the Eagle are, for example, h = 79 m, and s = 190 m/s, then we can directly evaluate the antecedent in the rule just typing the following expression at the Lisp console:

```
> (fl-truth-value-p-and-q? H dH 79 190)
: 0.7
```

Now, using (7-15), we have that the membership degree of z to *sT* is $\mu_{sT}(z) = 0.7$. In this moment, the question is: Given a 0.7 membership degree to the fuzzy set "strong thrust", what is the resulting crisp value z? A first answer comes from the FuzzyLisp function *(fl-alpha-cut)* as follows:

```
> (fl-alpha-cut sT 0.7)
: (strong-thrust 67.5 82.5)
```

Both 67.5 and 82.5 are crisp values of thrust that have a 0.7 membership degree to the fuzzy set "strong thrust", but Armstrong cannot spend his time doubting between applying a 67.5 % or a 82.5 % of thrust to the Eagle spacecraft. How can we solve this situation? The answer is not far from us. In fact is hidden inside the text written in the first paragraph of this section: "Even without being aware, lots of fuzzy compound propositions were inside the brain of Armstrong". That is: in general, for obtaining an adequate and useful crisp value from expression (7-13) we need several fuzzy compound propositions working together. In other words, we need several values and fuzzy sets z_1, C_1, z_2, C_2, … z_n, C_n for creating a defuzzification procedure that finally will result into a crisp value. With a unique fuzzy proposition of the type "If x is *A* and y is *B* then z is *C*" the more we can say is that z has a $\mu_C(z)$ membership degree to C.

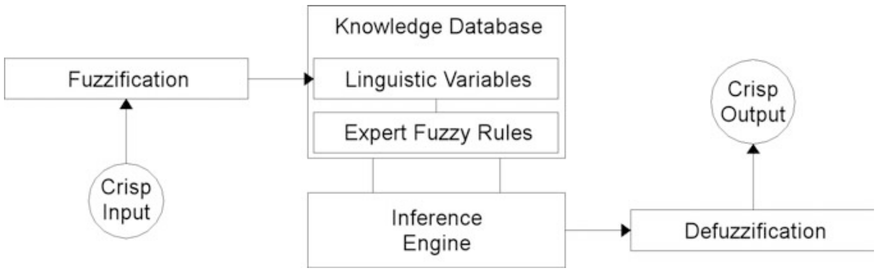

Fig. 7.6 Architecture of a Fuzzy Rule-Based System, FRBS

7.7.1 Fuzzy Rule-Based Systems

Fuzzy Rule-Based Systems, FRBS, are logical constructions that bring together several fuzzy-logic based processes and structures. The architecture of a FRBS is shown in Fig. 7.6.

The simple observation of Fig. 7.6 shows that a FRBS is a system that receives information (crisp input), processes it and then produces a crisp result. Usually the crisp input is based on available numerical information, resulting from measuring features of a physical system (e.g., density, speed, temperature, etc.) or numerical values representing any measurable entity (e.g., population, life expectancy at birth, GPD, etc.). The crisp output represents a numerical magnitude that is initially unknown but can be calculated by the inner workings of the fuzzy system. It is important to note that both the crisp input and crisp output can be formed by one or several numerical values, although in this chapter we shall be specially focused on FRBS with two numerical values as the crisp input and only one numerical output.

The first process in a fuzzy system is to fuzzify the crisp input (numerical data) and then translate this information into membership degrees. This process is known as "Fuzzification" and it is not new for us because we have already seen it in previous sections of this book. In fact, the key FuzzyLisp functions that allows to fuzzify any crisp numerical value are the functions *(fl-set-membership?)*, *(fl-dset-membership?)* and then their derived functions *(fl-lv-membership2?)* and *(fl-dlv-membership2?)*.

A Knowledge Database consists of several linguistic variables and a set of fuzzy propositions. Hereafter we shall call "rules" or "expert fuzzy rules" to the set of fuzzy propositions inside a FRBS. The term "expert" is important in this definition because every rule encompasses human knowledge, usually elicited from experts in a definite field of knowledge. Human knowledge is also embedded in the very definition of the linguistic variables at play, as we saw in the previous chapter when we spoke about fuzzy databases, so linguistic variables are also part of the knowledge database in a fuzzy system. Traditionally in fuzzy logic theory only the collection of expert fuzzy rules makes up a knowledge database, but this definition is incomplete because, as already mentioned, the expert knowledge does not come only from the fuzzy rules in the system, but from the meaning of the fuzzy sets that form the linguistic variables of the system.

The inference engine in a fuzzy system has a simple mission, as its name suggests: To make inferences. Until the inference engine starts to work, the only things we have in a fuzzy system are membership degrees, lots of fuzzy sets and expert fuzzy rules "floating up in the air". The inference engine is the glue that interacts with all these elements. It reads a rule, identifies its antecedent and consequent, finds the definition of a fuzzy set, gets the required membership degree, notes down this temporal data, obtains more membership degrees until the antecedent is computationally complete and then performs the logical inference in the rule, passing the obtained numerical values to the consequent. Then it processes the following rule and repeats the same procedure until finishing the complete set of rules. Using the same notation as in (7-13), the output of the inference engine in a FRBS has the following structure:

From Rule$_1$: "z is C_1" has an associated truth value $\mu_{C1}(z)$
From Rule$_2$: "z is C_2" has an associated truth value $\mu_{C2}(z)$
...
From Rule$_n$: "z is C_n" has an associated truth value $\mu_{Cn}(z)$

Or, written it formally, we can say that the inference engine ultimately produces the following set of numerical values (membership degrees):

$$Fuzzy\ Output = \{\mu_{C1}(z), \mu_{C2}(z), \ldots, \mu_{Cn}(z)\} \tag{7-18}$$

The procedure for obtaining a crisp output from the set of membership degrees in (7-18) is called "Defuzzification" and we shall discuss it immediately after stating an important remark: In a FRBS there is not a nucleus, that is, we can not say that the inference engine, or the knowledge database is the nucleus of the system. A Fuzzy Rule Based System has a holistic nature where all the subsystems work in a status of synergy. This is one of the strongest strengths of fuzzy systems.

7.7.2 Defuzzification

Let us do an analogy between the set of fuzzy rules of a knowledge database and the board of directors of a Club. In this analogy, every rule is represented by a member of the board where every member has a definite opinion on a certain subject matter that is represented in the analogy by a membership degree $\mu_{Ci}(z)$. The members of the board have a complete analysis and integral representation of the subject matter, but a final agreement must be reached, a final decision that democratically must take into account every member's opinion. This situation describes rather well the expression (7-18): We have a complete set of membership degrees as the result of the inferences made in the fuzzy system, but we need to come back to the realm of crisp values in order to get a final, definite result. Just imagine the Club is trying to

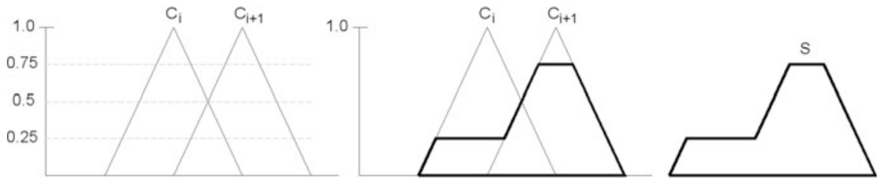

Fig. 7.7 Creating a geometrical shape from membership degrees in the fuzzy sets belonging to consequents in fuzzy rules

approve a final budget for the next year: they need a crisp quantity. The partners must know how much money the Club will have at its disposal for the next year. We need to obtain the *z*, crisp output from the system.

There are several defuzzification methods in fuzzy logic theory. For convenience in our exposition, we shall assume initially that only some expert fuzzy rules from the FRBS haven been fired in such a way that only two $\mu_{Ci}(z)$, $\mu_{Ci+1}(z)$ membership degrees from the fuzzy output expressed by (7-18) have a value bigger than zero. Then, if C_i, C_{i+1} are fuzzy sets belonging to the consequents of the knowledge database of fuzzy rules, we can create a geometrical shape as shown in Fig. 7.7.

As can be seen at left in the figure, we have chosen $\mu_{Ci}(z) = 0.25$, $\mu_{Ci+1}(z) = 0.75$ on the fuzzy sets C_i, C_{i+1}, producing a sort of mountain-shaped form S than can be easily observed at right.

An additional step in the process of defuzzification is shown in Fig. 7.8, where the shape S can be imagined, for a better exposition, as a materialized solid form made of metal, wood or any solid and rigid material you can name in this moment. This materialized shape S has, obviously, a center of gravity, G. Projecting the two-dimensional point G over the horizontal axis we obtain a real value x that represents the crisp, defuzzified numerical value *z*.

This method of defuzzification, known as the "Centroid Method" has a strong advantage that dwells from its intuitive formulation: it is very easy to understand. In fact, this is the reason why we have chosen it as the first one of the several defuzzification methods available. However it has two important disadvantages. First, it can become tedious to calculate, especially if the shape of the characteristic functions of the fuzzy sets C_i are Gaussian, bell-shaped, etc. The second disadvantage is that the Centroid Method is unable to provide a crisp output value *z* at the

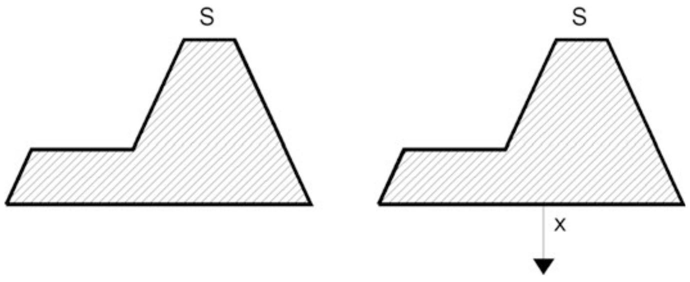

Fig. 7.8 Materializing the shape S and getting a crisp output x

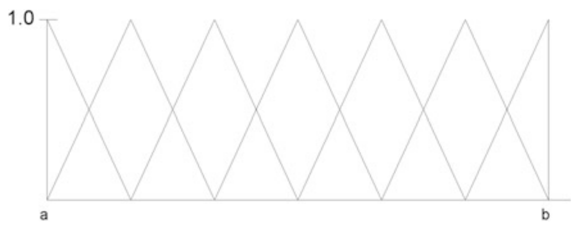

Fig. 7.9a A linguistic variable composed by seven fuzzy sets defined in the range [a, b]

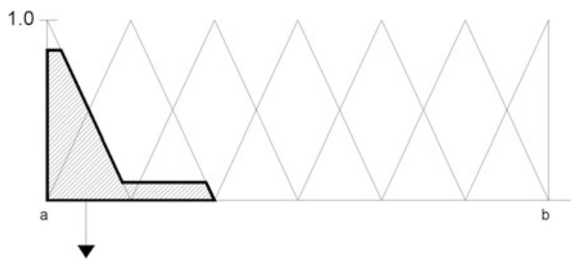

Fig. 7.9b Extreme *a* at *left* is never reached using the Centroid Method of defuzzification. The same happens with the extreme *b* at *right*

extremes of the linguistic variable where the fuzzy sets C_i are defined. Figures 7.9a and 7.9b illustrate this problem.

In Fig. 7.9a we see a linguistic variable composed by seven fuzzy sets. All of them have an isosceles triangular characteristic function except the two ones located at the extremes of the universe of discourse of the linguistic variable. These ones have a right triangle as characteristic functions in order to adapt the linguistic variable to its extremes *a*, *b*.

Very interestingly, Fig. 7.9b shows a situation where despite any possible value of the membership degrees in the consequents of a FRBS, a crisp value obtained by means of the Centroid Method of defuzzification never reaches the extremes of the linguistic variable. The figure shows a shape whose center of gravity is not far from the extreme *a*, but it never reaches it, even if only the leftmost fuzzy set had an associated membership degree equal to one and the rest of fuzzy set in the linguistic

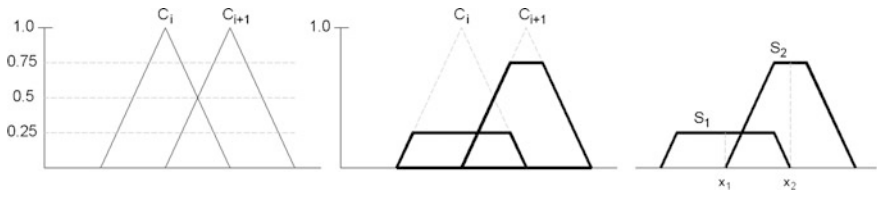

Fig. 7.10 Creating two geometrical shapes S_1, S_2 from membership degrees

variable had a null associated membership degree. The crisp output never reaches the extreme. FuzzyLisp does not support the Centroid Method of defuzzification because these two reasons.

Another well-known defuzzification method is the so named "Weighted average method". This defuzzification strategy is easily understood if we observe first Fig. 7.10. Again using only two $\mu_{Ci}(z)$, $\mu_{Ci+1}(z)$ membership degrees from the fuzzy output expressed by (7-18), then, if C_i, C_{i+1} are fuzzy sets belonging to the consequents of the knowledge database of fuzzy rules, we can create two geometrical shapes S1, S2 as shown in the figure.

Now, taking the vertical axis of symmetry of every shape we obtain the crisp values x_1, x_2 just on the real axis. If the associated membership degrees are, respectively, $\mu_1(z)$, $\mu_2(z)$, then the resulting defuzzified crisp value z is given by the formula:

$$z = (x_1\mu_1(z) + x_2\mu_2(z))/(\mu_1(z) + \mu_2(z)) \tag{7-19}$$

In general, for n membership degrees from the fuzzy output expressed by (7-18), the Weighted average method uses the following generalized expression:

$$z = \sum(x_i m_i(z))/\sum \mu_i(z) \tag{7-20}$$

The main advantage of the weighted average method is its easy implementation, although this method is only valid for cases where the output membership functions are symmetrical in shape. A variation of the weighted average method results from substituting membership degrees to "weights" associated to every shape obtained from fuzzy sets belonging to the consequents of the knowledge database of fuzzy rules.

As can be seen in Fig. 7.11, to every resulting shape S_i we obtain a weight w that is proportional to the area of the shape. In this example, the area of S_2 is 2.14 times the area of S_1. Applying these weights in the center of the basis of their respective

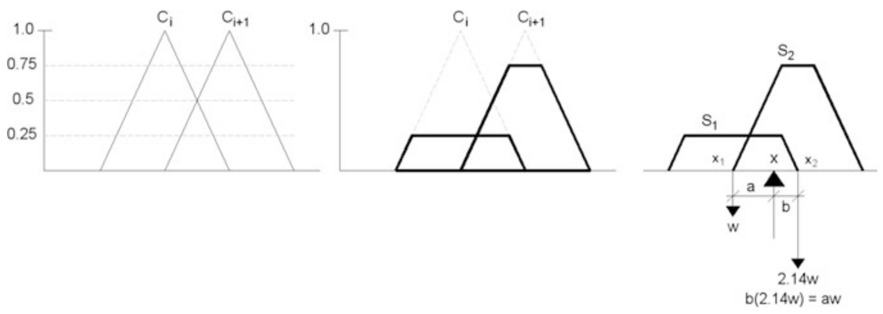

Fig. 7.11 Applying weights to shapes

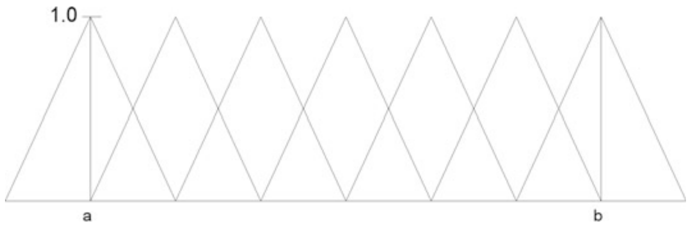

Fig. 7.12 Defuzzification: reaching the extremes

shapes we obtain a set of weights w_i. The defuzzificated crisp value z corresponds to a value x (shown in the figure by an arrow pointing upwards) on the real axis that equilibrates all the weights at play. Then, expression (7-20) converts to:

$$z = \sum x_i w_i / \sum w_i \tag{7-21}$$

The two exposed weighted average methods are easy to calculate, and the second one adds the concept of "weighting area" already implicit in the Centroid Method of defuzzification. However, we still cannot reach the extremes of the output linguistic variables, unless we strategically design its fuzzy sets as shown in Fig. 7.12.

This arrangement of fuzzy sets in a linguistic variable can seen bizarre at first, but does not violate any theoretical matter on fuzzy sets and solves the biggest inconvenience of the three defuzzification methods exposed so far.

7.7.2.1 Singletons

When fuzzy sets in a linguistic variable defining the output of a FRBS are formed by isosceles triangles and are arranged strategically as in Fig. 7.12, the two exposed versions of the weighted average method for defuzzification always apply the resulting membership degrees/weights on the vertical axis of symmetry of their characteristic functions. This suggests the use of another type of fuzzy set named "Singleton" that we are going to introduce just now with the following definition:

In fuzzy sets theory a singleton S is a special fuzzy set whose membership function is equal to zero except at a point x, where its membership degree is exactly equal to one. A singleton is always associated to a crisp real number x_0 in such a way that its characteristic function can be defined formally as:

$$\mu S(x) = 1 \quad \textit{if}\, x = x_0$$
$$\mu S(x) = 0 \quad \textit{if}\, x \neq x_0$$

The FuzzyLisp function *(fl-expand-contract-set)* already heralded Singletons in Sect. 7.6 of this chapter (source code number 6-18) when it takes a fuzzy set in

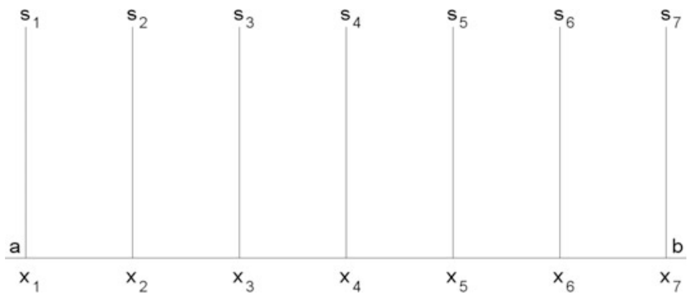

Fig. 7.13 Substituting isosceles shaped characteristic functions by singletons, S_i

FLSSR representation as its first argument and applies zero as its second argument. Again at the Lisp prompt:

```
> (fl-expand-contract-set '(a 2 3 5 6) 0)
: (a 4 4 4 4)
```

As can be seen from the list returned by Lisp, both the nucleus and support of a singleton is a single point on the real axis. The graphical representation of a singleton *S* can be seen at the right of Fig. 7.4. Under a practical point of view the list *(a 4 4 4 4)* is not computationally efficient for representing singletons, so in FuzzyLisp they are represented by expressions of the following type:

$$(singleton\text{-}label\, x_0)z$$

After several assignments of the type *(setq* s_i *(singleton-label*$_i$ x_i*))* a linguistic variable composed by singletons can be built by means of the following expression: *(setq singleton-LV '(*s_1 s_2 ... s_n*))*. For our practical interests, we can now represent the previous linguistic variable composed by seven triangular shaped fuzzy sets shown in Fig. 7.12 by means of seven singletons S_1 ... S_7 as shown in Fig. 7.13.

Since every singleton can be seen as a geometrical spike resting fixed on a real point x_i, we can apply the weighted average method of defuzzification for singletons in order to obtain the crisp output of a FRBS. After some reflection you will able to appreciate that, for defuzzification purposes, Fig. 7.13 is essentially the same as 7.12 in disguise. This is the reason behind formula (7-20) serves perfectly well for the defuzzification procedure if singletons are used as fuzzy sets in the consequents of every fuzzy rule in a FRBS. In fact, the output of an inference engine in a FRBS will have now the following structure:

From Rule$_1$: "z is S_1" has an associated truth value $\mu_1(z)$ *in point* x_1
From Rule$_2$: "z is S_2" has an associated truth value $\mu_2(z)$ *in point* x_2
...
From Rule$_n$: "z is S_n" has an associated truth value $\mu_n(z)$ *in point* x_n

and finally, for obtaining the crisp output *z* of the system, we can write again:

Fig. 7.14 Michio Sugeno at the European Centre for Soft Computing, Spain, circa 2010 (Photograph by the author)

$$z = \sum (x_i \mu_i(z)) / \sum \mu_i(z) \tag{7-22}$$

Michio Sugeno from Japan, one of the most important names in Fuzzy Logic, has been one of the researchers that first explored the elegance and computational power of singletons in fuzzy systems. As we shall see immediately, singleton defuzzification is the standard defuzzification method used by FuzzyLisp (Fig. 7.14).

7.8 Modeling FRBS with FuzzyLisp

Section 7.7 has been strongly theoretical, and it is now the time to introduce a few important FuzzyLisp functions in order to model Fuzzy Rule Based Systems. Since this is a practical book following a "learning by doing" approach, the next paragraphs will show the reader how to model the workings of an air-conditioner controller by means of fuzzy logic techniques. In this moment it is not important for us to model the best, finest air-conditioner controller available, but to show how to organize and process all the required information for the controller, or, expressed with other words, introduce a practical application example where the FRBS architecture exposed in Fig. 7.6 is translated into real FuzzyLisp code.

As it is well known, an air-conditioner system's goal is to maintain the temperature of a given enclosure at a constant value, where the enclosure can be a room, a building, an aircraft cockpit or the interior of a car, to name only some examples. The basic information needed to accomplish the goal is usually based on two values: actual temperature *t* and temperature variation *delta-t* obtained from two readings from a thermometer obtained every *s* seconds. Other interesting input values could be the quantity of solar radiation received by the enclosure and the time of the day and even the day of the year, but *t* and *delta-t* will be enough for our simple model. After the readings are made, our fuzzy controller will calculate the needed output airflow and temperature, AFT, to stabilize the enclosure's

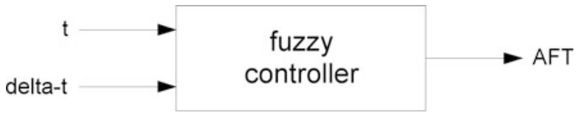

Fig. 7.15 Temperature and Delta-temperature are the inputs to the air-conditioner controller. The output is temperature and air flow, AFT

temperature at a temperature goal, let us say, T = 21 °C. A general formulation of the problem is given by the following expression:

$$AFT = f(t, delta\text{-}t) \tag{7-23}$$

Figure 7.15 shows graphically a representation of the intended fuzzy controller. If the reader is versed on differential equations then is highly probable that a ring has sounded in his head after the simple observation of expression (7-23). However no one differential equation will be needed to obtain a good working air-conditioner controller.

Since we have two input values, we shall create two specific linguistic variables to represent them. The first step is to define the ranges of these two linguistic variables: For representing temperature we shall use a range of 0–50 °C, while for temperature variation we shall use a range from −2.0 to 2.0 °C per minute. Now we need to create a partition on the linguistic variables by means of defining suitable fuzzy sets on them. Tables 7.4a and b show the fuzzy sets that make up the input linguistic variables lv-temperature and lv-delta-t, respectively.

In a similar way, Table 7.4c shows the fuzzy singletons that make up the output linguistic variable AFT (Air Flow Temperature). The final, crisp output of the controller will be positive if hot air is needed to stabilize the system. Conversely, it

Table 7.4a Fuzzy sets for the linguistic variable lv-temperature

Fuzzy set label	Membership function	Lisp representation
Very-cold	0, 0, 6, 11	*(setq T1 '(very-cold 0 0 6 11))*
Cold	6, 11, 16, 21	*(setq T2 '(cold 6 11 16 21))*
Optimal	16, 21, 21, 26	*(setq T3 '(optimal 16 21 21 26))*
Hot	21, 26, 31, 36	*(setq T4 '(hot 21 26 31 36))*
Very-hot	31, 36, 50, 50	*(setq T5 '(very-hot 31 36 50 50))*

Table 7.4b Fuzzy sets for the linguistic variable lv-delta-t

Fuzzy set label	Membership function	Lisp representation
Decreasing	−2, −2, −0.2, 0	*(setq dT1 '(decreasing -2-2 -0.2 0))*
No-change	−0.2, 0, 0, 0.2	*(setq dT2 '(no-change -0.2 0 0 0.2))*
Increasing	0, 0.2, 2, 2	*(setq dT3 '(increasing 0 0.2 2 2))*

Table 7.4c Fuzzy sets for the output linguistic variable AFT (Air Flow Temperature)

Fuzzy set label	Singleton at x_0	Lisp representation
Cold-strong	−100	*(cold-strong -100)*
Cold-medium	−50	*(cold-medium -50)*
Stop	0	*(stop 0)*
Heat-medium	50	*(heat-medium 50)*
Heat-strong	100	*(heat-strong 100)*

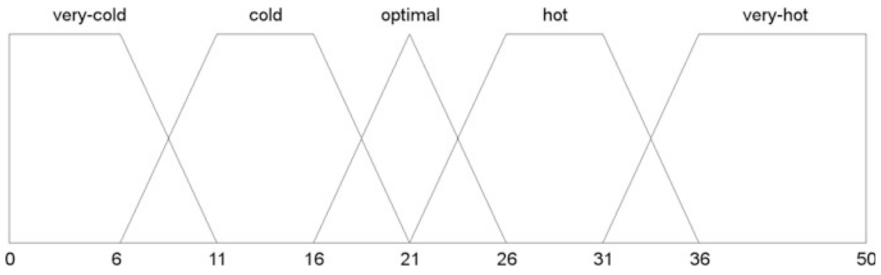

Fig. 7.16a Input linguistic variable lv-temperature

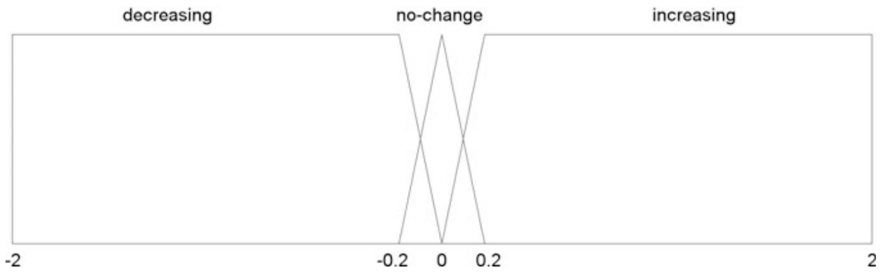

Fig. 7.16b Input linguistic variable lv-delta-t

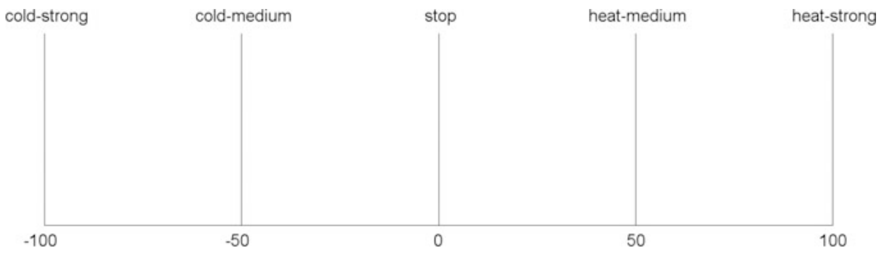

Fig. 7.16c Output linguistic variable AFT

will be negative if cold air is required. The numerical magnitude expresses the required flow of air in a range from 0 to 100 %, that is, from null to maximum output of the system.

Figures 7.16a, 7.16b and 7.16c show a graphical representation of the input and output variables of the air-conditioner controller.

After all the required input and output linguistic variables have been established we need a set of expert rules of the type given by expression (7-13) that links fuzzy sets from the input linguistic variables to the fuzzy singletons in the output. One example of rule for our fuzzy controller is the following one:

If t is very-cold and delta-t is decreasing then AFT is heat-strong

In plain English this rule means that if the temperature of the room is very cold and the temperature is decreasing, then the controller must react supplying hot air at a strong high rate flow. In Code 7-8 we can see the complete FuzzyLisp application for our air-conditioner fuzzy controller:

```
;code 7-8
(load "my-path/fuzzy-lisp.lsp")
;fuzzy sets and linguistic variables definitions:
(setq T1 '(very-cold 0 0 6 11))
(setq T2 '(cold 6 11 16 21))
(setq T3 '(optimal 16 21 21 26))
(setq T4 '(hot 21 26 31 36))
(setq T5 '(very-hot 31 36 50 50))
(setq lv-temperature '(T1 T2 T3 T4 T5))

(setq dT1 '(decreasing -2-2 -0.2 0))
(setq dT2 '(no-change -0.2 0 0 0.2))
(setq dT3 '(increasing 0 0.2 2 2))
(setq lv-delta-t '(dT1 dT2 dT3))

(setq AFT '(
     (cold-strong -100)
     (cold-medium -50)
     (stop 0)
     (heat-medium 50)
     (heat-strong 100))
)

;fuzzy rules section:

(setq rules-controller '((lv-temperature lv-delta-t AFT)
   (very-cold decreasing heat-strong AND-product)
   (very-cold no-change heat-strong AND-product)
   (very-cold increasing heat-strong AND-product)

   (cold decreasing heat-strong AND-product)
   (cold no-change heat-medium AND-product)
```

```
        (cold increasing heat-medium AND-product)

        (optimal decreasing heat-medium AND-product)
        (optimal no-change stop AND-product)
        (optimal increasing cold-medium AND-product)

        (hot decreasing cold-medium AND-product)
        (hot no-change cold-medium AND-product)
        (hot increasing cold-strong AND-product)

        (very-hot decreasing cold-strong AND-product)
        (very-hot no-change cold-strong AND-product)
        (very-hot increasing cold-strong AND-product))
   )
```

After the first lines of comments, the expression *(load "my-path/fuzzy-lisp.lsp")* tells NewLisp where to find FuzzyLisp. Here you will have to replace "my-path" for the path where the file fuzzy-lisp.lsp is located in your computer.

The following lines in the code create the fuzzy sets and linguistic variables already exposed in Tables 7.4a, b and c for the controller. Please note the used structure for the output linguistic variable AFT. Since singletons can be formed by very simple Lisp expressions, the output linguistic variable can be expressed in a more compact way when compared to the input linguistic variables.

After all the fuzzy sets and linguistic variables are created, then it comes the fuzzy rules section with a very definite ordering. Several important details must be remarked in this structure of rules. First, the complete set of fuzzy rules need a Lisp symbol for a proper identification. In this case we have chosen "rules-controller". In the same line of code, we have declared the names of the two input variables and the output linguistic variable, as can be seen in the very first sublist: *(lv-temperature lv-delta-t AFT)*. FuzzyLisp needs to know beforehand the ordering of the linguistic variables in the body of rules. Expressed with other words: The first rule in the body of rules in FuzzyLisp is in itself not a rule but an enumeration of the used linguistic variables.

Another important point is the use of the symbol `AND-product` in the rules. FuzzyLisp allows the designer to use several logical connectives in fuzzy rules. If the symbol `AND-min` is used, then expression (7-5) will be used for inferences, that is:

$$Tv(p \wedge q) = min(Tv(p), Tv(q)) = min(\mu_A(x), \mu_B(y))$$

However, when using `AND-product`, then the following inference is applied:

$$Tv(p \wedge q) = Tv(p) \cdot Tv(q) = \mu_A(x) \cdot \mu_B(y) \qquad (7\text{-}24)$$

That is, FuzzyLisp multiplies the membership degrees $\mu_A(x)$, $\mu_B(y)$. This usually produces a very smooth, continuous crisp output in the system. For expressing an "or" logical connective FuzzyLisp uses the symbol `OR-max` as defined by expression (7-6). Other symbols could be added in future versions of FuzzyLisp representing different ways of conjunction and disjunction (the interested reader will find more information in fuzzy-logic theoretical books under the terms “t-norms” and “t-conorms”. The study of t-norms and t-conorms is outside the scope of this book).

Another important remark emerges when observing the last rule:

(very-hot increasing cold-strong AND-product))

Just note the double closing parenthesis. As you know, in every Lisp program the number of left parenthesis equals the number of right parenthesis. Since the construction of a set of expert fuzzy rules in FuzzyLisp usually results into a relatively complex expression, special care should be put on matching left and right parenthesis. If a fuzzy model developed with FuzzyLisp crashes or does not behaves well, the first place to look is in the declaration part of rules and linguistic variables.

Code 7-8 represents the complete knowledge database for the fuzzy controller. Now, for fulfilling the architecture of a FRBS as shown in Fig. 7.6, we need a fuzzification-defuzzification mechanism and an inference engine. All these software components are included as functions in FuzzyLisp and now it is time to uncover them.

Maybe the most important function in FuzzyLisp is *(fl-translate-rule)*. This function is the nucleus of its inference engine. It takes a rule at a time from the fuzzy rules body, performs the adequate inferences and translates it into membership degrees. As arguments, it needs the first sublist from the body of rules where the enumeration of the used linguistic variables is expressed. In our controller example, the needed sublist is *(lv-temperature lv-delta-t AFT)*. The second argument is the rule to “translate”, as for example, *(optimal increasing cold-medium AND-product)*. Finally, the third and fourth arguments are the crisp input numerical values to the FRBS. In our example these values correspond to a given crisp values of temperature *t* and temperature variation *delta-t*. For example, *t* = 22C and *delta-t* = 0.25 C/min. Code 7-9 shows the function:

```
;code 7-9
(define (fl-translate-rule header rule x y,
    lv1 lv2 lv3 fset1 fset2 fset3 mu1 mu2 s
    operator m1 m2 wi wixi)

   ;extract linguistic variables from the header:
   (setq lv1 (nth 0 header) lv2 (nth 1 header) lv3 (nth 2
     header))
   ;extract fuzzy sets from the rule:
   (setq fset1 (nth 0 rule) fset2 (nth 1 rule) fset3 (nth 2
     rule))
   ;extract the operator in the rule:
   (setq operator (nth 3 rule))

   ;get first lv, calculate memberships given x and then
   ;select the membership degree corresponding to the first
```

```
    ;set in the rule:
    (setq mu1 (assoc fset1 (fl-lv-membership2? (eval lv1) x)))

    ;now do the same for second lv, y and second set in the
       rule:
    (setq mu2 (assoc fset2 (fl-lv-membership2? (eval lv2) y)))

    ;now get the appropiate singleton, s:
    (setq s (assoc fset3 (eval lv3)))

    ;extract the membership degrees:
    (setq m1 (last mu1) m2 (last mu2))
    (case operator
       (AND-product
          ;apply AND-product by making wi=m1xm2:
          (setq wi (mul m1 m2)) (setq wisi (mul wi (last s)))
       )
       (AND-min
          ;apply AND-min by making wi=min(m1,m2):
          (setq wi (min m1 m2)) (setq wisi (mul wi (last s)))
       )
       (OR-max
          ;apply OR-max by making wi=max(m1,m2):
          (setq wi (max m1 m2)) (setq wisi (mul wi (last s)))
       )
    );case end
    ;finally return the numerical translation of the rule
    (list m1 m2 wi wisi)
)
```

Function *(fl-translate-rule)* begins reading the header of the body rules *(first sublist)* and extracting the linguistic variables from the header and then the fuzzy sets at play. Later, several calls to the function *(fl-lv-membership2?)* are responsible for the fuzzification of the crisp input, storing the obtained membership degrees in the internal variables *mu1* and *mu2*. The singleton from the consequent in the rule is identified as it happens also with the operator located at the end of the analyzed rule. The second part of the function works by means of a *case* Lisp statement, selecting the appropriate inference to be performed on the rule. Finally, the function returns a list containing the membership degrees to the first and second fuzzy sets in the antecedent of the rule, then the result of the performed numerical inference, and finally the numerical value after multiplying it by the corresponding singleton in the consequent. Let us take as example rule number nine:

(optimal increasing cold-medium AND-product)

Calling the function we obtain:

> (fl-translate-rule (first rules-controller) (nth 9 rules-controller) 22 0.25)
: (0.8 1 0.8 -40)

The value 0.8 is the membership degree of 22 °C to the fuzzy set *optimal*, while 1 is the membership degree of 0.25 °C per minute to the fuzzy set *increasing*. The third value is the result of multiplying both membership degrees (inference dictated by the symbol *AND-product*), and the last value is obtained after multiplying 0.8 by the value associated to the singleton *cold-medium* (-50). Just note the appropriate use of the Lisp functions *(first)* and *(nth)* in the previous expression. This gives us a hint to create a function that will evaluate all the fuzzy rules contained in the knowledge database. This function, named *(fl-translate-all-rules)* is shown in Code 7-10:

```
;code 7-10
(define (fl-translate-all-rules set-of-rules x y, n i
   list-out)
   (setq list-out '())
   (setq n (length set-of-rules))
   (setq i 1);indexed on the first rule

   (while (< i n)
     (setq list-out (cons
             (fl-translate-rule (first set-of-rules)
             (nth i set-of-rules) x y) list-out))
     (++ i)
   );while end
   (reverse list-out)
)
```

as can be easily noted observing the code, this function scans all the rules from the body of rules and returns a list containing all the calculations made for each of them. As parameters, the function takes the name of the complete body of rules and two crisp input values to feed the system. Let us try the function at the Lisp prompt:

> (fl-translate-all-rules rules-controller 22 0.25)
: ((0 0 0 0) (0 0 0 0) (0 1 0 0) (0 0 0 0) (0 0 0 0) (0 1 0 0) (0.8 0 0 0) (0.8 0 0 0) (0.8 1 0.8 -40) (0.2 0 0 0) (0.2 0 0 0) (0.2 1 0.2 -20) (0 0 0 -0) (0 0 0 0) (0 1 0 0))

From the output we can see that rules #9 and #12 have been fired in the example when t = 22 °C and *delta-t* = 0.25 °C/min are used as crisp inputs. This is easy to observe because the last element in their respective sub-lists is not null. In fact, calling the functions *(fl-translate-rule)* and *(fl-translate-all-rules)* from the Lisp prompt is a good strategy when we whish to debug and/or in depth analyze a fuzzy application made with FuzzyLisp. Now it is time to take the output of the function *(fl-translate-all-rules)* and apply a defuzzification procedure to it. A new FuzzyLisp function, named *(fl-defuzzify-rules)* is dedicated specifically to this task, as shown in Code 7-11:

```
; Code 7-11
 (define (fl-defuzzify-rules translated-rules,
             i n sum-wi sum-wixi)
    (setq i 0)
    (setq sum-wi 0) (setq sum-wixi 0)
    (setq n (length translated-rules))
    (while (< i n)
        (setq sum-wi (add sum-wi (nth 2 (nth i translated-
            rules))))
        (setq sum-wixi (add sum-wixi
                        (nth 3 (nth i translated-rules))))
        (++ i)
    )
    (div sum-wixi sum-wi)
)
```

The function takes as arguments the output list of the function *(fl-translate-all-rules)* and then traverses it in order to apply the defuzzification mathematical expression (7-22), providing a crisp numerical result to the fuzzy system. Let us try this function at the Lisp prompt:

> (fl-defuzzify-rules (fl-translate-all-rules rules-controller 22 0.25))
: *-60*

So, our model tells that when temperature is $t = 22$ °C and variation of temperature is *delta-t* = 0.25 °C/min, the airflow temperature AFT must be AFT = −60. This is a great result, but once again, the call to the function *(fl-defuzzify-rules)* suggests the development a more comfortable to use one, named *(fl-inference)*, as shown in Code 7-12:

```
;code 7-12
(define (fl-inference rules x y)
     (fl-defuzzify-rules (fl-translate-all-rules rules x y))
)
```

This function puts it all together, allowing the NewLisp user to obtain the crisp output of a FRBS using only three parameters: The name of the complete body of

Table 7.5 Airflow temperatures AFT obtained as the crisp result of the fuzzy air-conditioner controller

	0	3	8.5	13.5	18.5	23.5	28.5	33.5	36.5	50
−1	100	100	100	100	75	0	−50	−75	−100	−100
−0.15	100	100	93.75	87.5	62.5	−6.25	−50	−75	−100	−100
−0.10	100	100	87.5	75	50	−12.5	−50	−75	−100	−100
−0.05	100	100	81.25	62.5	37.5	−18.75	−50	−75	−100	−100
0	100	100	75	50	25	−25	−50	−75	−100	−100
0.05	100	100	75	50	18.75	−37.5	−62.5	−81.25	−100	−100
0.10	100	100	75	50	12.5	−50	−75	−87.5	−100	−100
0.15	100	100	75	50	6.25	−62.5	−87.5	−93.75	−100	−100
1	100	100	75	50	0	−75	−100	−100	−100	−100

Input temperatures are shown in the first row. Temperature variation is shown in the first column

rules and then the crisp input to the system, that is, two numerical values. The function call for our air-conditioner controller example cannot be simpler:

> (fl-inference rules-controller 22 0.25)
: *-60*

In practical terms, this means that for developing fuzzy-models based on the architecture shown in Fig. 7.6 where two crisp values are used as input, a FuzzyLisp user only needs to declare fuzzy sets, linguistic variables and a set of expert fuzzy rules with the structure exposed in Code 7-8 and then, by simple callings to the function *(fl-inference)* all the internal processes of fuzzification, knowledge database management and defuzzification are automatically performed. In Chap. 1 of this book we stated that Lisp is a powerful language. I hope that once we have reached this milestone the reader will agree.

For extensively testing our air-conditioner fuzzy controller we can make successive calls to the function *(fl-inference)*, obtaining, for example, the results shown in Table 7.5. In the first row you can see some values for temperatures, *t*, while the first column at left represents some values for temperature variation, *delta-t*. Needless to say, the rest of values in the table represent output airflow temperatures, AFT.

Albeit impressive, this table requires 90 function calls to *(fl-inference)*, and this is a tedious task to perform manually. Fortunately, Lisp, besides allowing us to express our ingenuity and mathematical creativity, gives us also, as it happens with other computer languages, the opportunity to automatize repetitive work. This is the philosophy behind the function *(fl-3d-mesh)*. This FuzzyLisp function is in fact a procedure for creating tables as the one shown above these lines. Let us observe it in Code 7-13:

```
;code 7-13
(define (fl-3d-mesh namefile rules nx ny,
                        x1 x2 y1 y2 header lv1 lv2 stepx stepy x y)
```

```
    (setq header (nth 0 rules))
    ;get the extremes x1 x2 of the first lv
    (setq lv1 (nth 0 header) lv2 (nth 1 header))
    (setq x1 (nth 1 (eval (first (eval lv1)))))
    (setq x2 (last (eval (last (eval lv1)))))
    ;now get the extremes y1 y2 of the second lv
    (setq y1 (nth 1 (eval (first (eval lv2)))))
    (setq y2 (last (eval (last (eval lv2)))))

    (setq stepx (div (sub x2 x1) nx))
    (setq stepy (div (sub y2 y1) ny))
    (setq x x1 y y1)
    (println "Writing 3Dmesh ...")
    (device (open namefile "write"))
    (while (<= x x2)
       (while (<= y y2)
          (print x "," y "," (fl-inference rules x y) "\n")
          (setq y (add y stepy))
       ); end while y
       (setq y y1); reset y value
       (setq x (add x stepx))
    );end while x
    (close (device))
    (println "3Dmesh written to file")
)
```

Basically speaking, this function scans the range of two linguistic variables used as antecedents in a FRBS by making successive calls to the function *(fl-inference)* with changing input crisp values x,y inside the range of the linguistic variables at play. Let us try a call example in order to better understand how does it work:

>(fl-3d-mesh "air-conditioner-controller.csv" rules-controller 20 20)
: Writing 3Dmesh ...
3D mesh Written to File

As the output of the function call suggests, *(fl-3d-mesh)* writes a file to the computer's hard disk where all the calculations resulting from adequate calls to *(fl-inference)* are stored. As arguments, this function requires a string representing a filename for the desired output file, the name of the complete body of rules of the fuzzy model and then two numerical values nx, ny that inform the function how many discretization steps are needed for building the "table" of output data. As a rule of thumb, values of nx, ny between 20 and 50 are enough for any practical application. Let us take a view to the contents of the output file air-conditioner-controller.csv:

```
0,-2,100
0,-1.8,100
0,-1.6,100
...

22.5,-0.4,20
22.5,-0.2,20
22.5,0,-15
22.5,0.2,-65
22.5,0.4,-65
...

50,1.6,-100
50,1.8,-100
50,2,-100
```

As can be seen, the file has a comma separated value (csv) format, so you can open it not only in any text editor program such as Notepad or TextEdit, but also in Excel and many other programs that accept csv-type files for data importation. Since every line can be interpreted as the coordinates of a 3D point in space, many technical programs allows to create a 3D visualization of the interpolated three dimensional surface, as we can appreciate in Fig. 7.17.

While the base of the figure is represented by a bi-dimensional plane formed by *t* and *delta-t* (corresponding to the linguistic variables *lv-temperature* and *lv-delta-t*,

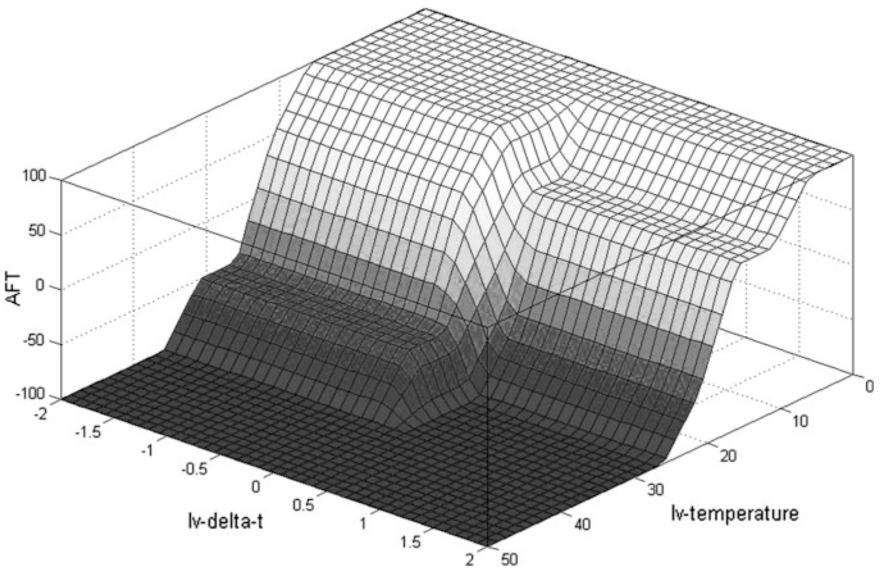

Fig. 7.17 Output surface of the fuzzy air-conditioner controller

respectively), the height of every point belonging to the 3D surface represents the output airflow temperature calculated by the fuzzy controller, AFT. As can be seen, dark areas in the surface represent low, negative values of AFT, while lighter ones represent positive AFT values. Several remarks must be made at this point: The first one is the steep region located about the middle of the surface. This is caused mainly because the relatively small support of the fuzzy set *no-change* in the input linguistic variable *lv-delta-t* and the large support of the fuzzy sets *decreasing* and *increasing* in the same linguistic variable. The second remark points to the presence of some flat regions in the output surface that we shall name "terraces", including the big decks corresponding to AFT = −100 and AFT = 100. While there are lots of creative work and parameters to choose when creating a FRBS, one thing is certain: If trapezoidal fuzzy sets are used on the linguistic variables that make up a fuzzy application then terraces will always appear on the resulting output surface. The existence of terraces is not, by itself, an inconvenient, and can be just the desired design of the fuzzy application, but if you don't desire terraces in your output surfaces you will need to use only triangular membership functions in your models. The reader is invited to modify and play with the fuzzy sets that make up the linguistic variables *lv-temperature* and *lv-delta-t.* As an example, the fuzzy set *no-change* can be expanded by means of using the FuzzyLisp function *(fl-expand-contract-set)* from the Lisp prompt, the fuzzy sets *decreasing* and *increasing* can be modified, shortening their support and nuclei, etc. The best advice in order to acquire experience in designing fuzzy systems is to play with them and observe their behavior. In the rest of this chapter some extra advice will be given for this conceptual challenge and also in the next chapter, but nothing replaces experience and experimentation. FuzzyLisp will help you in every step you make, but, as they say, there is no such a thing as a free lunch.

7.9 Fuzzy Logic in Motor Racing: Scoring in Regularity Rallies

Regularity rallies are a form of motor sport with strong tradition both in Europe and North America. The goal in this type of competition is not to complete an itinerary or a circuit in the shortest possible time but to complete a predefined route (usually with open traffic) on a precise time specified by the organization of the race. Let us imagine we are the organizers of a regularity rally where sponsors have asked us for some innovation in the regulations. Figure 7.18 represents a sort of ring, 18 km. in length, composed by three route sections. The first one starts in point A and follows a regional, low traffic road, until reaching point B. In this point the race goes along a national, heavy traffic road until reaching point C, where again another regional road leads to complete the ring at Finish (point A). The usual main rule for a race of this type is to estipulate an ideal time *t* for completing the route and then to time every participant car. Cars start the route not at the same time but with a time difference between them, e.g., one minute apart. After the race is over, every car C_i

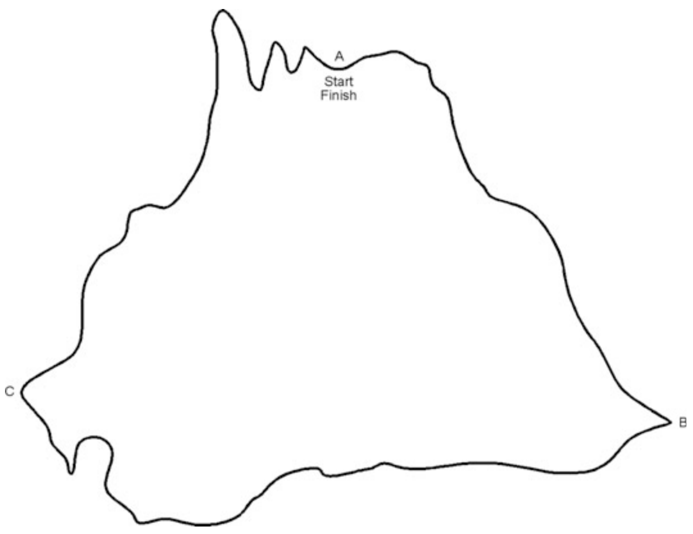

Fig. 7.18 Regularity rally track sections

will get a register time $t \pm dt_i$ and the car with the lower dt_i time will be the winner. For example, if we think about an average goal speed of 60 km/h, then the ring should be completed in exactly 18 min. Increasing deviations from this goal time will get decreasing ranking positions in the final classification.

A more complex rule can be derived from establishing two stages for the race in such a way that every car will complete the route in two lap times t_1, t_2. Then, the winner will be the car with the smaller difference $delta\text{-}time = abs(t_1 - t_2)$. As an example, if a car has registered lap times $t_1 = 17{:}57$ and $t_2 = 18{:}13$, (format is minutes: seconds) then its overall race time will be $delta\text{-}time = 16\ s$.

We are confident this is an improvement with respect to ordinary race regulations in regularity rallies, but after some reflection, we decide to go further. This time using fuzzy logic. First, we create two sharp rules in order to bind the model:

- Cars with lap time t1 or t2 < 16:00 will be eliminated
- Cars with lap time t1 or t2 > 20:00 will be eliminated

This is so because we do not want pilots achieving extremely high speeds (remember, roads are open to traffic in this type of competition). On the other hand slow cars are not the essence of motor sport, so an inferior time limit seems appropriate to apply. The design goal for our scoring model is to offer the pilots a variety of strategies to choose from, always having regularity in mind, but also offering them the possibility to opt for an aggressive strategy (higher speeds), a defensive, conservative strategy (lower speeds), or something in between. In any case, extreme aggressive strategies, and at a lesser extent, extreme defensive strategies, even with lap times well between 16 and 20 min, will have an implicit penalty. For example, if the pilot of a given car decides to go for lap times *t1* = 16:00:00 and *t2* = 16:00:00, then he will get 100 points, the maximum score. But if he gets, for example *t1 = 16:00:00* and *t2 = 16:00:05*, then he will get exactly zero points in the scoring. In some way the organizers are happy to have aggressive strategies in the race, but they give the teams the following warning: "if you want to run fast you must be extremely regular". The regulation completes with the following overall fuzzy rules:

- If you run really fast then you will need to be extremely regular to score the highest in the race.
- If you run relatively slow then you will need to be certainly regular to score high in the race.
- If you run at a moderate speed, then you will have the bigger tolerance with respect to delta-time, allowing great scoring in the race.

In order to create a fuzzy model for scoring the regularity rally, we shall create two linguistic variables as input (lap-time and delta-time) and an output linguistic variable named "points" for representing score. The membership functions and Lisp representation of the fuzzy set that make up these linguistic variables are shown in Tables 7.6a, b and c. Time units are expressed in seconds:

Table 7.6a Fuzzy sets for the linguistic variable "lap-time"

Fuzzy set label	Lisp representation of membership function
Very-quick	*(setq t1 '(very-quick 960 960 980 1000))*
Quick	*(setq t2 '(quick 980 1000 1040 1060))*
Slow	*(setq t3 '(slow 1040 1060 1100 1120))*
Rather-slow	*(setq t4 '(rather-slow 1100 1120 1160 1180))*
Very-slow	*(setq t5 '(very-slow 1160 1180 1200 1200))*

Table 7.6b Fuzzy sets for the linguistic variable "delta-time"

Fuzzy set label	Lisp representation of membership function
Delta-very-small	*(setq d1 '(delta-very-small 0 0 2 5))*
Delta-small	*(setq d2 '(delta-small 2 5 5 10))*
Delta-medium	*(setq d3 '(delta-medium 5 10 10 20))*
Delta-big	*(setq d4 '(delta-big 10 20 20 40))*
Delta-very-big	*(setq d5 '(delta-very-big 20 40 60 60))*

Table 7.6c Fuzzy sets for the output linguistic variable "points"

Fuzzy set label	Singleton at x_0	Lisp representation
Very-low	0	*(very-low 0)*
Low	25	*(low 25)*
Medium	50	*(medium 50)*
High	75	*(high 50)*
Very-high	100	*(very-high 100)*

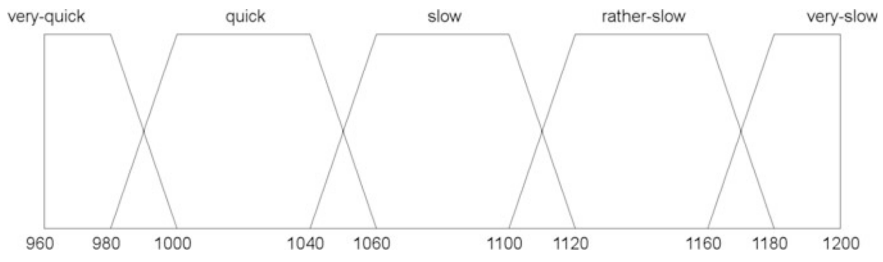

Fig. 7.19a Input linguistic variable lap-time, in seconds

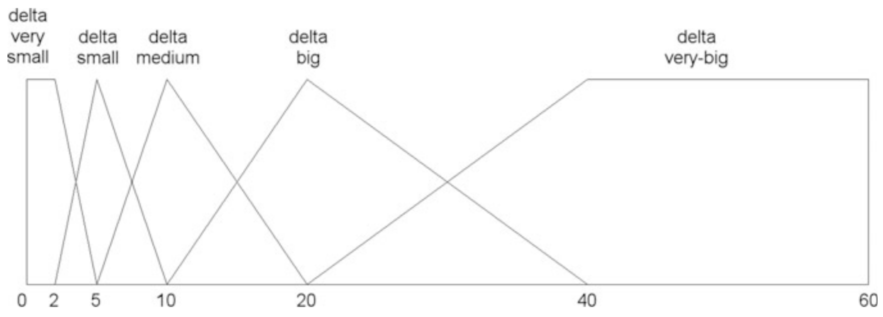

Fig. 7.19b Input linguistic variable delta-time, in seconds

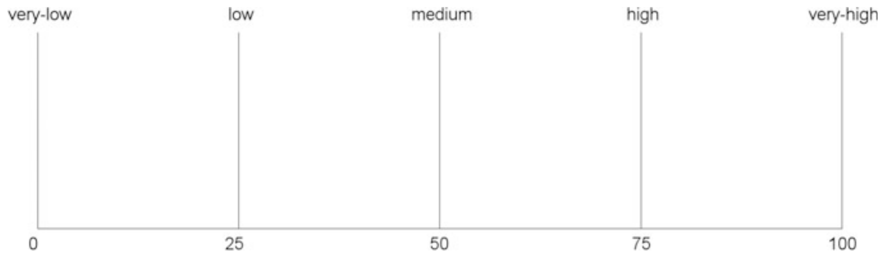

Fig. 7.19c Output linguistic variable representing score from 0 to 100

These fuzzy sets and linguistic variables are graphically shown in Figs. 7.19a, 7.19b and 7.19c.

The input variable "lap-time" has nothing special to comment, and, as can be seen, is composed by trapezoidal shaped fuzzy sets. The second input variable, "delta-time" is composed by fuzzy sets with both triangular and trapezoidal membership functions. The most important feature in the design of this linguistic variable is the non-uniform distribution of its fuzzy sets. These appear "compressed" towards the left because an important reason: As designers we wish the scoring model to be sensitive to differences of delta-time, and extremely sensitive when delta-time values are small and very small, hence the concentration of fuzzy sets towards the left of the linguistic variable. The rules for organizing the linguistic variables in the model are shown in Code 7-14:

```
;code 7-14
(setq rules-cars '((lap-time delta-time points)
    (very-quick delta-very-small very-high AND-product)
    (very-quick delta-small very-low AND-product)
    (very-quick delta-medium very-low AND-product)
    (very-quick delta-big very-low AND-product)
    (very-quick delta-very-big very-low AND-product)
    (quick delta-very-small high AND-product)
    (quick delta-small high AND-product)
    (quick delta-medium high AND-product)
    (quick delta-big medium AND-product)
    (quick delta-very-big very-low AND-product)

    (slow delta-very-small very-high AND-product)
    (slow delta-small very-high AND-product)
    (slow delta-medium very-high AND-product)
    (slow delta-big very-high AND-product)
    (slow delta-very-big medium AND-product)

    (rather-slow delta-very-small very-high AND-product)
    (rather-slow delta-small high AND-product)
    (rather-slow delta-medium high AND-product)
    (rather-slow delta-big high AND-product)
    (rather-slow delta-very-big very-low AND-product)

    (very-slow delta-very-small high AND-product)
    (very-slow delta-small high AND-product)
    (very-slow delta-medium medium AND-product)
    (very-slow delta-big low AND-product)
    (very-slow delta-very-big very-low AND-product))
)
```

Running the model by means of the FuzzyLisp functions *(fl-inference)* and *(fl-3d-mesh)* we obtain, after exporting the obtained data to a surface graphing application, the output surface from Fig. 7.20.

We can observe that small values of delta-time lead to the highest scores, and also an abrupt collapse in scoring when lap-time is very small and delta-time starts to increase (rightmost region of the Figure), as we desired on the initial design of the model. However, it is also easy to observe the presence of several terraces on the output surface. For obtaining a good classification in a sport competition this is not a desirable result because it may lead to many possible *ex-aequo* instances among the participants. As we commented in the previous section, the presence of

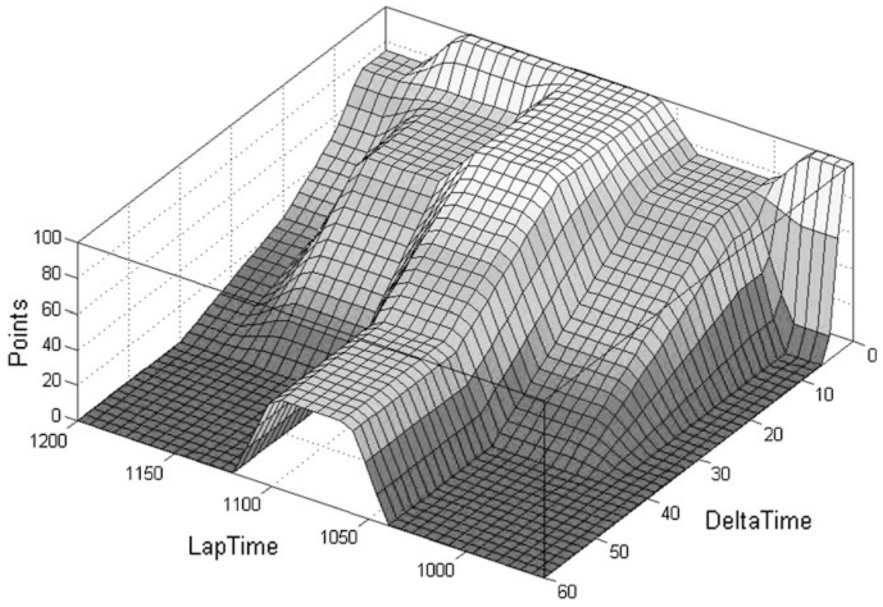

Fig. 7.20 Output surface of the scoring model

Table 7.7 Changing fuzzy sets for the linguistic variable “lap-time”

Fuzzy set label	Lisp representation of membership function
Very-quick	*(setq t1 ‘(very-quick 960 960 960 1200))*
Quick	*(setq t2 ‘(quick 960 1020 1020 1200))*
Slow	*(setq t3 ‘(slow 960 1080 1080 1200))*
Rather-slow	*(setq t4 ‘(rather-slow 960 1140 1140 1200))*
Very-slow	*(setq t5 ‘(very-slow 960 1200 1200 1200))*

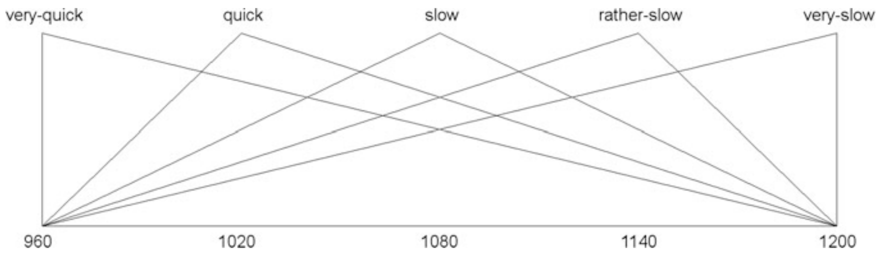

Fig. 7.21 Input linguistic variable lap-time, modified

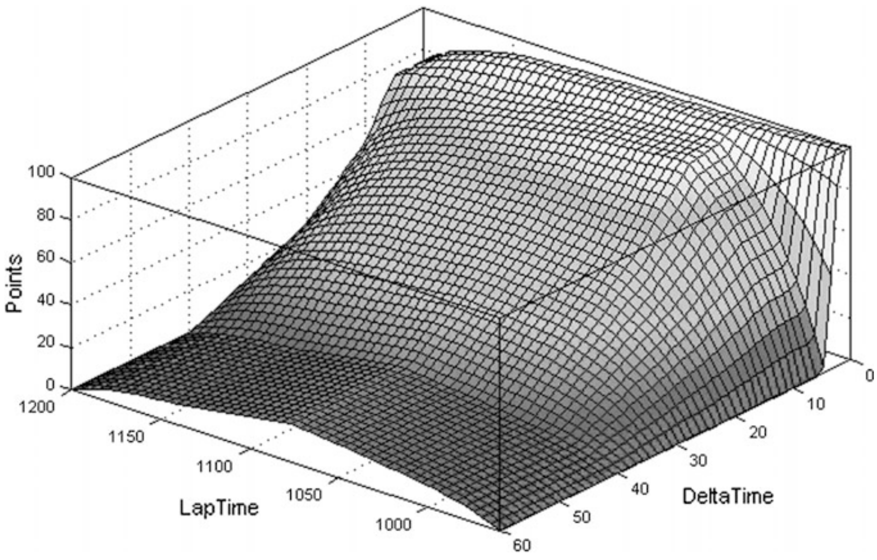

Fig. 7.22 Improved output surface of the scoring model

these terraces is due to the existence of trapezoidal membership functions in the linguistic variable lap-time. A workaround to this problem is to use triangles for the membership functions in the linguistic variable. Let us experiment what happens when using a maximum fuzziness linguistic variable where all the fuzzy sets at play share a common support. Table 7.7 shows the new linguistic variable lap-time.

The new sets are shown in Fig. 7.21.

Running again the model by means of the FuzzyLisp functions *(fl-inference)* and *(fl-3d-mesh)* we obtain a new output surface as shown in Fig. 7.22.

Now we can realize the improvement in smoothness we have got on the output surface. The overall design of the model is still there, but now we have got a more adequate result. In any case, the use of maximum fuzziness linguistic variables in fuzzy modeling sometimes generates too much "smoothness" and many times the use of standard triangular membership functions for the fuzzy sets in the model is desirable. The reader is again invited to play with the model, transforming it and observing the results. The complete Lisp code for this application is shown in Code 7-15:

```
;code 7-15
(load "my-path/fuzzy-lisp.lsp")
;fuzzy sets and linguistic variables definitions:
;lap time, all times in seconds
;For qualifying, time must be between 16 and 20 min
;that is, from 960 to 1200 s
(setq t1 '(very-quick 960.0 960.0 960.0 1020.0))
```

```
(setq t2 '(quick 960.0 1020.0 1020.0 1200.0))
(setq t3 '(slow 960.0 1080.0 1080.0 1200.0))
(setq t4 '(rather-slow 960.0 1140.0 1140.0 1200.0))
(setq t5 '(very-slow 960.0 1200.0 1200.0 1200.0))
(setq lap-time '(t1 t2 t3 t4 t5))

;delta-time, all times in seconds, from 0 s to 60 s
(setq d1 '(delta-very-small 0 0 2 5))
(setq d2 '(delta-small 2 5 5 10))
(setq d3 '(delta-medium 5 10 10 20))
(setq d4 '(delta-big 10 20 20 40))
(setq d5 '(delta-very-big 20 40 60 60))
(setq delta-time '(d1 d2 d3 d4 d5))

(setq points '(
   (very-high 100.0)
   (high 75.0)
   (medium 50.0)
   (low 25.0)
   (very-low 0.0))
)
;fuzzy rules section:

(setq rules-cars '((lap-time delta-time points)
   (very-quick delta-very-small very-high AND-product)
   (very-quick delta-small very-low AND-product)
   (very-quick delta-medium very-low AND-product)
   (very-quick delta-big very-low AND-product)
   (very-quick delta-very-big very-low AND-product)

   (quick delta-very-small high AND-product)
   (quick delta-small high AND-product)
   (quick delta-medium high AND-product)
   (quick delta-big medium AND-product)
   (quick delta-very-big very-low AND-product)

   (slow delta-very-small very-high AND-product)
   (slow delta-small very-high AND-product)
   (slow delta-medium very-high AND-product)
   (slow delta-big very-high AND-product)
   (slow delta-very-big medium AND-product)

   (rather-slow delta-very-small very-high AND-product)
   (rather-slow delta-small high AND-product)
   (rather-slow delta-medium high AND-product)
```

```
    (rather-slow delta-big high AND-product)
    (rather-slow delta-very-big very-low AND-product)

    (very-slow delta-very-small high AND-product)
    (very-slow delta-small high AND-product)
    (very-slow delta-medium medium AND-product)
    (very-slow delta-big low AND-product)
    (very-slow delta-very-big very-low AND-product))
)
```

Before finishing this section, let's imagine that it's raining the day of the motor race. Needless to say, lap times should be changed accordingly in order to allow the pilots to race safely. For obtaining a new definition of the linguistic variable "lap-time" we can write a function that internally calls the FuzzyLisp function *(fl-fuzzy-shift)*. Such a function is shown in Code 7-16:

```
;code 7-16
(define (shift-lap-time x)
    (setq t1 (fl-fuzzy-shift t1 x))
    (setq t2 (fl-fuzzy-shift t2 x))
    (setq t3 (fl-fuzzy-shift t3 x))
    (setq t4 (fl-fuzzy-shift t4 x))
    (setq t5 (fl-fuzzy-shift t5 x))
    true
)
```

Now, for shifting two minutes (120 s) the linguistic variable lap-time, we only need to type at the Lisp prompt:

> (shift-lap-time 120)
: true

and for testing the modified fuzzy-sets we can type:

> (fl-list-sets lap-time)
: (very-quick 1080 1080 1080 1140)
(quick 1080 1140 1140 1320)
(slow 1080 1200 1200 1320)
(rather-slow 1080 1260 1260 1320)
(very-slow 1080 1320 1320 1320)

As the reader can imagine, it is the responsibility of the organization of the race to give every racing team a copy of the output surfaces in the form of scoring tables (both in wet and dry conditions) to allow them to choose their race strategy well ahead of time. Table 7.5 is a simplified example of how to share a fuzzy model.

Usually more resolution is required, but this is easy to achieve by means of using the function *(fl-3d-mesh)*.

Incidentally, the use of function *(fl-3d-mesh)* allows the reader to use FuzzyLisp as a general purpose fuzzy modeling toolbox that produces neutral text files representing the knowledge embedded into the model. Later, if you use Java, C#, or any other programming language you only need to load the produced text files into your applications and then, by using bilinear interpolation, you automatically incorporate fuzzy-logic features in them.

7.10 FRBS Using Fuzzy Sets with Discrete Membership Functions

In Sects. 7.8 and 7.9 we have introduced two example fuzzy models based on a FuzzyLisp Standard Set Representation (FLSSR), that is, models whose fuzzy sets have either a triangular or trapezoidal characteristic function. However, in Chap. 6 we emphasized too on fuzzy sets with a discrete characteristic function (FLDSR). It would be a pity that after the effort of writing dedicated FuzzyLisp functions to this type of fuzzy sets we would not have a way to develop fuzzy rule based systems with them.

For such an undertaking we need only a slight variation of the functions *(fl-translate-rule)*, *(fl-translate-all-rules)*, *(fl-inference)* and *(fl-3d-mesh)* and then some changes in the Lisp structure of a fuzzy model. The names of the new FuzzyLisp functions for handling systems composed by discrete fuzzy sets are shown in the right column from Table 7.8.

As can be seen in the table, the simple addition of the letter "*d*" inside the name of the function signals its proper use. The Lisp source code of these functions is practically identical to their "continuous" counterparts, so we shall not show it here in order to save space. You can freely examine it after downloading the file fuzzylisp.lsp from the book's Web site. The important thing to remember is that both variations of the functions have exactly the same arguments as inputs and they produce the same structure of data in their output. Essentially their use is identical. There is only one restriction: You cannot mix continuous and discrete fuzzy sets for a given set of fuzzy rules. That is, if you decide to use an input linguistic variable

Table 7.8 FuzzyLisp functions for developing Fuzzy Rule Based Systems

Functions for FLSSR	Functions for FLDSR
(fl-translate-rule)	*(fl-dtranslate-rule)*
(fl-translate-all-rules)	*(fl-dtranslate-all-rules)*
(fl-inference)	*(fl-dinference)*
(fl-3d-mesh)	*(fl-3d-dmesh)*

Functions for dealing with continuous fuzzy sets are shown on the left column. Functions for discrete fuzzy sets are shown on the right column composed by fuzzy sets with discrete membership functions, then the other input variable must be composed also by discrete fuzzy sets. The use of singletons for the output share exactly the same structure in both cases.

Code 7-17 shows a variation of the model of an air-conditioner controller where bell-shaped fuzzy sets are used for the input linguistic variables lv-temperature and lv-delta-t. Please note the needed definitions f_i of every bell function and the now mandatory use of the function *(fl-discretize)* in order to follow an adequate FuzzyLisp sets definition. For example, the fuzzy set "optimal" needs first an adequate function definition, expressed by:

(setq f3 '(div (add 1.0 (cos (mul 0.2 pi (sub x 21.0)))) 2.0))

and then we only need to create the corresponding fuzzy set by means of the following expression:

(setq T3 (fl-discretize-fx 'optimal f3 20 16 26))

the membership function of the fuzzy set "optimal" (*T3*) is thus centered on x = 21, and its support is defined in the interval [16,26]. By means of the function *(fl-discretize)* we establish also the used resolution on the resulting fuzzy set, in this case n = 20. Note also that the fuzzy rules remain unchanged, as it happens also with the singletons for the consequents in the rules, being identical as the ones shown in Table 7.4c.

```
;code 7-17
(load "my-path/fuzzy-lisp.lsp")

;functions and definitions for first linguistic variable:
(setq f1 '(div (add 1.0 (cos (mul 0.0909 pi (sub x 0.0)))) 2.0))
(setq f2 '(div (add 1.0 (cos (mul 0.13333 pi (sub x 13.5))))
   2.0))
(setq f3 '(div (add 1.0 (cos (mul 0.2 pi (sub x 21.0)))) 2.0))
(setq f4 '(div (add 1.0 (cos (mul 0.13333 pi (sub x 28.5))))
   2.0))
(setq f5 '(div (add 1.0 (cos (mul 0.05263 pi (sub x 50.0))))
   2.0))

(setq T1 (fl-discretize-fx 'very-cold f1 20 0 11))
(setq T2 (fl-discretize-fx 'cold f2 20 6 21))
(setq T3 (fl-discretize-fx 'optimal f3 20 16 26))
(setq T4 (fl-discretize-fx 'hot f4 20 21 36))
(setq T5 (fl-discretize-fx 'very-hot f5 20 31 50))
(setq lv-temperature '(T1 T2 T3 T4 T5))

;functions and definitions for second linguistic variable:
```

```
(setq f6 '(div (add 1.0 (cos (mul 0.5 pi (sub x -2.0)))) 2.0))
(setq f7 '(div (add 1.0 (cos (mul 1.0 pi (sub x 0.0)))) 2.0))
(setq f8 '(div (add 1.0 (cos (mul 0.5 pi (sub x 2.0)))) 2.0))

(setq dT1 (fl-discretize-fx 'decreasing f6 20 -2.0 0.0))
(setq dT2 (fl-discretize-fx 'no-change f7 20 -1.0 1.0))
(setq dT3 (fl-discretize-fx 'increasing f8 20 0.0 2.0))
(setq lv-delta-t '(dT1 dT2 dT3))

(setq AFT '(
     (cold-strong -100)
     (cold-medium -50)
     (stop 0)
     (heat-medium 50)
     (heat-strong 100))
)

(setq rules-controller '((lv-temperature lv-delta-t AFT)
   (very-cold decreasing heat-strong AND-product)
   (very-cold no-change heat-strong AND-product)
   (very-cold increasing heat-strong AND-product)

   (cold decreasing heat-strong AND-product)
   (cold no-change heat-medium AND-product)
   (cold increasing heat-medium AND-product)

   (optimal decreasing heat-medium AND-product)
   (optimal no-change stop AND-product)
   (optimal increasing cold-medium AND-product)

   (hot decreasing cold-medium AND-product)
   (hot no-change cold-medium AND-product)
   (hot increasing cold-strong AND-product)

   (very-hot decreasing cold-strong AND-product)
   (very-hot no-change cold-strong AND-product)
   (very-hot increasing cold-strong AND-product))
)
```

For running the model when, for example, temperature is 21 °C and temperature variation is 0.5 °C/min, we only need to type:

> (fl-dinference rules-controller 21 0.5)
: *-11.32704597*

Table 7.9 Airflow temperatures AFT obtained as the crisp result of a variation of the fuzzy air-conditioner controller using *discrete* membership functions

	0	7.5	13.5	17.5	21	22.5	28.5	32.5	38	50
−2	100	100	100	84.26	49.99	39.26	−50	−51.80	−100	−100
−1.5	100	100	100	84.26	49.99	39.26	−50	−51.80	−100	−100
−1.0	100	100	100	84.26	49.99	39.26	−50	−51.80	−100	−100
−0.5	100	88.67	61.33	45.58	11.33	4.74	−50	−51.80	−100	−100
0	100	85.35	50	34.26	0	−5.37	−50	−51.80	−100	−100
0.5	100	85.35	50	30.69	−11.33	−16.69	−61.33	−62.72	−100	−100
1.0	100	85.35	50	18.51	−49.99	−55.37	−100	−100	−100	−100
1.5	100	85.35	50	18.51	−49.99	−55.37	−100	−100	−100	−100
2.0	100	85.35	50	18.51	−49.99	−55.37	−100	−100	−100	−100

Input temperatures are shown in the first row. Temperature variation is shown in the first column

Looking into the details at the model is easy with the function *(fl-dtranslate-all-rules):*

```
> (fl-dtranslate-all-rules rules-controller 21 0.5)
: ((0 0 0 0) (0 0.5 0 0) (0 0.1464466094 0 0) (0 0 0 0) (0 0.5 0) (0 0.1464466094
0 0) (1 0 0 0) (1 0.5 0.5 0) (1 0.1464466094 0.1464466094 -7.322330472) (0 0 0 0)
(0 0.5 0 0) (0 0.1464466094 0 0) (0 0 0 0) (0 0.5 0 0) (0 0.1464466094 0 0))
```

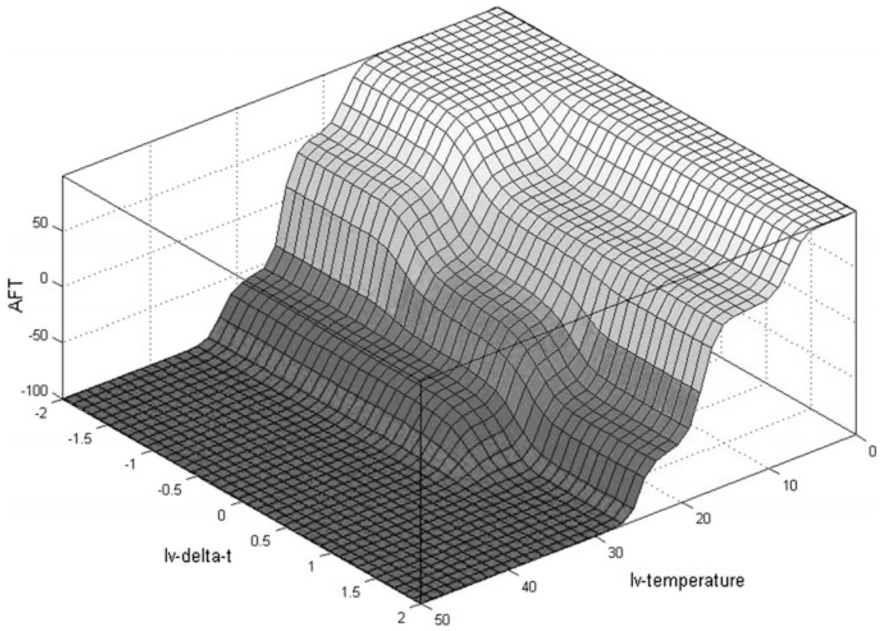

Fig. 7.23 Output surface of the modified fuzzy air-conditioner controller

In Table 7.9 we can have a first look at the behavior of the model. It offers different results when comparing it with the previous model whose linguistic variables are composed by fuzzy sets with triangular or trapezoidal membership functions. This is so because an important reason: while it is easy to replace triangles by bell-shaped functions, trapeziums are more difficult, especially when their support is large with respect to the whole range of the linguistic variable they belong. Due to this, the supports of the fuzzy sets belonging to the linguistic variable lv-delta-t have been more equally spaced.

Figure 7.23 shows the surface output of the model. If we compare it to the surface output shown in Fig. 7.17 we shall note immediately the effect caused by the new design of the output linguistic variable lv-delta-t, but at the same time it is easy to realize the resemblance between the two outputs, showing a familiar appearance. Besides having, as stated, identical singletons for the consequents and similar fuzzy sets in the input linguistic variable lv-temperature, the rules belonging to the knowledge database are exactly the same. Models based on bell shaped fuzzy sets exhibit a smoother output surface when compared to triangular and trapezoidal shaped fuzzy sets, but not a big difference of character is observed between them. On the other hand, creating FRBS with fuzzy sets based on discrete membership functions require more code in FuzzyLisp, and more internal calculations in general. In the rest of the book we shall use triangular and trapezoidal membership functions in the application models.

7.11 As a Summary

This chapter has allowed us to develop a conceptual transition from fuzzy sets theory to fuzzy logic. As already stated, this transition has been really smooth due to the existent isomorphism between membership degree to a (fuzzy) set and the truth-value of a (fuzzy) proposition. The following paragraphs summarize the main ideas exposed on this chapter:

- In general terms, logic is a structured pattern of thinking that makes it possible to arrive to new knowledge from previous established information. In traditional logic, a proposition is natural language expressing an assertion that can be either true or false, but not true and false at the same time.
- Simple propositions take the form “x is P” where x is called the subject of the proposition and P is called the predicate of the proposition, usually expressing a property or feature of x. After a proposition is evaluated, a truth-value results from the evaluation. In classic logic there are only two possible results: either “true” or “false”.
- In compound propositions their truth value results from the truth values of every simple proposition taking part in it and the way they are connected, that is, from the type of logical connective, or logical operator, used for establishing the link

between simple propositions. The main logical operators are "conjunction, "disjunction", "negation" and "implication".

- Conjunction: Given two propositions p, q, we call their conjunction to the compound proposition "p and q", denoting them by the logical expression "p ∧ q". This compound proposition is true if and only if both p and q propositions are true.
- Disjunction: Given two prepositions p, q, we call their disjunction to the compound proposition "p or q", denoting them by the logical expression "p ∨ q". This compound proposition is true when at least one of both propositions, p or q, is true, or, obviously, when both are true. For being false it needs that both propositions are false.
- Negation: From any proposition p we shall call "negation of p" to the proposition "not p", denoting it by the expression "¬p". The proposition ¬p will be true when p is false and will be false when p is true.
- Implication: Also known as "conditional proposition", it is expressed by the sentence "p implies q", and it is usually denoted by the expression "p → q". When p → q, it means that the conditional proposition is true except when the proposition p is true and q is false. In other words, if p (also named the "antecedent" in implication) is true, then q (the consequent) must be also true to fulfill a conditional proposition.
- There is an important relationship between set theory and logic in such a way that an isomorphism do exist between the logical operators "and", "or", "not", "implication" and the set operations "intersection", "union", "complement" and "inclusion", respectively. In practical terms this means that all the material exposed in Sect. 5.2 of this book about the classic theory of sets is of entire application in the field of logic after the simple substitution of the concept of membership or not membership of an element x to a set A and the concept of true or false when referring to a proposition p. Naturally, this isomorphism extends also to fuzzy sets and fuzzy logic.
- A fuzzy proposition is natural language declaring an assertion that has implicitly associated a truth-value expressed by a real number in the closed interval [0,1]. Simple fuzzy propositions have the following structure: x is P. It should be noted that the subject x of a fuzzy proposition P is usually not fuzzy. What distinguishes a classic proposition from a fuzzy proposition is the characterization of the predicate P. The fuzziness of P is what generates fuzzy propositions. In fact, fuzzy propositions result from the existing isomorphism between fuzzy sets and fuzzy predicates. In order to evaluate a simple fuzzy proposition p, we need to know the definition of its predicate, which is generally given by the definition of a fuzzy set.
- The conjunction of two fuzzy propositions p, q, represented by $p \wedge q$, is the result of the minimum truth value of both p and q, that is: $Tv(p \wedge q) = min(\mu_A(x), \mu_B(y))$, where A and B are the fuzzy sets representing the predicates associated to the fuzzy propositions p and q, respectively, while x and y are the feature values associated to their respective subjects.

- The disjunction of two fuzzy propositions *p*, *q*, represented by $p \vee q$, is the result of the maximum truth value of both *p* and *q,* that is: $Tv(p \vee q) = max(\mu_A(x), \mu_B(y))$, where again *A* and *B* are the fuzzy sets representing the predicates associated to the fuzzy propositions *p* and *q*, respectively, while *x* and *y* are the feature values associated to their respective subjects.
- The negation of a fuzzy proposition *p*, denoted by $\neg p$, is the result of subtracting the truth-value of *p* from one, that is: $Tv(\neg p) = 1 - (\mu_A(x))$.
- The fuzzy conditional proposition, denoted by $p \rightarrow q$ is the most complex one from the four logical exposed connectives. Its truth-value is given by the expression $\mathrm{Tv}(p \rightarrow q) = \min(1, 1 + \mu_B(y) - \mu_A(x))$. As it happens with the previous fuzzy connectives, *A* and *B* are the fuzzy sets representing the predicates associated to the fuzzy propositions *p* and *q*, respectively, while x and y are the feature values associated to their respective subjects.
- Fuzzy hedges are linguistic modifiers applied to fuzzy predicates, so they also modify fuzzy sets related to fuzzy predicates. As a consequence, fuzzy hedges affect fuzzy propositions and then their truth-values, too. In general, a fuzzy hedge H can be represented by an unary operator on the closed interval [0,1] in such a way that a fuzzy proposition of the type "*x* is *P*" converts into: *x* is H*P*. If *A* is a fuzzy set associated to a fuzzy predicate *P*, then we have: $\mathrm{H}A = \mathrm{H}(\mu_A(x))$. The most used fuzzy hedges are the linguistic modifiers H_1, "very" and H_2,"fairly", defined respectively by the following expressions: $H_1 : H_1(\mu_A(x)) = (\mu_A(x))^2$ and $H_2 : H_2(\mu_A(x)) = (\mu_A(x))^{1/2}$.
- When n tends to infinity, the n-iteration of the hedge VERY on a fuzzy number *A* produces another special fuzzy set *A'* whose nucleus and support is established exactly at x_0, where its membership degree equals one, that is: $VERY^n(A) \rightarrow (x_0, \mu(x_0) = 1.0)$. This happens also using the FuzzyLisp function *(fl-expand-contract-set)* when its second argument k equals zero. These peculiar fuzzy sets, named singletons, play an important role in fuzzy logic.
- Fuzzy compound expressions of the type "if x is *A* and y is *B* then z is *C*" where *A, B, C* are fuzzy sets and x, y and z are crisp numerical values (z being the only data unknown in the expression) are certainly one of the most useful and practical ones in fuzzy logic theory.
- Fuzzy Rule-Based Systems, FRBS, are logical constructions that bring together several fuzzy-logic based processes and structures. The most usual structure in a FRBS is a set of fuzzy compound propositions of the type "if x is *A* and y is *B* then z is *C*", named "rules" or "expert rules". A FRBS is a system that receives information (crisp input), processes it and then produces a crisp result. The crisp output represents a numerical magnitude that is initially unknown but can be calculated by the inner workings of the fuzzy system.
- A Knowledge Database consists of several linguistic variables and a set of fuzzy propositions (expert rules). Traditionally in fuzzy logic theory only the collection of expert fuzzy rules makes up a knowledge database, but this definition is incomplete because the expert knowledge does not come only from the fuzzy rules in the system, but also from the meaning of the fuzzy sets that make up the linguistic variables of the system.

- The inference engine's task in a fuzzy system is, simply put, to make inferences. Until the inference engine starts to work, the only things we have in a fuzzy system are membership degrees, several fuzzy sets and expert fuzzy rules "floating up in the air". The inference engine is the glue that interacts with all these elements.
- The procedure for obtaining a crisp output from an output set of membership degrees is called "Defuzzification". There are several defuzzification methods in fuzzy logic theory. The "Centroid Method" has a strong advantage that dwells from its intuitive formulation: it is very easy to understand. However it has two important disadvantages. First, it can become tedious to calculate, and second, the Centroid Method is unable to provide a crisp output value *z* at the extremes of the linguistic variable where the output fuzzy sets are defined. Another well-known defuzzification method is the so named "Weighted average method". In general, for *n* membership degrees from a fuzzy output the weighted average method uses the expression $z = \Sigma(x_i\mu_i(z))/\Sigma\mu_i(z)$ for calculating the crisp output result.
- As previously suggested, a singleton *S* is a special fuzzy set whose membership function is equal to zero except at a point x_0, where its membership degree is exactly equal to one. In FuzzyLisp they are represented by expressions of the type *(singleton-label* x_0*)*. Since every singleton can be seen as a geometrical spike resting fixed on a real point x_i, we can apply the weighted average method of defuzzification for singletons in order to obtain the crisp output of a FRBS. Singleton defuzzification is the standard defuzzification method used by FuzzyLisp.
- Along the chapter new FuzzyLisp functions have been introduced for developing practical fuzzy logic models. The functions *(fl-translate-rule)*, *(fl-translate-all-rules)*, *(fl-inference)* and *(fl-3d-mesh)* provide powerful management of FRBS when the fuzzy sets contained in their knowledge database are formed by triangular or trapezoidal membership functions. An alternate set of functions named *(fl-dtranslate-rule)*, *(fl-dtranslate-all-rules)*, *(fl-dinference)* and *(fl-3d-dmesh)* are offered for dealing with FRBS in those cases when the fuzzy sets contained in their knowledge database have discrete characteristic functions.
- Functions *(fl-3d-mesh)* and *(fl-3d-dmesh)* can be understood as a bridge between FuzzyLisp and any other programming language. The text files produced by these functions can be loaded into any software project and then, by using bilinear interpolation, all the expert knowledge from the previously developed fuzzy models can be incorporated in those software projects. As an example, this opens the possibility for using FuzzyLisp as a software tool for developing intelligent apps for smartphones and other mobile devices.

After all the theory and fundamental FuzzyLisp functions for dealing with fuzzy logic inferences and FRBS management have been exposed, we have dedicated a good number of pages to develop two practical examples of fuzzy models: A fuzzy air-conditioner controller and a fuzzy model for scoring a motor sport regularity rally. These models are simple but have served us well for discussing their construction and for exploring some approaches of fuzzy model design. We shall continue to explore the development of more sophisticated fuzzy models in the last chapter of this book.

References

Bojadziev, G., Bojadziev, M.: Fuzzy Logic for Business, Finance and Management. World Scientific (1999)

Klir, G., Yuan, B.: Fuzzy Sets and Fuzzy Logic: Theory and Applications. Prentice Hall, Englewood Cliffs, NJ (1995)

Kranz, G.: Failure is Not an Option: Mission Control from Mercury to Apollo 13 and Beyond. Simon & Schuster, New York (2009)

Kundu, S., Chen, J.: Fuzzy logic or Lukasiewicz Logic: A clarification. Lecture Notes in Computer Science, vol 869 (1994)

McFerran, D.: A Syllogism Playbook: Using PLN for Deductive Logic. McFerran (2014)

Priest, G.: Logic: A Very Short Introduction. Oxford University Press, Oxford (2001)

Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach, 3rd edn. Prentice-Hall, Englewood Cliffs, NJ (2009)