Appendix A
NewLisp Versus Common Lisp

NewLisp has been used as the computing vehicle for developing this book. Being a Lisp dialect, it contains the overall Lisp programming paradigm and shares the standard Lisp features. However, as it happens with languages and dialects, it differs in details. The most important ones are discussed in this appendix when compared with Common Lisp. For further details, you can consult the online NewLisp documentation and/or the documentation of your Common Lisp compiler.

A.1 Arithmetic

As we saw in Chap. 1, NewLisp has different functions for distinguishing between integer and real arithmetic operations. When handling integers, the functions are +, -, *, and /, so for example: 
\( (+2 \ 7) \rightarrow 9 \), \( (-4 \ 3) \rightarrow 1 \), \( (*3 \ 4) \rightarrow 12 \), \( (/12 \ 4) \rightarrow 3 \) and so on.

On the other hand, if we wish to operate with real numbers, we need to use the functions \((add)\), \((sub)\), \((mul)\) and \((div)\), and then, for example: \( (add \ 3.1 \ 6.9) \rightarrow 10 \), \( (sub \ 7.2 \ 3) \rightarrow 4 \), \( (mul \ 2.5 \ 4.3) \rightarrow 10.75 \) and \( (div \ 7 \ 4) \rightarrow 1.75 \). Common Lisp uses only the +, -, * and / functions, but does it in a different way. It reads the supplies data first, and then decide if the arithmetic operation involves integer numbers, real numbers, or a combination of both. For example, \( (+2 \ 7) \rightarrow 9 \), \( (-4 \ 3) \rightarrow 1 \), \( (*3 \ 4) \rightarrow 12 \), \( (/12 \ 4) \rightarrow 3 \), as in NewLisp, but \( (+2 \ 7.5) \rightarrow 9.5 \), \( (-5 \ 1.5) \rightarrow 3.5 \), \( (*2 \ 2.2) \rightarrow 4.4 \) and \( (/2.2 \ 2.2) \rightarrow 1.0 \), but, for example:

CL> (/ 1 3)
: 1/3

When using integers in a division resulting into a rational number, Common Lisp returns fractions, thus, it never loses precision. If we wish that Common Lisp returns the decimal expansion of a rational number we shall need to tell it explicitly, writing at least one of the numbers in decimal format:

CL> (/ 1 3.0)
: 0.33333334
A practical and useful rule of thumb is to always use decimal points in Common Lisp when handling real numbers, as it happens in Fuzzy Logic. For example, for describing a set of young people, you can write in NewLisp:

```
(setq age1 ‘(young 15 25 25 35))
```

but in Common Lisp is advisable to always write:

```
(setq age1 ‘(young 15.0 25.0 25.0 35.0))
```

### A.2 Fundamental Lisp Functions

As we saw in Chap. 2, there are many Lisp functions, but the number of functions that build up the core of the language are relatively small. Table A.1 shows some fundamental Lisp functions and their equivalence in both versions of Lisp.

From these functions, some of them show differences in their behavior. After writing, for example, `(setq a ‘(x y z))`, then in NewLisp, `(last a) → z`, but in Common Lisp, `(last a) → (z)`, that is, instead of returning the last element as an atom, Common Lisp returns the last element as a list. However, for both Lisp

<table>
<thead>
<tr>
<th>NewLisp function</th>
<th>Common Lisp equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>(quote)</code></td>
<td><code>(quote)</code></td>
</tr>
<tr>
<td><code>(eval)</code></td>
<td><code>(eval)</code></td>
</tr>
<tr>
<td><code>(atom?)</code></td>
<td><code>(atom)</code></td>
</tr>
<tr>
<td><code>(symbol?)</code></td>
<td><code>(symbolp)</code></td>
</tr>
<tr>
<td><code>(number?)</code></td>
<td><code>(numberp)</code></td>
</tr>
<tr>
<td><code>(list?)</code></td>
<td><code>(lisp)</code></td>
</tr>
<tr>
<td><code>(first)</code></td>
<td><code>(car)</code></td>
</tr>
<tr>
<td><code>(rest)</code></td>
<td><code>(cdr)</code></td>
</tr>
<tr>
<td><code>(nth)</code></td>
<td><code>(nth)</code></td>
</tr>
<tr>
<td><code>(last)</code></td>
<td><code>(last)*</code></td>
</tr>
<tr>
<td><code>(length)</code></td>
<td><code>(length)</code></td>
</tr>
<tr>
<td><code>(cons)</code></td>
<td><code>(cons)*</code></td>
</tr>
<tr>
<td><code>(list)</code></td>
<td><code>(list)</code></td>
</tr>
<tr>
<td><code>(assoc)</code></td>
<td><code>(assoc)</code></td>
</tr>
<tr>
<td><code>(append)</code></td>
<td><code>(append)</code></td>
</tr>
<tr>
<td><code>(reverse)</code></td>
<td><code>(reverse)</code></td>
</tr>
<tr>
<td><code>(pop)</code></td>
<td><code>(pop)</code></td>
</tr>
<tr>
<td><code>(rotate)</code></td>
<td><code>n/a</code></td>
</tr>
<tr>
<td><code>(random)</code></td>
<td><code>(random)*</code></td>
</tr>
</tbody>
</table>

Those with different behavior are marked with an asterisk.
dialects: \((nth\ 2\ a) \rightarrow z\). This suggest the following code in Common Lisp for obtaining the same behavior:

\[
\begin{align*}
&\text{(defun last1 (list)} \\
&\quad (\text{nth } (- (\text{length list}) 1) \text{ list}))
\end{align*}
\]

or, alternatively:

\[
\begin{align*}
&\text{(defun last1 (list)} \\
&\quad (\text{car (last list)}))
\end{align*}
\]

In both cases, now from Common Lisp, \((last1\ a) \rightarrow z\).

The case of \((\text{cons})\) shows also a difference between dialects. As an example, in NewLisp \((\text{cons} 'a\ b) \rightarrow (a\ b)\), while in Common Lisp \((\text{cons} 'a\ b) \rightarrow (a \cdot b)\) that is, in Common Lisp we get a dotted pair while in NewLisp we get a standard list. This must be handled with care. For example, after writing the following in Common Lisp: \((\text{setq} x (\text{cons} 'a 'b)) \rightarrow (a \cdot b)\), and then: \((\text{car} x) \rightarrow a, (\text{cdr} x) \rightarrow b, \text{but (last x) } \rightarrow (a \cdot b)\). On the other hand, the Lisp expression \((\text{cons} '(\text{this}\ \text{is})\ '(\text{not}\ \text{hard})) \rightarrow ((\text{this}\ \text{is})\ \text{not}\ \text{hard})\) produces the same result in both dialects.

The NewLisp function \((\text{rotate})\) is not implemented in standard Common Lisp, but here is an example for moving the first element of a list to its end:

\[
\begin{align*}
&\text{(defun left-rotate (list)} \\
&\quad (\text{append (rest list) (list (first list))}))
\end{align*}
\]

and now, remembering the previous assignment \((\text{setq} a '(x\ y\ z))\), we have \((\text{left-rotate}\ a) \rightarrow (y\ z\ x)\).

Finally, the function \((\text{random})\) is differently implemented in both dialects, but the goal is identical, that is, to produce a random number. In NewLisp we have, for example: \((\text{random}) \rightarrow 0.7830992238\), that is, it produces a random real number between 0 and 1, and no parameter is needed to call the function. On the other hand, in Common Lisp we need to pass an integer parameter to the function and then the language will return an integer between 0 and the integer passed as argument. For example, for generating a number between 0 and 100 we only need to write: \((\text{random} 100) \rightarrow 67\).

### A.3 Defining Functions

From Chap. 3 we remember that the Basic anatomy of a NewLisp function has the following structure:

\[
(\text{define (function-name arguments)} \\
\quad (\text{lisp-expression}_i))
\]
In Common Lisp, it takes the following form:

```lisp
(defun function-name (arguments)
  (lisp-expression)
)
```

As an example, in Code 3-3 we introduced a function for calculating the Body Mass Index. In Common Lisp it would be as follows:

```lisp
(defun BMI(mass height)
  (/ mass (* height height))
)
```

Interesting things happen when we need to use local variables in functions. The following code is Code 3-9 expressed in Common Lisp.

```lisp
(defun bmi2 (mass height)
  (let
    ((result) (advice))
    (setq result (/ mass (* height height)))

    (if (< result 18.5)
      (setq advice "You are excessively thin")
    )
    (if (and (>= result 18.5) (< result 24.9))
      (setq advice "Congrats, your weight is ok")
    )
    (if (and (>= result 24.9) (< result 29.9))
      (setq advice "You should try some diet and exercise because you have some overweight")
    )
    (if (>= result 29.9)
      (setq advice "Warning, you suffer obesity. Speak with your doctor")
    )
    advice
  )
)
```

As you can observe, the internal variables `result` and `advice` are declared by means of the Common Lisp reserved word `let`, telling the language that the symbols located inside its matching parenthesis are local variables. Testing the function, we have, for example:
CL > (BMI2 75 1.80)
: "Congrats, your weight is ok"

Speaking about variables, we must add that global variables must be declared explicitly in Common Lisp by means of the keyword defparameter. As an example, if we wish to create a global linguistic variable glucose for describing glucose levels in blood in mg/dl, we could write:

```
(defparameter gl_1 '(Very-low 60 (60 1) (60 1) 82))
(defparameter gl_2 '(Rathar-low 60 (87.5 1) (87.5 1) 115))
(defparameter gl_3 '(Low 60 (115 1) (115 1) 165))
(defparameter gl_4 '(Medium-low 115 (165 1) (165 1) 219.5))
(defparameter gl_5 '(Medium-high 165 (219.5 1) (219.5 1) 274))
(defparameter gl_6 '(High 219.5 (300 1) (300 1) 300))
(defparameter glucose '(gl_1 gl_2 gl_3 gl_4 gl_5 gl_6))
```

**A.4 Iteration**

Along this book we have extensively used the while NewLisp keyword in order to help people with previous experience in other programming languages where “while loops” are always used. However, in Common Lisp the do macro is the fundamental iteration operator and the while keyword is usually not included in the language. The following code implements while loops in Common Lisp:

```
; a definition of while in Common Lisp
(defmacro while (test &rest body)
  '(do ()
    ((not,test))
    ,@body)
)
```

Now, let us rewrite, for example Code 3-17 in Common Lisp:

```
(defun my-find (atm lst)
  (let
    ((n) (i) (aux) (rest))
  )

  (setq n (length lst))
  (setq i 0)
  (setq result nil)
```
(while (< i n)
   (progn
      (setq aux (nth i lst))
      (if (eq aux atm)
          (setq result (cons i atm))
          )
      (setq i (+ 1 i))
   );progn end
);while end
(list (car result) (cdr result))
)

CL> (my-find 'RED '(YELLOW RED GREEN BLUE WHITE))
: (1 RED)

Several additional things must be noted in this Common Lisp function:

- The (eq) function. In NewLisp we use the comparator operator “=” for comparing any type of values. In Common Lisp the comparator operator “=” is only used for comparing numerical values. Everything else must be compared with the (eq) function.
- The use of (progn): In NewLisp we use (begin). In Common Lisp, (progn) is used instead.
- The last line differs between both languages: In NewLisp we used simply result. However in CommonLisp we need to use the expression (list (car result) (cdr result)) in order to avoid a dotted pair as a function result since it would return (1 . RED) otherwise.

### A.5 nil and true

In NewLisp nil and true are Boolean constants. In Common Lisp nil has an additional role as a list terminator. For example, in NewLisp we have: (cons 'x nil) → (x nil), while in Common Lisp: (cons 'x nil) → (x).

As usually, “practice makes perfect”, so the reader used to NewLisp will need some days in order to get used to Common Lisp. On the other hand, experienced Common Lisp programmers will find the NewLisp code in this book easy to follow and I suspect they will soon rewrite the most important functions for adapting them to their programming styles.

As additional resources, the interested reader can move to Common Lisp using the excellent introductory book “Common Lisp: A Gentle Introduction to Symbolic Computation”, by David Touretzky, at about the same level of the introduction to NewLisp in this book. At a higher level, the de-facto standard in Common Lisp is the great book “ANSI Common Lisp”, by Paul Graham.
Appendix B
Glossary of FuzzyLisp Functions

This Appendix alphabetically shows all the functions that build up FuzzyLisp. For every function the information is structured as follows:

- Name of the function: Gives the name of the function.
- Explanation: Explains how the function works and what the function returns.
- Syntax: Offers the syntax of the function, that is, its name and all its required arguments.
- FuzzyLisp representation: either FLSSR (FuzzyLisp Standard Set Representation) or FLDSR (FuzzyLisp Discrete Set Representation).
- Example: Shows a practical example that helps to put the function in context.
- Source code number: Gives the source code number in X-Y format, where X is the number of the chapter and Y is the number of the code inside the chapter for quickly locating the source code in this book.

**fl-3d-mesh**

Explanation: This function creates an ASCII output file in comma-separated values format (CSV) where every line adopts the following structure: $x_i$, $y_i$, $z_i$. Both $x_i$, $y_i$ are input crisp values from the universes of discourse of their respective linguistic variables from a Fuzzy Rule Based System (FRBS). On the other hand, $z_i$ is the inferred value from every possible pair ($x_i$, $y_i$). The output file is in fact a discretized geometrical 3D mesh.

Syntax: $(\text{fl-3d-mesh namefile set-of-rules nx ny})$

- *namefile*: file name for storing the output data on the computer’s hard disk.
- *set-of-rules*: The complete set of expert rules of a FRBS expressed in list format.
• **nx**: Resolution of the 3D mesh over the x-axis.
• **ny**: Resolution of the 3D mesh over the y-axis.

**FuzzyLisp representation**: FLSSR

Example(s): (fl-3d-mesh “air-conditioner-controller.csv” rules-controller 20 20) → Writing 3Dmesh... 3Dmesh written to file

Source code number: 7-13.

Note: The FuzzyLisp function (fl-3d-dmesh) is suited to deal with FRBS where input linguistic variables are composed by fuzzy sets with discrete membership functions (FLDSR).

**fl-alpha-cut**

Explanation: (fl-alpha-cut) scans a trapezoidal or triangular membership function from left to right and returns the obtained alpha-cut alpha as a list, including the name of the original fuzzy set.

Syntax: (fl-alpha-cut fset alpha)

• **fset**: a list representing a fuzzy set with a continuous membership function, either a triangle or a trapezium.
• **alpha**: a real number representing the horizontal line \( y = \alpha \) for obtaining an alpha-cut. It is required that \( \alpha \in [0,1] \).

**FuzzyLisp representation**: FLSSR

Example(s): (fl-alpha-cut '(B1 7 10 12 15) 0.7) → (B1 9.1 12.9)

Source code number: 6-4.

**fl-belongs?**

Explanation: (fl-belongs?) returns true if a crisp value \( x \) defined on the real axis belongs to the fuzzy set fset, else returns nil.

Syntax: (fl-belongs? fset x)

• **fset**: a list representing a fuzzy set with a continuous membership function, either a triangle or a trapezium.
• **x**: a real number.

**FuzzyLisp representation**: FLSSR

Example(s): (fl-belongs '(medium 10.0 20.0 30.0 40.0) 23.0) → true; (fl-belongs '(medium 10.0 20.0 30.0 40.0) 3.0) → nil

Source code number: 6-1.

**fl-belongs2?**

Explanation: (fl-belongs2?) is a sort of mix of the functions (fl-belongs?) and (fl-set-membership?). If the crisp value \( x \) is contained in the support of fset it returns the membership degree of \( x \) to fset, otherwise, it returns nil.
Syntax: \((\text{fl-belong2? } \text{fset } x)\)
- \(\text{fset}\): a list representing a fuzzy set with a continuous membership function, either a triangle or a trapezium.
- \(x\): a real number.

FuzzyLisp representation: FLSSR
Example(s): After assigning, e.g., \((\text{setq } S \text{ '(medium } 10 \ 20 \ 30 \ 40))\), then \((\text{fl-belong2? } S \ 22) → \text{(medium } 1)\), \((\text{fl-belong2? } S \ 100) → \text{nil}\)
Source code number: 6-3.

\textbf{fl-db-new-field}
Explanation: This function creates a new field in a CSV format database. The new field contains the fuzzified values from an already existing numerical field. The fuzzification is obtained by means of a given fuzzy set.
Syntax: \((\text{fl-db-new-field } \text{lst } \text{sharp-field } \text{fz-field } \text{fset } \text{mode})\)
- \(\text{lst}\): list containing an entire database.
- \(\text{sharp-field}\): a string representing the name of a numerical field in the database.
- \(\text{fz-field}\): a string for naming the new field to create.
- \(\text{fset}\): a list representing a fuzzy set, either in a FuzzyLisp standard set representation or in a discrete set representation.
- \(\text{mode}\): an integer. A value of 1 means that \text{fset} has a FLSSR. A value of 2 means a FLDSR.

FuzzyLisp representation: FLSSR/FLDSR
Example(s): assuming the fuzzy set \(BM\) defined by \((\text{setq } BM \text{'(bright-magnitude } -1 \ -1 \ 3 \ 5))\) and that all the rest of function parameters have been correctly initialized, the function call \((\text{setq messier (fl-db-new-field messier "Magnitude" } "fz-magnitude" BM 1))\) creates a new field named “fz-magnitude” where all the numerical values from the field “Magnitude” have been fuzzified by the fuzzy set \(BM\).
Source code number: 6-26.

\textbf{fl-def-set}
Explanation: \((\text{fl-def-set})\) defines and creates a fuzzy set by means of two alpha-cuts \(a\text{-cut1}, a\text{-cut2}\). The returned fuzzy set has either a triangular or trapezoidal membership function.
Syntax: \((\text{fl-def-set } \text{name } a\text{-cut1 } a\text{-cut2})\)
- \(\text{name}\): symbol for associating a name to the resulting fuzzy set.
- \(a\text{-cut1}\): first alpha-cut expressed by a list in the following format \((\text{extreme-left } \text{extreme-right alpha-cut-value})\).
- \(a\text{-cut2}\): second alpha-cut with the same format as \(a\text{-cut1}\). It is required that \(a\text{-cut1} < a\text{-cut2}\).
FuzzyLisp representation: FLSSR
Example(s): \( (\text{fl-def-set} \ '\text{young} \ ' \ (15.0 \ 35.0 \ 0) \ (25.0 \ 25.0 \ 1.0)) \rightarrow (\text{young} \ 15 \ 25 \ 25 \ 35) \)
Source code number: 6-5a.

\textbf{fl-defuzzify-rules}

Explanation: This function takes as input the list obtained from either \( \text{(fl-dtranslate-all-rules)} \) or \( \text{(fl-dtranslate-all-rules)} \) and then converts that fuzzy information into a crisp numerical value.
Syntax: \( \text{(fl-defuzzify-rules \ translated-rules)} \)

- translated-rules: list representing the output of either \( \text{(fl-dtranslate-all-rules)} \) or \( \text{(fl-dtranslate-all-rules)} \)

FuzzyLisp representation: FLSSR/FLDSR
Example(s): \( (\text{fl-defuzzify-rules} \ (\text{fl-translate-all-rules} \ \text{rules-controller} \ 22 \ 0.25)) \rightarrow -60 \)
Source code number: 7-11.

\textbf{fl-discretize}

Explanation: \( \text{(fl-discretize)} \) takes a fuzzy set with triangular or trapezoidal characteristic function and discretizes it with a resolution given by \textit{steps}. In other words, it transforms a FuzzyLisp Standard Set Representation into a FuzzyLisp Discrete Set Representation.
Syntax: \( \text{(fl-discretize \ fset \ steps)} \)

- \textit{fset}: a list representing a fuzzy set with a continuous membership function, either a triangle or a trapezium.
- \textit{steps}: an integer representing the resolution of the discretization process.

FuzzyLisp representation: FLSSR \( \rightarrow \) FLDSR
Example(s): \( (\text{fl-discretize} \ '(\text{B1} \ 7 \ 10 \ 12 \ 15) \ 4) \rightarrow (\text{B1} \ (7 \ 0) \ (7.75 \ 0.25) \ (8.5 \ 0.5) \ (9.25 \ 0.75) \ (10 \ 1) \ (10.5 \ 1) \ (11 \ 1) \ (11.5 \ 1) \ (12 \ 1) \ (12.75 \ 0.75) \ (13.5 \ 0.5) \ (14.25 \ 0.25) \ (15 \ 0)) \)
Source code number: 6-6.

\textbf{fl-discretize-fx}

Explanation: This function discretizes any continuous function \( y = f(x) \) in \( n \) steps between \( x = a \) and \( x = b \).
Syntax: \( \text{(fl-discretize-fx \ name \ fx \ steps \ a \ b)} \)

- \textit{name}: Symbol for associating a name to the function’s resulting fuzzy set.
- \textit{fx}: mathematical continuous function to discretize, expressed in Lisp format as a list.
- \textit{steps}: Integer value for expressing the required resolution in the discretization process.
• \(a\): starting point for discretization. Real value.
• \(b\): ending point for discretization. Real value.

FuzzyLisp representation: FLDSR

Example(s): After defining a bell-shaped continuous function to the symbol \(f\) by means of the expression `(setq f `(div (add 1.0 (cos (mul 2.0 pi (sub x 2.0)))) 2.0))`, then, e.g.: `(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 0.9045084972) (2.2 0.6545084972) (2.3 0.34549150283) (2.4 0.09549150283) (2.5 0))`

Source code number: 6-8.

---

**fl-dlv-membership2?**

Explanation: This function returns as a list all the membership degrees of a crisp value \(x\) to every fuzzy set contained in a linguistic variable. All the fuzzy sets from the linguistic variable have a discrete characteristic function.

Syntax: `(fl-dlv-membership2? lv x)`

• \(lv\): a list representing a linguistic variable composed by discrete fuzzy sets.
• \(x\): a real number.

FuzzyLisp representation: FLDSR

Example(s): assuming the linguistic variable \(lv-age-bells\) has been adequately initialized, then the function call `(fl-dlv-membership2? lv-age-bells 23)` produces the following output: `((Young 0.1302642245) (Young+ 0.5478879113) (Mature 0) (Mature+ 0) (Old 0))`

Source code number: 6-25.

---

**fl-dset-hedge**

Explanation: This function applies a fuzzy hedge (linguistic modifier) to a fuzzy set.

Syntax: `(fl-dset-hedge dset hedge)`

• \(dset\): a list representing a discrete fuzzy set.
• \(hedge\): a Lisp symbol, either VERY or FAIRLY.

FuzzyLisp representation: FLDSR

Example(s): `(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))`

Source code number: 7-7.

---

**fl-dset-membership?**

Explanation: `(fl-dset-membership?)` returns the interpolated membership degree of a crisp value \(x\) defined on the real axis to the discrete fuzzy set \(fset\). In practical terms the difference with the function `(fl-set-membership?)` is based on the type of
used representation for fuzzy sets. \((\text{fl}-\text{dset-membership}?)\) is used for FLDSR, while
\((\text{fl}-\text{set-membership}?)\) is used for FLSSR.
Syntax: \((\text{fl}-\text{dset-membership}? \text{ dfset} \text{ x})\)
- \text{dfset}: a list representing a fuzzy set with a discrete membership function.
- \text{x}: a real number.

FuzzyLisp representation: FLDSR
Example(s): After assigning, e.g., \((\text{setq} \text{ dA} (\text{fl-discretize} \ (\text{B1} \ 7 \ 10 \ 12 \ 15) \ 4))\),
then \((\text{fl}-\text{dset-membership}? \text{ dA} \ 8.2) \rightarrow \ (\text{B1} \ 0.4)\)
Source code number: 6-7a.

\textbf{fl-dtruth-value-fuzzy-implication-p-q?}
Explanation: This function returns the truth-value of a compound fuzzy implication \(p \rightarrow q\).
Syntax: \((\text{fl-dtruth-value-fuzzy-implication-p-q}? \text{ P Q x y})\)
- \text{P}: a list representing a discrete fuzzy set associated to the predicate of a fuzzy proposition \(p\).
- \text{Q}: a list representing a discrete fuzzy set associated to the predicate of a fuzzy proposition \(q\).
- \text{x}: a real number for expressing the subject of a fuzzy proposition \(p\).
- \text{y}: a real number for expressing the subject of a fuzzy proposition \(q\).

FuzzyLisp representation: FLDSR
Example(s): assuming the fuzzy set \(P\) defined by \((\text{setq} \text{ P} (\text{fl-discretize} \ (\text{old} \ 50 \ 90 \ 90 \ 90) \ 4))\) and another fuzzy set \(Q\) defined by \((\text{setq} \text{ Q} (\text{fl-discretize} \ (\text{young} \ 0 \ 0 \ 15 \ 30) \ 4))\), the fuzzy implication “if John is old then Eva is young” when John is 55 years old and Eva is 18 can be represented by the function call \((\text{fl-dtruth-value-implication-p-q}? \text{ P Q} \ 55 \ 18) \rightarrow 1\).
Source code number: 7-6b.

\textbf{fl-dtruth-value-negation-p?}
Explanation: This function returns the truth-value of the negation of a fuzzy proposition.
Syntax: \((\text{fl-dtruth-value-negation-p}? \text{ P x})\)
- \text{P}: a list representing a discrete fuzzy set associated to the predicate of a fuzzy proposition \(p\).
- \text{x}: a real number for expressing the subject of a fuzzy proposition \(p\).

FuzzyLisp representation: FLDSR
Example(s): assuming the fuzzy set \(P\) defined by \((\text{setq} \text{ Q} (\text{fl-discretize} \ (\text{young} \ 0 \ 0 \ 15 \ 30) \ 4))\), the fuzzy proposition “Eva is not young” when Eva is 18 years old can be represented by the function call \((\text{fl-dtruth-value-negation-p}? \text{ Q} \ 18) \rightarrow 0.2\)
Source code number: 7-5b.
**fl-dtruth-value-p-and-q?**
Explanation: This function returns the truth-value of a compound fuzzy proposition containing the logical connective “and”.
Syntax: `(fl-dtruth-value-p-and-q? P Q x y)`
- P: a list representing a discrete fuzzy set associated to the predicate of a fuzzy proposition p.
- Q: a list representing a discrete fuzzy set associated to the predicate of a fuzzy proposition q.
- x: a real number for expressing the subject of a fuzzy proposition p.
- y: a real number for expressing the subject of a fuzzy proposition q.

FuzzyLisp representation: FLDSR
Example(s): assuming the fuzzy set P defined by `(setq P (fl-discretize '(old 50 90 90 90) 4))` and another fuzzy set Q defined by `(setq Q (fl-discretize '(young 0 0 15 30) 4))`, the fuzzy compound proposition “John is old and Eva is young” when John is 55 years old and Eva is 15 can be represented by the function call `(fl-dtruth-value-p-and-q? P Q 55 18) → 0.125`
Source code number: 7-3b.

**fl-dtruth-value-p-or-q?**
Explanation: This function returns the truth-value of a compound fuzzy proposition containing the logical connective “or”.
Syntax: `(fl-dtruth-value-p-or-q? P Q x y)`
- P: a list representing a discrete fuzzy set associated to the predicate of a fuzzy proposition p.
- Q: a list representing a discrete fuzzy set associated to the predicate of a fuzzy proposition q.
- x: a real number for expressing the subject of a fuzzy proposition p.
- y: a real number for expressing the subject of a fuzzy proposition q.

FuzzyLisp representation: FLDSR
Example(s): assuming the fuzzy set P defined by `(setq P (fl-discretize '(old 50 90 90 90) 4))` and another fuzzy set Q defined by `(setq Q (fl-discretize '(young 0 0 15 30) 4))`, the fuzzy compound proposition “John is old or Eva is young” when John is 55 years old and Eva is 18 can be represented by the function call `(fl-dtruth-value-p-or-q? P Q 55 18) → 0.8`
Source code number: 7-4b.

**fl-expand-contract-set**
Explanation: This function expands or contracts a fuzzy set. The returned fuzzy set is still placed over its original position, but its support and nucleus are expanded or contracted accordingly.
Syntax: `(fl-expand-contract-set fset k)`
- \textit{fset}: a list representing a fuzzy set with a continuous membership function, either a triangle or a trapezium.
- \textit{k}: a real number.

FuzzyLisp representation: FLSSR

Example(s):
\begin{align*}
(\text{fl-expand-contract-set} \ (a \ 0 \ 1 \ 1 \ 2) \ 2.0) & \rightarrow (a \ -1 \ 1 \ 1 \ 3), \\
(\text{fl-expand-contract-set} \ (a \ -1 \ 1 \ 1 \ 3) \ 0.5) & \rightarrow (a \ 0 \ 1 \ 1 \ 2)
\end{align*}

Source code number: 6-18.

**fl-fuzzy-add**

Explanation: Returns a fuzzy number as the result of adding two fuzzy numbers \(A, B\).

Syntax: (fl-fuzzy-add name A B)

- \textit{name}: a symbol for associating a name to the function’s resulting fuzzy number.
- \textit{A}: first fuzzy number to add.
- \textit{B}: second fuzzy number to add.

FuzzyLisp representation: FLSSR

Example(s): After defining two fuzzy numbers, e.g., (setq A \(\{\text{around-2 1.75 2 2 2.25}\}\)) and (setq B \(\{\text{around-5 4.8 5 5 5.2}\}\)), then (fl-fuzzy-add \'A+B A B) \(\rightarrow\) (A+B 6.55 7 7 7.45)

Source code number: 6-13a.

**fl-fuzzy-add-sets**

Explanation: This function returns a fuzzy number as the result of adding all the fuzzy numbers contained in a set of fuzzy numbers.

Syntax: (fl-fuzzy-add-sets fsets name)

- \textit{fsets}: A list containing all the fuzzy numbers to add.
- \textit{name}: a symbol for associating a name to the function’s resulting fuzzy number.

FuzzyLisp representation: FLSSR

Example(s): After creating several fuzzy numbers, e.g., (setq F1 \(\{\text{set1 -2 0 0 2}\}\)), (setq F2 \(\{\text{set2 3 5 5 7}\}\)), (setq F3 \(\{\text{set3 6 7 7 8}\}\)), (setq F4 \(\{\text{set4 7 9 11 12}\}\)), (setq F5 \(\{\text{set5 8 10 10 12}\}\)), then (setq Fsets \(\{\text{F1 F2 F3 F4 F5}\}\)), and finally: (setq SFs (fl-fuzzy-add-sets Fsets \'Sum-of-Fs)) \(\rightarrow\) (Sum-of-Fs 22 31 33 41)

Source code number: 6-19.

**fl-fuzzy-average**

Explanation: This function returns a fuzzy number as the average of \(n\) fuzzy numbers contained in a set of fuzzy numbers.
Syntax: \((fl\text{-fuzzy-average } fsets \text{ name})\)
- \(fsets\): A list containing all the fuzzy numbers to average.
- \(name\): a symbol for associating a name to the function’s resulting fuzzy number.

FuzzyLisp representation: FLSSR
Example(s): \((fl\text{-fuzzy-average } Fsets \text{ ’Average} ) \rightarrow (\text{Average } 4.4 \ 6.2 \ 6.6 \ 8.2)\). See the assignments for building \(Fsets\) in the entry for the function \((fl\text{-fuzzy-add-sets})\).
Source code number: 6-20.

\textbf{fl-fuzzy-div}
Explanation: This function returns a fuzzy number as the result of dividing two fuzzy numbers \(A, B\). \(A\) and \(B\) are represented by triangular or trapezoidal shaped membership functions. The resulting fuzzy number \(A/B\) is represented by means of a discrete characteristic function.
Syntax: \((fl\text{-fuzzy-div } name \ A \ B \ n)\)
- \(name\): a symbol for associating a name to the function’s resulting fuzzy number.
- \(A\): first fuzzy number involved in the \(A/B\) division process.
- \(B\): second fuzzy number involved in the \(A/B\) division process.
- \(n\): integer for expressing the resolution of the process.

FuzzyLisp representation: FLSSR \(\rightarrow\) FLDSR
Example(s): After defining two fuzzy numbers, e.g., \((setq A \text{ ’(around-2 1.75 2 2 2.25)}\) and \((setq B \text{ ’(around-5 4.8 5 5 5.2)}\), then \((fl\text{-fuzzy-div } B/A \ B \ A \ 5) \rightarrow (B/A (2.133333333 0) (2.2 0.2) (2.269767442 0.4) (2.342857143 0.6) (2.419512195 0.8) (2.5 1) (2.584615385 0.8) (2.673684211 0.6) (2.767567568 0.4) (2.866666667 0.2) (2.971428571 0))\)
Source code number: 6-15.

\textbf{fl-fuzzy-factor}
Explanation: This function takes a fuzzy number \(A\) and then multiplies it by a crisp number \(k\), returning the fuzzy number \(k.A\). In practical terms when \(k > 1\) it performs a multiplication and when \(k < 1\) it performs a division by \(k\).
Syntax: \((fl\text{-fuzzy-factor } fset \ k)\)
- \(fset\): a list representing a fuzzy number with a continuous membership function, either a triangle or a trapezium (fuzzy interval).
- \(K\): a real number.

FuzzyLisp representation: FLSSR
Example(s): After defining a fuzzy number, e.g., \((setq A \text{ ’(AI -2 3 3 8)}\), then \((fl\text{-fuzzy-factor } A \ 3) \rightarrow (AI \ -6 \ 9 \ 9 \ 24), \text{ and } (fl\text{-fuzzy-factor } A \ 0.25) \rightarrow (AI \ -0.5 \ 0.75 \ 0.75 \ 2)\)
Source code number: 6-16.
**file-fuzzy-mult**

Explanation: Returns a fuzzy number as the result of multiplying two fuzzy numbers $A$, $B$. $A$ and $B$ are represented by triangular or trapezoidal shaped membership functions. The resulting fuzzy number $A \cdot B$ is represented by means of a discrete characteristic function.

Syntax: $(\text{fl-fuzzy-mult name } A \ B \ n)$

- **name**: a symbol for associating a name to the function’s resulting fuzzy number.
- **$A$**: first fuzzy number to multiply.
- **$B$**: second fuzzy number to multiply.
- **$n$**: integer for expressing the resolution of the process.

FuzzyLisp representation: FLSSR $\rightarrow$ FLDSR

Example(s): After defining two fuzzy numbers, e.g., $(\text{setq A } '(\text{around-2 } 1.75 \ 2 \ 2.25))$ and $(\text{setq B } '(\text{around-5 } 4.8 \ 5 \ 5 \ 5.2))$, then $(\text{fl-fuzzy-mult 'AxB A B } 5) \rightarrow (\text{AxB} \ (8.4 \ 0) \ (8.712 \ 0.2) \ (9.028 \ 0.4) \ (9.348 \ 0.6) \ (9.672 \ 0.8) \ (10 \ 1) \ (10.332 \ 0.8) \ (10.668 \ 0.6) \ (11.008 \ 0.4) \ (11.352 \ 0.2) \ (11.7 \ 0))$

Source code number: 6-14.

**file-fuzzy-shift**

Explanation: This function shifts (moves horizontally) a fuzzy set towards left or right over the real axis $X$ by an amount given by a real value $x$, returning the shifted fuzzy set.

Syntax: $(\text{fl-fuzzy-shift fset } x)$

- **fset**: a list representing a fuzzy set with a continuous membership function, either a triangle or a trapezium.
- **$x$**: a real number.

FuzzyLisp representation: FLSSR

Example(s): $(\text{fl-fuzzy-shift (young-plus 15.0 25.0 35.0 45.0) 5.0}) \rightarrow (\text{young-plus} \ 20 \ 30 \ 40 \ 50)$

Source code number: 6-17.

**file-fuzzy-sub**

Explanation: Returns a fuzzy number as the result of subtracting two fuzzy numbers $A$, $B$.

Syntax: $(\text{fl-fuzzy-sub name } A \ B)$

- **name**: a symbol for associating a name to the function’s resulting fuzzy number.
- **$A$**: first fuzzy number involved in the $A-B$ subtraction process.
- **$B$**: second fuzzy number involved in the $A-B$ subtraction process.
FuzzyLisp representation: FLSSR
Example(s): After defining two fuzzy numbers, e.g., (setq A '(around-2 1.75 2 2 2.25)) and (setq B '(around-5 4.8 5 5 5.2)), then (fl-fuzzy-sub 'A-B A B) → (A-B -3.45 -3 -3 -2.55)
Source code number: 6-13b.

**fl-inference**
Explanation: This function is an automatic call to the functions (fl-translate-all-rules) and (fl-defuzzify-rules) in a sort of black box that directly transforms two input crisp values entering a Fuzzy Rule Based System (FRBS) into a resulting crisp value.
Syntax: (fl-inference x y)
- x: first crisp input numerical value to the FRBS.
- y: second crisp input numerical value to the FRBS
FuzzyLisp representation: FLSSR
Example(s): (fl-inference rules-controller 22 0.25) → -60
Source code number: 7-12.
Note: The FuzzyLisp function (fl-dinference) is suited to deal with FRBS where input linguistic variables are composed by fuzzy sets with discrete membership functions (FLDSR).

**fl-int-div**
Explanation: Returns a list representing the division of two intervals.
Syntax: (fl-intv-div x1 x2 x3 x4)
- x1, x2: real numbers expressing the left and right extremes of an interval [x1, x2].
- x3, x4: real numbers expressing the left and right extremes of an interval [x3, x4].
FuzzyLisp representation: n/a
Example(s): (fl-intv-div 2 4 1 3) → (0.6666666667 4)
Source code number: 6-12d.

**fl-intv-add**
Explanation: Returns a list representing the addition of two intervals.
Syntax: (fl-intv-add x1 x2 x3 x4)
- x1, x2: real numbers expressing the left and right extremes of an interval [x1, x2].
- x3, x4: real numbers expressing the left and right extremes of an interval [x3, x4].
FuzzyLisp representation: n/a
Example(s): (fl-intv-add 2 4 1 3) → (3 7)
Source code number: 6-12a.

**fl-intv-mult**

Explanation: Returns a list representing the multiplication of two intervals.
Syntax: (fl-intv-mult x1 x2 x3 x4)
- \(x1, x2\): real numbers expressing the left and right extremes of an interval \([x1, x2]\).
- \(x3, x4\): real numbers expressing the left and right extremes of an interval \([x3, x4]\).

FuzzyLisp representation: n/a
Example(s): (fl-intv-mult 2 4 1 3) → (2 12)
Source code number: 6-12c.

**fl-intv-sub**

Explanation: Returns a list representing the subtraction of two intervals.
Syntax: (fl-intv-sub x1 x2 x3 x4)
- \(x1, x2\): real numbers expressing the left and right extremes of an interval \([x1, x2]\).
- \(x3, x4\): real numbers expressing the left and right extremes of an interval \([x3, x4]\).

FuzzyLisp representation: n/a
Example(s): (fl-intv-sub 2 4 1 3) → (-1 3)
Source code number: 6-12b.

**fl-list-sets**

Explanation: This function prints all the fuzzy sets belonging to a linguistic variable at the Lisp console.
Syntax: (fl-list-sets lv)
- \(lv\): a list representing a linguistic variable.

FuzzyLisp representation: FLSSR
Example(s): assuming the linguistic variable \(lv-age\) has been adequately initialized, then the function call (fl-list-sets lv-age) produces the following output:
  : (young 0 0 15 30)
  : (young-plus 15 30 30 45)
  : (mature 30 45 45 60)
  : (mature-plus 45 60 60 75)
  : (old 60 75 90 90)
Source code number: 6-22.
**fl-lv-membership?**
Explanation: This function prints all the membership degrees of a crisp value \( x \) to every fuzzy set contained in a linguistic variable at the Lisp console.
Syntax: \((fl-lv-membership? \ lv \ x)\)
- \( \lv \): a list representing a linguistic variable.
- \( x \): a real number.
FuzzyLisp representation: FLSSR
Example(s): \((fl-lv-membership? \ lv\text{-age} \ 32)\) produces the following output:
: (young 0)
: (young-plus 0.8666666667)
: (mature 0.1333333333)
: (mature-plus 0)
: (old 0)
Source code number: 6-23.

**fl-lv-membership2?**
Explanation: This function returns as a list all the membership degrees of a crisp value \( x \) to every fuzzy set contained in a linguistic variable.
Syntax: \((fl-lv-membership2? \ lv \ x)\)
- \( \lv \): a list representing a linguistic variable.
- \( x \): a real number.
FuzzyLisp representation: FLSSR
Example(s): \((fl-lv-membership2? \ lv\text{-age} \ 32)\) → ((young 0) (young-plus 0.8666666667) (mature 0.1333333333) (mature-plus 0) (old 0))
Source code number: 6-24.

**fl-set-complement-membership?**
Explanation: This function returns the membership degree of a crisp value \( x \) to the complementary set of \( \text{fset} \).
Syntax: \((fl-set-complement-membership? \ \text{fset} \ x)\)
- \( \text{fset} \): a list representing a fuzzy set with a continuous membership function, either a triangle or a trapezium.
- \( x \): a real number.
FuzzyLisp representation: FLSSR
Example(s): \((fl-set-complement-membership? \ '(B1 7 10 12 15) \ 9)\) → (B1 0.3333333333)
Source code number: 6-9.
**fl-set-intersect-membership?**

Explanation: \( (fl-set-intersect-membership?) \) returns the membership degree of the crisp value \( x \) to the intersection of fuzzy sets \( fset1 \) and \( fset2 \).

Syntax: \( (fl-set-intersect-membership? name fset1 fset2 x) \)

- **name**: a symbol for associating a name to the function’s resulting list.
- **fset1**: a list representing a fuzzy set with a continuous membership function, either a triangle or a trapezium.
- **fset2**: a list representing a fuzzy set with a continuous membership function, either a triangle or a trapezium.
- **x**: a real number.

FuzzyLisp representation: FLSSR

Example(s): After defining two fuzzy sets \( A \) and \( B \) e.g.: \( (setq A '(Triangle 0 5 5 10)) \) and \( (setq B '(Trapezium 5 10 15 20)) \), then: \( (fl-set-intersect-membership? 'AintB A B 8) \) → \( (AintB 0.4) \)

Source code number: 6-11.

**fl-set-membership?**

Explanation: \( (fl-set-membership?) \) returns the membership degree of a crisp value \( x \) defined on the real axis to the fuzzy set \( fset \).

Syntax: \( (fl-set-membership? fset x) \)

- **fset**: a list representing a fuzzy set with a continuous membership function, either a triangle or a trapezium.
- **x**: a real number.

FuzzyLisp representation: FLSSR

Example(s): \( (fl-set-membership? 'young 12 20 28 36) 24) \) → \( 1; \)
\( (fl-set-membership? 'young 12 20 28 36) 54) \) → \( 0 \)

Source code number: 6-2.

**fl-set-union-membership?**

Explanation: \( (fl-set-union-membership?) \) returns the membership degree of the crisp value \( x \) to the union of fuzzy sets \( fset1 \) and \( fset2 \).

Syntax: \( (fl-set-union-membership? name fset1 fset2 x) \)

- **name**: a symbol for associating a name to the function’s resulting list.
- **fset1**: a list representing a fuzzy set with a continuous membership function, either a triangle or a trapezium.
- **fset2**: a list representing a fuzzy set with a continuous membership function, either a triangle or a trapezium.
- **x**: a real number.
FuzzyLisp representation: FLSSR

Example(s): After defining two fuzzy sets A and B e.g.: (setq A ‘(Triangle 0 5 5 10)) and (setq B ‘(Trapezium 5 10 15 20)), then: (fl-set-union-membership? ’AuB A B 7.5) → (AuB 0.5)

Source code number: 6-10.

**fl-simple-defuzzification**
Explanation: This function takes a fuzzy number and produces a crisp number for it with a simple algorithm.
Syntax: (fl-simple-defuzzification fset mode)
- *fset*: a list representing a fuzzy set with a continuous membership function, either a triangle or a trapezium.
- *mode*: an integer value from 1 to 4. This parameter gives increasing weight to the nucleus in the process of defuzzification.

FuzzyLisp representation: FLSSR
Example(s): (fl-simple-defuzzification '(q 0 1 1 5) 1) → 2; (fl-simple-defuzzification '(q 0 1 1 5) 4) → 1.375
Source code number: 6-21.

**fl-translate-all-rules**
Explanation: This function evaluates all the fuzzy rules contained in the knowledge database of a Fuzzy Rule Based System (FRBS), calling iteratively to the function *(fl-translate-rule).*
Syntax: (fl-translate-all-rules set-of-rules x y)
- *set-of-rules*: The complete set of expert rules of a FRBS expressed in list format.
- *x*: first crisp input numerical value to the FRBS.
- *y*: second crisp input numerical value to the FRBS

FuzzyLisp representation: FLSSR
Example(s): (fl-translate-all-rules rules-controller 22 0.25) → ((0 0 0 0) (0 0 0 0) (0 0 0 0) (0 0 0 0) (0 0 0 0) (0 0 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 1 0.2 -20) (0 0 0 -0) (0 0 0 0) (0 1 0 0))
Source code number: 7-10.
Note: The FuzzyLisp function *(fl-dtranslate-all-rules)* is suited to deal with FRBS where input linguistic variables are composed by fuzzy sets with discrete membership functions (FLDSR).
**fl-translate-rule**

Explanation: This function takes an expert rule at a time from a Fuzzy Rule Based System (FRBS), performs the adequate inferences and translates the rule into membership degrees, that is, into numerical values.

Syntax: \( (fl\text{-translate-rule header rule x y}) \)

- **header**: first sublist from the body of rules in the FRBS where the enumeration of the used linguistic variables is expressed.
- **rule**: rule to translate from the FRBS in its adequate list format.
- **x**: first crisp input numerical value to the FRBS.
- **y**: second crisp input numerical value to the FRBS.

FuzzyLisp representation: FLSSR

Example(s): \( (fl\text{-translate-rule (first rules-controller) (nth 9 rules-controller) 22 0.25}) \to (0.8 1 0.8 -40) \)

Source code number: 7-9.

Note: The FuzzyLisp function \( (fl\text{-dtranslate-rule}) \) is suited to deal with FRBS where input linguistic variables are composed by fuzzy sets with discrete membership functions (FLDSR).

**fl-truth-value-fuzzy-implication-p-q?**

Explanation: This function returns the truth-value of a compound fuzzy implication \( p \rightarrow q \).

Syntax: \( (fl\text{-truth-value-fuzzy-implication-p-q? P Q x y}) \)

- **P**: a list representing a fuzzy set associated to the predicate of a fuzzy proposition \( p \).
- **Q**: a list representing a fuzzy set associated to the predicate of a fuzzy proposition \( q \).
- **x**: a real number for expressing the subject of a fuzzy proposition \( p \).
- **y**: a real number for expressing the subject of a fuzzy proposition \( q \).

FuzzyLisp representation: FLSSR

Example(s): assuming the fuzzy set \( P \) defined by \( (setq P \ (old 50 90 90 90)) \) and another fuzzy set \( Q \) defined by \( (setq Q \ (young 0 0 15 30)) \), the fuzzy compound proposition “John is old or Eva is young” when John is 55 years old and Eva is 18 can be represented by the function call \( (fl\text{-truth-value-implication-p-q? P Q 55 18}) \to 1 \). When John is 90 years old and Eva is 30, then \( (fl\text{-truth-value-implication-p-q? P Q 90 30}) \to 0 \)

Source code number: 7-6.

**fl-truth-value-negation-p?**

Explanation: This function returns the truth-value of the negation of a fuzzy proposition.
Syntax: \((fl\text{-truth-value-negation-p}\ P \ x)\)
- \(P\): a list representing a fuzzy set associated to the predicate of a fuzzy proposition \(p\).
- \(x\): a real number for expressing the subject of a fuzzy proposition \(p\).

FuzzyLisp representation: FLSSR
Example(s): assuming the fuzzy set \(P\) defined by \((setq \ P \ '(\text{old} \ 50 \ 90 \ 90 \ 90))\), the fuzzy proposition “John is not old” when John is 55 years can be represented by the function call \((fl\text{-truth-value-negation-p}\ P \ 55) \rightarrow 0.875\)
Source code number: 7-5.

\textbf{fl\text{-truth-value-p-and-q}?}
Explanation: This function returns the truth-value of a compound fuzzy proposition containing the logical connective “and”.
Syntax: \((fl\text{-truth-value-p-and-q}\ P \ Q \ x \ y)\)
- \(P\): a list representing a fuzzy set associated to the predicate of a fuzzy proposition \(p\).
- \(Q\): a list representing a fuzzy set associated to the predicate of a fuzzy proposition \(q\).
- \(x\): a real number for expressing the subject of a fuzzy proposition \(p\).
- \(y\): a real number for expressing the subject of a fuzzy proposition \(q\).

FuzzyLisp representation: FLSSR
Example(s): assuming the fuzzy set \(P\) defined by \((setq \ P \ '(\text{old} \ 50 \ 90 \ 90 \ 90))\) and another fuzzy set \(Q\) defined by \((setq \ Q \ '(\text{young} \ 0 \ 0 \ 15 \ 30))\), the fuzzy compound proposition “John is old and Eva is young” when John is 55 years old and Eva is 18 can be represented by the function call \((fl\text{-truth-value-p-and-q}\ P \ Q \ 55 \ 18) \rightarrow 0.125\)
Source code number: 7-3.

\textbf{fl\text{-truth-value-p-or-q}?}
Explanation: This function returns the truth-value of a compound fuzzy proposition containing the logical connective “or”.
Syntax: \((fl\text{-truth-value-p-or-q}\ P \ Q \ x \ y)\)
- \(P\): a list representing a fuzzy set associated to the predicate of a fuzzy proposition \(p\).
- \(Q\): a list representing a fuzzy set associated to the predicate of a fuzzy proposition \(q\).
- \(x\): a real number for expressing the subject of a fuzzy proposition \(p\).
- \(y\): a real number for expressing the subject of a fuzzy proposition \(q\).
FuzzyLisp representation: FLSSR
Example(s): assuming the fuzzy set $P$ defined by 
\[
\text{(setq } P \text{ '(old 50 90 90 90))}
\]
and another fuzzy set $Q$ defined by 
\[
\text{(setq } Q \text{ '(young 0 0 15 30))}
\]
the fuzzy compound proposition “John is old or Eva is young” when John is 55 years old and Eva is 18 can be represented by the function call 
\[
(fl-truth-value-p-or-q? P Q 55 18) \rightarrow 0.8
\]
Source code number: 7-4.