Proposal for a fuzzified Arden Syntax based on version 2.8

Karsten Fehre

Medexter Healthcare GmbH
Borschkegasse 7/5, A-1090 Vienna, Austria

+43-1-968 03 24 tel
+43-1-968 09 22 fax
kf@medexter.com
www.medexter.com

November 4, 2011

Contents

1. Truth Value 3
2. Fuzzy Data Types 3
   2.1. Fuzzy Number 3
   2.2. Fuzzy Time 5
   2.3. Fuzzy Duration 5
3. Applicability 5
4. Fuzzy Set ... (unary, right associative) 6
5. ... Fuzzified By ... (binary, non-associative) 8
6. Defuzzified ... (unary, right associative) 10
7. Applicability [Of] ... (binary, non-associative) 11
8. ... As Truth Value (unary, left associative) 14
9. ... Is Fuzzy (unary, non-associative) 15
10. ... Is Crisp (unary, non-associative) 16
11. Conclude 17
12. … Is In … (binary, non-associative) 18
13. … <= … (binary, non-associative) 18
14. … >= … (binary, non-associative) 19
15. ... Or ... (binary, left associative) 19
16. ... And ... (binary, left associative) 20
17. not … (unary, non-associative) 21
18. Sort Expressions 22
19. If-Then Statements 23
20. If-Then-Else Statements 23
21. If-Then-Elseif Statements 24
22. If-Then-ElseIf-Aggregate Statement 25
23. Simple Switch-Case Statement 27
24. Switch-Case-Default Statement 27
25. Switch-Case- Aggregate Statement 27
26. Linguistic Variables 28
27. At Least ... Of ... (binary, right associative) 30
28. At Most ... Of ... (binary, right associative) 30
29. Operator Precedence and Associativity 32

1. Truth Value

Affected chapters: (8.2)

Insert as chapter: 8.13 or as replacement for 8.2

Text to be inserted:

The data type of propositional variables is denoted by **truth value** or—for reasons of backwards compatibility—, equivalently, Boolean. A variable of this type stores real numbers between 0 and 1. The Boolean value **true** is equal to the truth value 1 and the Boolean value **false** is equal to the truth value 0. We may write:

`Var := truth value 0;` or, equivalently `Var := false;`

`Var := truth value 0.667;`

`Var := truth value 1;` or, equivalently `Var := true;`

Affected BNF: the non-terminal "`boolean_value`" must be extended to:

```
<boolean_value> ::=
  "TRUE"
| "FALSE"
| "TRUTH VALUE" <number>
| "TRUTH VALUE" "TRUE"
| "TRUTH VALUE" "FALSE"
```

2. Fuzzy Data Types

Affected chapters: none

Insert as chapter: 8.14

Text to be inserted:

Fuzzy data types are fuzzy sets over the data types number, time or duration.

2.1. Fuzzy Number

Affected chapters: none

Insert as chapter: 8.14.1

Text to be inserted:

The data type **fuzzy number** is dedicated to fuzzy sets over the reals. A fuzzy number partitions the reals into a finite number of (possibly unbounded) intervals, on each of which the fuzzy set is linear and continuous.

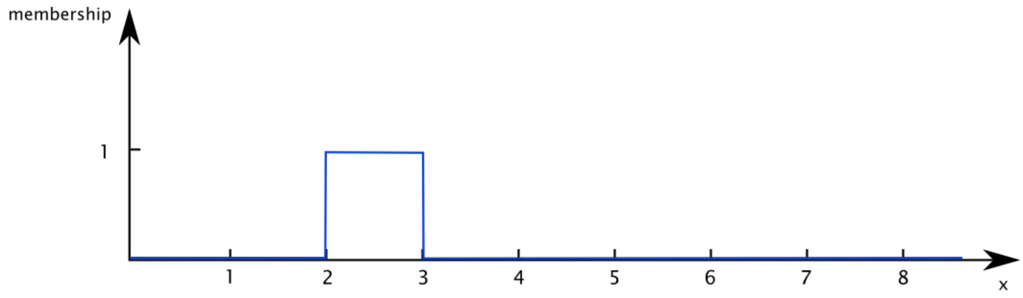
Formally, a fuzzy set **u : R->[0, 1]** can be stored in a variable of type fuzzy number if the following condition is met: There are **a_1 < a_2 < … < a_k** with **k >= 1**, such that u is linear on each open interval **(a_1; a_2), … , (a_{k-1}; a_k)**, **u** is constant on **(-∞; a_1)** and **(a_k; +∞)**, and for each **x** in **R**, **u(x)** coincides either with the left limit or the right limit of **u** at **x**. If **u** is continuous, we then define:

```
Fuzzyset u := fuzzy set (a1,t1), (a2,t2), ..., (ak,tk);
```

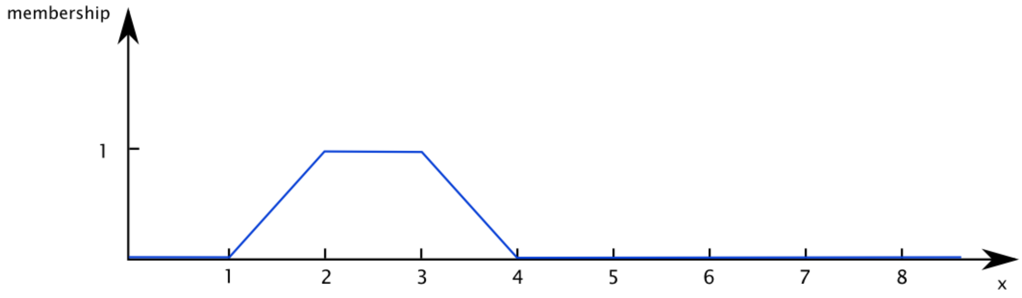
where **t_i = u(a_i) for i = 1, …, k**.

The characteristic functions are allowed to include discontinuities, which are not likely to be required in applications, but should at least be definable. At discontinuity points we denote the left as well as the right limit. The first assignment is the value at that point, unless the second one appears twice. For instance,

```
TwotoThree := fuzzy set (2, 0), (2, 1), (2, 1), (3, 1), (3,0);
```



```
OnetoFour := fuzzy set (1, 0), (2, 1), (2, 1), (3, 1), (4,0);
```



Fuzzy sets whose graph forms a symmetrical triangle around one point, which is mapped to 1, are called triangular normal fuzzy sets. A simplified notation is permitted for these:

an expression of the form fuzzy set (a - b, 0), (a, 1), (a + b, 0), where a; b in R and b > 0, may also be written as:

```
a fuzzified by b
```

2.2. Fuzzy Time

Affected chapters: none
Insert as chapter: 8.14.2
Text to be inserted:
The data type **fuzzy time** is dedicated to fuzzy sets over times. Except for the simplified notation, all definitions of fuzzy number apply mutatis mutandis to fuzzy time.

For the simplified notation, a time constant can only be fuzzified by duration. Thus, we define

```
AfuzzyTime := today fuzzified by 1 day;

simple := 2009-10-10 fuzzified by 12 hours;

complex := fuzzy set (2009-10-10,0), (2009-10-11,1), (2009-11-10,1), (2009-
11-11,0);
```

2.3. Fuzzy Duration

Affected chapters: none
Insert as chapter: 8.14.3
Text to be inserted:
All definitions of a fuzzy number apply mutatis mutandis to **fuzzy duration**.

```
simple := 14 days fuzzified by 1 day;

complex := fuzzy set (2 days,0), (3 days,1), (14 days,1), (31 days,0);
```

3. Applicability

Affected chapters: none
Insert as chapter: 8.15
Text to be inserted:
All simple data types (Truth Value, Boolean, Number, Time, Duration, String, Term, Query Results, Time-of-Day, Day-of-Week, Fuzzy Types) are endowed with additional information called the **degree of applicability**. This degree of applicability stores a truth

value expressing, which refers to the degree to which it is reasonable to use the value of a variable. It is 1 by default, and—whenever the program branches—it is reduced automatically according to the weight assigned to that branch. The programmer may decide to make explicit use of this component but he is not required to do so. To access the degree of applicability of an expression, the Arden Syntax programmer is referred to the **applicability [of]** operator (Chapter 9.19.4).

4. Fuzzy Set … (unary, right associative)

Affected chapters: none

Insert as chapter: 9.19.1 (into a new chapter "9.19 Fuzzy Operators")

Text to be inserted:

The **fuzzy set …** operator creates a new fuzzy set as described in 8.14.1, 8.14.2, or 8.14.3, according to the provided parameter. Its usage is:

```
<var> := FUZZY SET "(" <number>, <truth-value>")", "(" <number>, <truth-value>")", ...;

Var1 := fuzzy set (2, 0), (3, 1), (4, 1), (5, 0);


<var> := FUZZY SET "(" <time>, <truth-value>")", "(" <time>, <truth-value>")", ...;

Var2 := fuzzy set (now – 2 days, 0), (now, 1), (now + 1 day, 0);

Var3 := fuzzy set (2001-12-12, 0), (2003-12-12, 1), (2009-01-01, 0);


<var> := FUZZY SET "(" <duration>, <truth-value>")", "(" <duration>, <truth-value>")", ...;

Var4 := fuzzy set (2 days, 0), (3 days, 1), (4 days, 1), (5 days, 0);
```

Affected BNF:

```
<logic_assignment> ::=
             <identifier_becomes> <expr>
             | <time_becomes> <expr>
             | <applicability_becomes> <expr>
```

```
       | <identifier_becomes> <call_phrase>
       | “(” <data_var_list> “)” “:=„ <call_phrase>
       | “LET” “(” <data_var_list> “)” “BE“ <call_phrase>
       | <identifier_becomes> <new_object_phrase>
       | <identifier_becomes> <fuzzy_set_phrase>

<expr_fuzzy_set> ::=
       <expr>
       | <fuzzy_set_phrase>
<expr_factor_atom> ::=
       <identifier>
       | <number>
       | <string>
       | <time_value>
       | <boolean_value>
       | <weekday_literal>
       | “TODAY”
       | “TOMORROW”
       | “NULL”
       | “CONCLUDE” /* only available in the action slot */
       | <it>
       | “(” “)”
       | “(” <expr_fuzzy_set> “)”

<data_assign_phrase> ::=
        “READ“ <read_phrase>
       | “MLM” <term>
       | “MLM” <term> “FROM” “INSTITUTION” <string>
       | “MLM” “MLM SELF”
       | “INTERFACE“ <mapping_factor>
       | “EVENT“ <mapping_factor>
       | “MESSAGE“ <mapping_factor>
       | “MESSAGE” “AS” <identifier> <mapping_factor>
       | “MESSAGE” “AS” <identifier>
       | “DESTINATION“ <mapping_factor>
       | “DESTINATION” “AS” <identifier> <mapping_factor>
       | “DESTINATION” “AS” <identifier>
       | “ARGUMENT”
       | “OBJECT“ <object_definition>
       | “LINGUISTIC VARIABLE“ <object_definition>
```

```
                     | <call_phrase>
                     | <new_object_phrase>
                     | <fuzzy_set_phrase>
                     | <expr>
<fuzzy_set_phrase> ::=
                     “FUZZY SET” <fuzzy_set_init_list>
                     | <expr_duration> “FUZZIFIED BY” <expr_duration>
                     | <expr_factor> “FUZZIFIED BY” <expr_factor>
<fuzzy_set_init_list> ::=
                     <fuzzy_set_init_element>
                     | <fuzzy_set_init_list> “,” <fuzzy_set_init_element>
<fuzzy_set_init_element> ::=
                     “(” <fuzzy_set_init_factor> “,” <expr_factor> “)”
<fuzzy_set_init_factor> ::=
                     <expr_factor>
                     | <number> <duration_op>
```

5. … Fuzzified By … (binary, non-associative)

Affected chapters: none

Insert as chapter: 9.19.2 (into a new chapter “9.19 Fuzzy Operators”)

Text to be inserted:

The **… fuzzified by …** operator creates a new triangular fuzzy set as described in 8.14.1, 8.14.2, or 8.14.3, according to the provided parameter. The operator returns **null** if the data types are not compatible. Its usage is:

```
<var> := <number> FUZZIFIED BY <number>;

Var1 := 7 fuzzified by 2;


<var> := <time> FUZZIFIED BY <duration>;

Var2 := now fuzzified by 2 days;


<var> := <duration> FUZZIFIED BY <duration>;

Var3 := 7 days  fuzzified by 2 hours;
```

Affected BNF:

```
<logic_assignment> ::=
             <identifier_becomes> <expr>
             | <time_becomes> <expr>
             | <applicability_becomes> <expr>
             | <identifier_becomes> <call_phrase>
             | “(” <data_var_list> “)” “:=„ <call_phrase>
             | “LET” “(” <data_var_list> “)” “BE“ <call_phrase>
             | <identifier_becomes> <new_object_phrase>
             | <identifier_becomes> <fuzzy_set_phrase>

<expr_fuzzy_set> ::=
             <expr>
             | <fuzzy_set_phrase>
<expr_factor_atom> ::=
             <identifier>
             | <number>
             | <string>
             | <time_value>
             | <boolean_value>
             | <weekday_literal>
             | “TODAY”
             | “TOMORROW”
             | “NULL”
             | “CONCLUDE” /* only available in the action slot */
             | <it>
             | “(” “)”
             | “(” <expr_fuzzy_set> “)”

<data_assign_phrase> ::=
              “READ“ <read_phrase>
             | “MLM” <term>
             | “MLM” <term> “FROM” “INSTITUTION” <string>
             | “MLM” “MLM SELF”
             | “INTERFACE“ <mapping_factor>
             | “EVENT“ <mapping_factor>
             | “MESSAGE“ <mapping_factor>
             | “MESSAGE” “AS” <identifier> <mapping_factor>
             | “MESSAGE” “AS” <identifier>
             | “DESTINATION“ <mapping_factor>
             | “DESTINATION” “AS” <identifier> <mapping_factor>
             | “DESTINATION” “AS” <identifier>
```

```
                         | “ARGUMENT”
                         | “OBJECT“ <object_definition>
                         | “LINGUISTIC VARIABLE“ <object_definition>
                         | <call_phrase>
                         | <new_object_phrase>
                         | <fuzzy_set_phrase>
                         | <expr>
      <fuzzy_set_phrase> ::=
                         “FUZZY SET” <fuzzy_set_init_list>
                         | <expr_duration> “FUZZIFIED BY” <expr_duration>
                         | <expr_factor> “FUZZIFIED BY” <expr_factor>
      <fuzzy_set_init_list> ::=
                         <fuzzy_set_init_element>
                         | <fuzzy_set_init_list> “,” <fuzzy_set_init_element>
      <fuzzy_set_init_element> ::=
                         “(” <fuzzy_set_init_factor> “,” <expr_factor> “)”
      <fuzzy_set_init_factor> ::=
                         <expr_factor>
                         | <number> <duration_op>
```

6. Defuzzified … (unary, right associative)

Affected chapters: none

Insert as chapter: 9.19.3 (into a new chapter “9.19 Fuzzy Operators”)

Text to be inserted:

The **defuzzified** operator expects a fuzzy data type value as its argument. The operator converts a fuzzy set into a crisp data type. To calculate the result the **mean of maximum** method is used. That is, the average of the midpoints of the intervals, mapping to the supremum of the fuzzy set image, is calculated. If it is not exclusive to finite intervals, **null** is returned. The usage of the operator is:

```
<n:crisp_type> := DEFUZZIFIED <n:fuzzy_type>

7 := Defuzzified 7 fuzzified by 2;
```

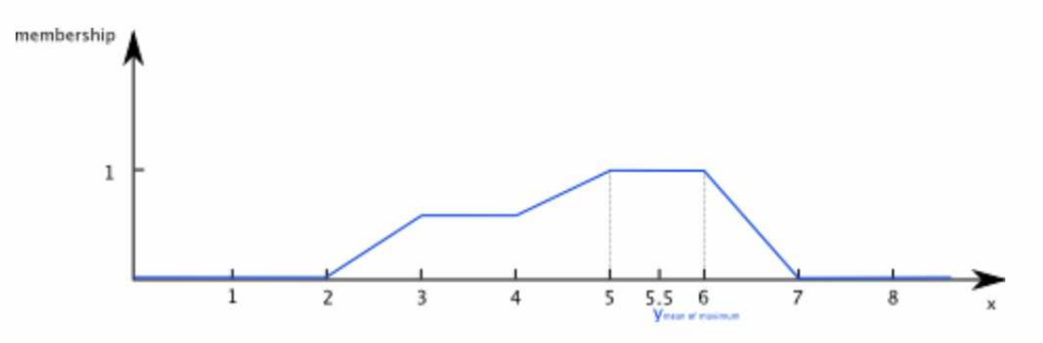


Figure 1: Mean Of Maximum

Affected BNF:

```
<of_noread_func_op> ::=
              “ANY” | “ALL”
              | “NO” | “SLOPE”
              | “STDDEV” | “VARIANCE”
              | “INCREASE” | “PERCENT” “INCREASE”
              | “%” “INCREASE” | “DECREASE”
              | “PERCENT” “DECREASE” | “%” “DECREASE”
              | “INTERVAL” | “TIME”
              | “TIME” “OF” “DAY” | “DAY” “OF” “WEEK”
              | “ARCCOS” | “ARCSIN”
              | “ARCTAN” | “COSINE”
              | “COS” | “SINE”
              | “SIN” | “TANGENT”
              | “TAN” | “EXP”
              | “FLOOR” | “INT”
              | “ROUND” | “CEILING”
              | “TRUNCATE” | “LOG”
              | “LOG10” | “ABS”
              | “SQRT” | “EXTRACT” “YEAR”
              | “EXTRACT” “MONTH” | “EXTRACT” “DAY”
              | “EXTRACT” “HOUR” | “EXTRACT” “MINUTE”
              | “EXTRACT” “SECOND” | “EXTRACT” “TIME” “OF” “DAY”
              | “STRING” | “EXTRACT” “CHARACTERS”
              | “REVERSE” | “LENGTH”
              | “UPPERCASE” | “LOWERCASE”
              | “CLONE” | “EXTRACT” “ATTRIBUTE” “NAMES”
              | “APPLICABILITY” | “DEFUZZIFIED”
```

7. Applicability [Of] … (binary, non-associative)

Affected chapters: none

Insert as chapter: : 9.19.4 (into a new chapter "9.19 Fuzzy Operators")

Text to be inserted:

The **applicability of** operator returns the degree applicability of a value. Since **null** is not allowed as degree of applicability, a value is always returned (default degree of applicability is 1). The result of the **applicability** operator preserves the primary time and degree of applicability of its argument; so **applicability applicability x** is equivalent to **applicability x**. Its usage is (assuming that data0 has the degree of applicability of 0.44):

```
<n:truth-value> := APPLICABILITY [OF] <n:any-type>

0.44 := APPLICABILITY OF data0

0.44 := APPLICABILITY APPLICABILITY data0

(1,1) := APPLICABILITY (3,4)
```

The inverse of the **applicability** operator (to set the degree of applicability of a value) can be achieved by using the **applicability** operator on the left side of an assignment statement. For example:

```
APPLICABILITY [OF] <n:any-type> := <n:truth-value>;

APPLICABILITY data1 := 0.44;
```

If the identifier on the left hand side of an assignment statement refers to a list, the behavior of the **applicability** assignment is undefined.

Affected BNF:

```
<logic_assignment> ::=
             <identifier_becomes> <expr>
             | <time_becomes> <expr>
             | <applicability_becomes> <expr>
             | <identifier_becomes> <call_phrase>
             | "(" <data_var_list> ")" ":=„ <call_phrase>
             | "LET" "(" <data_var_list> ")" "BE" <call_phrase>
             | <identifier_becomes> <new_object_phrase>
             | <identifier_becomes> <fuzzy_set_phrase>

<applicability_becomes> ::=
             "APPLICABILITY" "OF" <identifier> ":="
             | “APPLICABILITY” <identifier> “:=”
             | “LET” “APPLICABILITY” “OF” <identifier> “BE”
             | “LET” “APPLICABILITY” <identifier> “BE”

<of_noread_func_op> ::=
             “ANY” | “ALL”
             | “NO” | “SLOPE”
             | “STDDEV” | “VARIANCE”
             | “INCREASE” | “PERCENT” “INCREASE”
             | “%” “INCREASE” | “DECREASE”
             | “PERCENT” “DECREASE” | “%” “DECREASE”
             | “INTERVAL” | “TIME”
             | “TIME” “OF” “DAY” | “DAY” “OF” “WEEK”
             | “ARCCOS” | “ARCSIN”
             | “ARCTAN” | “COSINE”
             | “COS” | “SINE”
             | “SIN” | “TANGENT”
             | “TAN” | “EXP”
             | “FLOOR” | “INT”
             | “ROUND” | “CEILING”
             | “TRUNCATE” | “LOG”
             | “LOG10” | “ABS”
             | “SQRT” | “EXTRACT” “YEAR”
             | “EXTRACT” “MONTH” | “EXTRACT” “DAY”
             | “EXTRACT” “HOUR” | “EXTRACT” “MINUTE”
             | “EXTRACT” “SECOND” | “EXTRACT” “TIME” “OF” “DAY”
             | “STRING” | “EXTRACT” “CHARACTERS”
             | “REVERSE” | “LENGTH”
             | “UPPERCASE” | “LOWERCASE”
             | “CLONE” | “EXTRACT” “ATTRIBUTE” “NAMES”
             | “APPLICABILITY” | “DEFUZZIFIED”
<data_assignment> ::=
             <identifier_becomes> <data_assign_phrase>
             | <time_becomes> <expr>
             | <applicability_becomes> <expr>
             | “(” <data_var_list> “)” “:=” “READ“ <read_phrase>
             | “LET” “(” <data_var_list> “)” “BE” “READ“ <read_phrase>
             | “(” <data_var_list> “)” “:=” “READ” “AS” <identifier>
             <read_phrase>
```

```
                      | “LET” “(” <data_var_list> “)” “BE” “READ” “AS” <identifier>
                      <read_phrase>
                      | “(” <data_var_list> “)” “:=” “ARGUMENT”
                      | “LET” “(” <data_var_list> “)” “BE” “ARGUMENT”
        <action_statement> ::=
                      /* empty */
                      | “IF“ <action_if_then_else2>
                      | “SWITCH” <identifier> <action_switch>
                      | “FOR” <identifier> “IN” <expr> “DO“ <action_block> “;” “ENDDO”
                      | “WHILE” <expr> “DO“ <action_block> “;” “ENDDO”
                      | <call_phrase>
                      | <call_phrase> “DELAY” <expr>
                      | “WRITE” <expr>
                      | “WRITE” <expr> “AT” <identifier>
                      | “RETURN” <expr>
                      | <identifier_becomes> <expr>
                      | <time_becomes> <expr>
                      | <applicability_becomes> <expr>
                      | <identifier_becomes> <new_object_phrase>
```

8. … As Truth Value (unary, left associative)

Affected chapters: none

Insert as chapter: 9.20.4 (into a new chapter “9.20 Type Conversion Operators” – together with “as number”, “as time”, and “as string”)

Text to be inserted:

The **as truth value** operator attempts to convert a Number or Boolean to a truth value. If the conversion to a truth value is possible, the truth value is returned, otherwise **null** is returned. The primary time of the argument is preserved.

The usual use for this is to convert a calculated number into the corresponding truth value. If the number is not between 0 and 1, the result will be **null**.

Boolean values are translated at follows: Boolean **true** is represented as truth value 1 and Boolean **false** is represented as truth value 0.

```
<n:truth value> := <n:number> AS TRUTH VALUE;

0.33 := 0.33 AS TRUTH VALUE;

null := "xyz" AS TRUTH VALUE;
null := 400 AS TRUTH VALUE;

<n:truth value> := <n:Boolean> AS TRUTH VALUE;

1 := True AS TRUTH VALUE;

0 := False AS TRUTH VALUE;

() := () AS TRUTH VALUE;
```

Affected BNF:

```
<unary_comp_op> ::=
                "PRESENT"
                | "NULL"
                | "BOOLEAN"
                | "TRUTH VALUE"
                | "CRISP"
                | "FUZZY"
                | "NUMBER"
                | "TIME"
                | "DURATION"
                | "STRING"
                | "LIST"
                | "OBJECT"
                | "LINGUISTIC VARIABLE"
                | <identifier>
                | "TIME" "OF" "DAY"
<as_func_op> ::=
                "NUMBER"
                | "TRUTH VALUE"
```

9. ... Is Fuzzy (unary, non-associative)

Affected chapters: none

Insert as chapter: 9.6.27

Text to be inserted:

The **is fuzzy** operator returns **true** if the argument's data type is fuzzy number, fuzzy time or fuzzy duration. Otherwise it returns **false**. **Is fuzzy** never returns **null**. Its usage is:

```
<n:Truth Value> := <n:any-type> IS FUZZY

false := 3 IS FUZZY
```

```
true := (FUZZY SET (0,0), (1,1)) IS FUZZY

true := (today fuzzified by 2 days) IS FUZZY
```

Affected BNF:

```
<unary_comp_op> ::=
             “PRESENT”
             | “NULL”
             | “BOOLEAN”
             | “TRUTH VALUE”
             | “CRISP”
             | “FUZZY”
             | “NUMBER”
             | “TIME”
             | “DURATION”
             | “STRING”
             | “LIST”
             | “OBJECT”
             | “LINGUISTIC VARIABLE”
             | <identifier>
             | “TIME” “OF” “DAY”
```

10.... Is Crisp (unary, non-associative)

Affected chapters: none

Insert as chapter: 9.6.28

Text to be inserted:

The **is crisp** operator returns **true** if the argument's data type is not a fuzzy number, fuzzy time or fuzzy duration. Otherwise it returns **false**. **Is crisp** never returns **null**. Its usage is:

```
<n:Truth Value> := <n:any-type> IS CRISP

true := 3 IS CRISP

false := (FUZZY SET (0,0), (1,1)) IS CRISP

false := (today fuzzified by 2 days) IS CRISP
```

Affected BNF:

```
<unary_comp_op> ::=
             “PRESENT”
             | “NULL”
```

```
                          | "BOOLEAN"
                          | "TRUTH VALUE"
                          | "CRISP"
                          | "FUZZY"
                          | "NUMBER"
                          | "TIME"
                          | "DURATION"
                          | "STRING"
                          | "LIST"
                          | "OBJECT"
                          | "LINGUISTIC VARIABLE"
                          | <identifier>
                          | "TIME" "OF" "DAY"
```

11.Conclude

Affected chapters: 10.2.4

Insert as chapter: none

Text to be inserted:

If the expression (<expr>) in the conclude statements is a truth value > 0, the applicabilities of all variables are multiplied by this value, and the action slot is executed immediately. Otherwise the whole MLM or the current branch of the MLM terminates immediately.

…

Furthermore the reserved word "**conclude**" can be used in the action slot to retrieve the degree of applicability the action slot is executed with.

```
Applicability_of_action_slot:= conclude;
```

Affected BNF:

```
<expr_factor_atom> ::=
                          <identifier>
                          | <number>
                          | <string>
                          | <time_value>
                          | <boolean_value>
                          | <weekday_literal>
| “TODAY”
| “TOMORROW”
| “NULL”
| “CONCLUDE” /* only available in the action slot */
| <it>
| “(” “)”
| “(” <expr_fuzzy_set> “)”
```

12.... Is In ... (binary, non-associative)

Affected chapters: 9.6.14

Insert as chapter: none

Text to be inserted:

...

The operator **is in** also checks for containment in a fuzzy set, returning a **truth value**. The arguments are of a crisp type and a fuzzy type. The fuzzy type must be derived from the rough crisp type of the other argument (e.g.: if the crisp value is a number, the fuzzy value must be a fuzzy number), otherwise **false** is returned. If we define a fuzzy number as:

```
u := fuzzy set (a1,t1), (a2,t2),...,(ak,tk);
```

Then a crisp number **r**, contained in **Var**, may be correlated to the fuzzy number **u** contained in **FuzzyVar**, by the expression

```
Var is in FuzzyVar
```

This simply gives the value of **u** at **r**, **u(r)**.

If one argument is **null**, then **null** is always returned.

Primary times are not used in determining the result. The primary time of the result is determined by the rules in Section 9.1.4. The usage of the **... is [in] ...** operator is:

```
<n:truth value> := <n:crisp> is <n:fuzzy>

0.5 := 4 is in 5 fuzzified by 2

0.5 := 2 is in Fuzzy Set (0,0), (4,1), (5,0)
```

13.... <= ... (binary, non-associative)

Affected chapters: 9.5.4
Insert as chapter: none
Text to be inserted:
The **<=** operators also support the same arguments as the **is [in]** operator. When the first argument is a crisp type and the second a fuzzy type, the **<=** operator returns the **supremum** of u(x) for all r <= x, while r is the value stored in the first argument and u(x) is the fuzzy set provided by the second argument.

14.... >= ... (binary, non-associative)

Affected chapters: 9.5.6
Insert as chapter: none
Text to be inserted:
The **>=** operators also support the same arguments as the **is [in]** operator. When the first argument is a crisp type and the second a fuzzy type, the **>=** operator returns the **supremum** of u(x) for all r >= x, while r is the value stored in the first argument and u(x) is the fuzzy set provided by the second argument.

15.... Or ... (binary, left associative)

Affected chapters: 9.4.1
Insert as chapter: none
Text to be inserted:
The **or** operator performs the logical disjunction of its two arguments. If one argument is true and the other is not a truth value, the result is truth value true. If both arguments are truth values, the maximum of both arguments is returned. Otherwise the result is **null**. Its usage is:

```
<1:Boolean> := <1:any-type> OR <1:any-type>

true := true OR false

false := false OR false

true := true OR null

null := false OR null

null := false OR 3.4
```

```
(true, true) := (true, false) OR (false, true)

() := () OR ()
```

The operator’s truth table is given below. The column and row “**other**” means any of these data types: null, number, time, duration, or string.

		Right argument			
Left Argument	OR	TRUE	other Truth Value	other	NULL
	TRUE	TRUE	True	TRUE	TRUE
	other Truth Value	TRUE	Max(a,b)	NULL	NULL
	other	TRUE	NULL	NULL	NULL
	NULL	TRUE	NULL	NULL	NULL

16.... And ... (binary, left associative)

Affected chapters: 9.4.1
Insert as chapter: none
Text to be inserted:
The **and** operator performs the logical conjunction of its two arguments. If one argument is false and the other is not a truth value, the result is the truth value false. If both arguments are truth values, the minimum of both arguments is returned. Otherwise the result is **null**. Its usage is:

```
<1:Boolean> := <1:any-type> AND <1:any-type>

false := true AND false

false := false AND false

null := true AND null

false := false AND null

null := true AND 3.4
```

```
(false, true) := (true, true) AND (false, true)

() := () AND ()
```

The operator's truth table is given below. The column and row "**other**" means any of these data types: null, number, time, duration, or string.

		Right argument			
	AND	FALSE	other Truth Value	other	NULL
Left Argument	FALSE	FALSE	FALSE	FALSE	FALSE
	other Truth Value	FALSE	Min(a,b)	NULL	NULL
	other	FALSE	NULL	NULL	NULL
	NULL	FALSE	NULL	NULL	NULL

17.not … (unary, non-associative)

Affected chapters: none
Insert as chapter: 8.14.3
Text to be inserted:
The **not …** operator performs the logical negation of its argument. If the argument is a truth value, the negation is the subtraction from 1. Its usage is:

```
<n:truth value> := AND <n:truth value>

false := NOT true;

true := NOT false;

null := NOT null

truth value 0.8 := NOT truth value 0.2

(true, false) := NOT (false, true)

() := NOT ()
```

18.Sort Expressions

Affected chapters: 9.2.4

Insert as chapter: none

Text to be replaced:

The sort operator reorders a list based on element keys, which are either the element values (keyword **data**), the primary times (keyword **time**), or the applicability (keyword **applicability**). An optional modifier may be use with the sort operator. If used, the modifier must be placed immediately after the sort keyword. The following keywords can be placed after the **sort** keyword: **data**, **time**, or **applicability**, which are mutually exclusive. If no modifier is used, the sort operator defaults to a data sort. Direction of sorting is always ascending. For a descending sort, **reverse** can be used.

The sort options are considered to be part of the sort operator for precedence purposes. This resolves the potential conflict with the **time [of]** operator (9.17.1) or the applicability [of] operator. Thus the expression "**sort time x**" should be parsed as "sort the list x by time" rather than as "extract the primary times from the list x and sort the list of times."

When sorting by primary times, if any of the elements do not have primary times, the result is null. (The sort argument can always be qualified by **where time of it is present**, if this is not desired behavior.) Elements with the same key will be kept in the same order as they appear in the argument. If any pair of element key cannot be compared because of type clashes, **sort** returns **null** (that is, when sorting by data, any null value (or non-comparable value); when sorting by time, any null primary time results in **null**). The sorting by applicabilities is defined as sorting by primary times.

Its usage is (assuming that **data1** has a data value of **30, 10, 20** with time values **1991-01-01T00:00:00, 1991-02-01T00:00:00, 1991-01-03T00:00:00 and applicability values truth value 0.3, truth value 0.5, truth value 0.7**):

```
<n:any-type> := SORT <n:any-type>

<n:any-type> := SORT [DATA | TIME | APPLICABILITY] <n:any-type>

…

(10, 20, 30) := SORT  APPLICABILITY data1

(30, 20, 10) := REVERSE (SORT  APPLICABILITY data1)
```

```
      null := SORT  APPLICABILITY (3,1,2,null)
       () := SORT  APPLICABILITY ()
```

Affected BNF:

```
      <sort_option> ::=
                    /*empty*/
                    | “TIME”
                    | “DATA”
                    | “APPLICABILITY”
```

19.If-Then Statements

Affected chapters: 10.2.2

Insert as chapter: none

Text to be inserted:

The **if-then** statement permits conditional execution based upon the value of an expression. It tests whether the expression (<expr>) is equal to a single Boolean **true**. If it is, then a block of statements (<block>) is executed. (A block of statements is simply a collection of valid statements possibly including other if-then statements; thus the if-then statement is a nested structure.) If the expression is a list, or if it is any single item other than Boolean true or not a truth value, then the block of statements is not executed. The flow of control then continues with subsequent statements.

20.If-Then-Else Statements

Affected chapters: 10.2.2.2

Insert as chapter: none

Text to be inserted:

This form executes **<block1>** if **<expr1>** is **true**; otherwise it executes **<block2>**:

```
IF <expr1> THEN
   <block1>
ELSE
   <block2>
ENDIF;
```

If, however, **<expr1>** is any truth value (t) between 0 and 1, the program splits: **<block1>** and **<block2>**, named program branches in the sequel, will be executed in parallel. To this end each branch is provided with its own set of variables which, accordingly, are duplicated.

Moreover, the degree of applicability of each variable is in case of **<block1>** multiplied by t, in case of **<block2>** multiplied by 1 − t. t and 1 − t are called the relative weights of **<block1>** and **<block2>**, respectively. The program may branch several times. Each command executed during the run of the program is assigned a weight in the straightforward manner. The weight is 1 as long as the program does not split; when the weight is w and the program enters a branch with relative weight t, the weight will be reduced to w · t.

In a branch of weight **w**, the range of the degree of applicability of any variable is [0, **w**]. Whenever the content of a variable is changed, its degree of applicability will be reduced to **w** if necessary. For example:

```
Var := 0;

Con := truth value 0.2;

If con then

  Var := Var + 1;

Else

  Var := Var + 3;

Endif
```

The result of this example are two branches of the MLM execution, where in the first branch **Var** has the value 1 and the degree of applicability 0.2 and in the second branch **Var** has the value 3 and the degree of applicability 0.8. This mean the execution of the MLM will return two different values with different degree of applicability.

21.If-Then-Elseif Statements

Affected chapters: 10.2.2.3
Insert as chapter: none
Text to be inserted:
… (Current content) …

If the expressions are truth values the execution of the MLM is split into N branches. Branching into n + 1 blocks is possible by the following statement:

```
IF <expr1> then <block1>

ELSEIF <expr2> then <block2>

.  .  .

ELSEIF <exprN> then <blockN>

ELSE <blockN+1>

ENDIF;
```

In this case the relative weight t_i of the i-th branch is given by **<expr$_i$>**, where i = 1, ..., n. If **<expr$_i$>** is undefined, it is treated as $t_i = 0$, in which case the branch is not executed. Moreover, if the sum of the t_i is strictly smaller than 1, the relative weight of block$_{n+1}$ will be $1 - t_1 - ... - t_n$, otherwise this block is skipped.

22.If-Then-ElseIf-Aggregate Statement

Affected chapters: none
Insert as chapter: 10.2.2.6
Text to be inserted:
As shown in chapter 10.2.2.2, the program execution is split if the condition of an if-then-else statement evaluates to any truth value between 0 and 1.

Once all branches of a program have completed their execution in parallel, because of an unsharp condition, it is difficult to issue a general recommendation how the program should proceed. Two possibilities exist:

(A) The program remains split, that is, all subsequent commands are executed in parallel as well, the action slot included.

(B) The program reunifies. The multiplied variables are merged into single ones.

Both options are available and possibility (A) is the default. The more appropriate option in the individual situation should be decided on the basis of the specific application.
If (A) is selected, the MLM's results will be provided by each branch separately. The application to which the results are sent–the host system or the calling MLM–must be prepared to deal with the situation. If the MLM is called by another MLM and returns

data, the calling MLM splits accordingly as well.
The possibility (B) implies that the task of combining divergent pieces of information is executed within the MLM itself. To opt for (B), the final line of an if-then-else statement is modified: after the key word endif, the key word aggregate is added. Thus, when writing

```
IF <expr> then <block1>

ELSE <block2>

ENDIF aggregate;
```

the two branches unify after their execution. The program weight is then set to the sum of the weight of the branches, i.e., to the same value as before.

Moreover, corresponding variables are aggregated.

Let Var be a variable defined in at least one branch. As far as the main component is concerned, the procedure is as follows.

If the content of Var is the same in each branch, the content is taken over.

Otherwise, if Var is defined in all branches and of the same simple data type except string, the contents are aggregated according to their **weighted middle**.

If Var is of the same compound type in all branches, we proceed successively with the components in the same manner.

In the remaining cases Var is set to null.

The aggregation of the contents of variables, with respect to the degree of applicability and the primary time, is straightforward. The primary time of Var is transferred if coincident in all branches. If distinct times appear, the primary time will be set to **null**.

Furthermore, as might be expected, the degrees of applicability are added. Thus, if left unchanged during the execution of all branches, the degree of applicability prior the execution of the if-then-else statement will be restored.

Affected BNF:

```
<logic_elseif> ::=
             <logic_endif>
             | “ELSE” <logic_block> “;” <logic_endif>
             | “ELSEIF” <logic_if_then_else2>
```

```
<logic_endif> ::=
             "ENDIF"
             | "ENDIF" "AGGREGATE"
```

23.Simple Switch-Case Statement

Affected chapters: 10.2.3.1

Insert as chapter: none

Text to be inserted:

Equivalent to the **if-then-elseif statement** (see Chapter 10.2.2.3), the execution of **a switch-case** statement can split the program execution into several program branches which will be executed in parallel. This happens if the comparison between the value of a variable and an **<expr>** evaluates to a truth value between 0 and 1.

24.Switch-Case-Default Statement

Affected chapters: 10.2.3.2

Insert as chapter: none

Text to be inserted:

Equivalent to the **if-then-else statement** (see Chapter 10.2.2.2), the execution of **a switch-case-default** statement can split the program execution into several program branches which will be executed in parallel. This happens if the comparison between the value of a variable and an **<expr>** evaluates to a truth value between 0 and 1.

25.Switch-Case- Aggregate Statement

Affected chapters: none

Insert as chapter: 10.2.3.3

Text to be inserted:

The aggregate operator in the switch-case-aggregate or switch-case-default-aggregate statement acts exactly like in the if-then-elseif-aggregate statement. See chapter 10.2.2.6 for more details.

Affected BNF:

```
<logic_switch> ::=
              "SWITCH" <identifier> ":"
              <logic_switch_cases>
```

```
                 <logic_endswitch>

<logic_endswitch> ::=
             “ENDSWITCH” “;”
             | “ENDSWITCH” “AGGREGATE” “;”

<data_switch> ::=
              “SWITCH” <identifier>
              <data_switch_cases>
             <data_endswitch>

<data_endswitch> ::=
             “ENDSWITCH” “;”
             | “ENDSWITCH” “AGGREGATE” “;”
             | “ENDSWITCH” “AGGREGATE WITH” <fuzzy_method> “;”

<action_endswitch> ::=
             “ENDSWITCH” “;”
             | “ENDSWITCH” “AGGREGATE” “;”
             | “ENDSWITCH” “AGGREGATE WITH” <fuzzy_method> “;”
```

26.Linguistic Variables

Affected chapters: none

Insert as chapter: 10.2.20

Text to be inserted:

The clear recommendation to the programmer when using fuzzy data types is, to use fuzzy sets never singly but always in conjunction with other, to define together a subdivision of a value range. Assume a value, stored in the variable parameter, out of an arbitrary interval W. Furthermore, assume three fuzzy sets u_1, u_2, and u_3 over W representing the ranges “low”, “middle”, and “high”. In such a case, it is necessary to save these three fuzzy sets together in a single variable of the type **object** whose fields are named according to the ranges, such as:

```
Range :=  object  [low, middle, high];

Value :=  new  Range;
```

```
Value.low :=  /definition of the fuzzy set u1 /;

Value.middle :=  /definition of the fuzzy set u2 /;

Value.high :=  /definition of the fuzzy set u3 /;
```

Whenever a parameter has a low, medium or high value, it can be evaluated by the following expressions, which provide three truth values, whose sum is truth value 1.

```
Parameter  =  Value.low

Parameter  =  Value.middle

Parameter  =  Value.high
```

To clarify the significance of the fuzzy sets, the keyword **linguistic variable** is used for object declarations where all components are fuzzy data types.

```
RangeOfAge  := linguistic  variable  [young,  middleAge,  old];

Age  :=  new  RangeOfAge;

Age.young := (0  years,  1),  (25  year,  1),  (35  years,  0);

Age.middleAge := (25  years,  0),  (35  years,  1),  (55  years,  1),  (65
years,  0);

Age.old := (55  years,  0),  (65  years,  1);
```

Now, if the variable myAge interpreted as the age of a person, **myAge is Age.young** returns a truth value that indicates the degree to which the statement "is the person young" is justified.

Affected BNF:

```
<data_assign_phrase> ::=
              "READ" <read_phrase>
             | "MLM" <term>
             | "MLM" <term> "FROM" "INSTITUTION" <string>
             | "MLM" "MLM SELF"
             | "INTERFACE" <mapping_factor>
             | "EVENT" <mapping_factor>
             | "MESSAGE" <mapping_factor>
             | "MESSAGE" "AS" <identifier> <mapping_factor>
             | "MESSAGE" "AS" <identifier>
             | "DESTINATION" <mapping_factor>
             | "DESTINATION" "AS" <identifier> <mapping_factor>
             | “DESTINATION” “AS” <identifier>
             | “ARGUMENT”
             | “OBJECT“ <object_definition>
             | “LINGUISTIC VARIABLE“ <object_definition>
             | <call_phrase>
             | <new_object_phrase>
             | <fuzzy_set_phrase>
             | <expr>
```

27.At Least ... Of ... (binary, right associative)

Affected chapters: none
Insert as chapter: 9.13.5
Text to be inserted:

The **at least ... of ...** operator expects a number (call it N) as its first argument and a homogeneous list of truth values as its second argument. The **at least** operator returns the n-th largest value of the list of truth values. If the first argument is not a number or the second parameter contains a non truth value, **null** is returned. If N is greater than the cardinality of the list, the truth value **false** is returned. The primary times of the arguments are not maintained. The degree of applicability of the result is always 1. The usage of the operator is:

```
<1:truth_value> := AT LEAST <1:number> OF <n:any-type>

0.7 := AT LEAST 2 OF (TRUE, truth value 0.7, truth value 0.1, FALSE)

1 := APPLICABILITY OF (AT LEAST 2 OF (TRUE, Truth value 0.7, FALSE))

FALSE := AT LEAST 7 OF (TRUE, truth value 0.1,FALSE)

null := AT LEAST 2 YEARS OF (TRUE, truth value 0.1,FALSE)

null := AT LEAST 2 OF (TRUE, "true", truth value 0.1,FALSE)

1 := APPLICABILITY OF (AT LEAST 2 OF (TRUE, "true", truth value 0.1,FALSE))
```

28.At Most ... Of ... (binary, right associative)

Affected chapters: none
Insert as chapter: 9.13.6
Text to be inserted:

The **at most … of …** operator expects a number (call it N) as its first argument and a homogeneous list of truth values as its second argument. The **at most** operator returns the n-th smallest value of the list of truth values. If the first argument is not a number or the second parameter contains a non truth value, **null** is returned. If N is greater than the cardinality of the list, the truth value **false** is returned. The primary times of the arguments are not maintained. The degree of applicability of the result is always 1. The usage of the operator is:

```
<1:truth_value> := AT MOST <1:number> OF <n:any-type>

0.6 := AT MOST 2 OF (TRUE, truth value 0.4, truth value 0.7, FALSE)

1 := APPLICABILITY OF (AT MOST 2 OF (TRUE,0.5,0.7,0.1,FALSE))

FALSE := AT MOST 7 OF (TRUE, truth value 0.5, FALSE)

null := AT MOST 2 YEARS OF (TRUE,0.5,0.7,0.1,FALSE)

null := AT MOST 2 OF (TRUE,"true",0.7,0.1,FALSE)

1 := APPLICABILITY OF (AT MOST 2 OF (TRUE,"true",0.7,0.1,FALSE))
```

29.Operator Precedence and Associativity

... fuzzified by ... (non-associative)

Fuzzy Set ... (Right associative)

... , ... (left associative)

... merge ... (left associative)

sort ... (non-associative)

... where ... (non-associative)

... or ... (left associative)

... and ... (left associative)

not ... (non-associative)

... = ...|... eq ...|... is equal ... (non-associative)

... <> ...|... ne ...|... is not equal ... (non-associative)

... < ...|... lt ...|... is less than ...|... is not greater than or equal ... (non-associative)

... <= ...|... le ...|... is less than or equal ...|... is not greater than ... (non-associative)

... > ...|... gt ...|... is greater than ...|... is not less than or equal ... (non-associative)

... >= ...|... ge ...|... is greater than or equal ...|... is not less than ... (non-associative)

... is within ... to ... (non-associative)

... is not within ... to ... (non-associative)

... is within ... preceding ... (non-associative)

... is not within ... preceding ... (non-associative)

... is within ... following ... (non-associative)

... is not within ... following ... (non-associative)

... is within ... surrounding ... (non-associative)

... is not within ... surrounding ... (non-associative)

... is within past ... (non-associative)

... is not within past ... (non-associative)

... is within same day as ... (non-associative)

... is not within same day as ... (non-associative)

... is before ... (non-associative)

... is not before ... (non-associative)

... is after ... (non-associative)

... is not after ... (non-associative)

... occur equal ... | ...occur at ...(non-associative)

... occur within ... to ... (non-associative)

```
... occur not within ... to ... (non-associative)
... occur within ... preceding ... (non-associative)
... occur not within ... preceding ... (non-associative)
... occur within ... following ... (non-associative)
... occur not within ... following ... (non-associative)
... occur within ... surrounding ... (non-associative)
... occur not within ... surrounding ... (non-associative)
... occur within past ... (non-associative)
... occur not within past ... (non-associative)
... occur within same day as ... (non-associative)
... occur not within same day as ... (non-associative)
... occur before ... (non-associative)
... occur not before ... (non-associative)
... occur after ... (non-associative)
... occur not after ... (non-associative)
... is in ... | ...in ...(non-associative)
... is not in ... | ...not in ...(non-associative)
... is present | ... is not null (non-associative)
... is not present | ... is null (non-associative)
... is Boolean (non-associative)
... is not Boolean (non-associative)
... is number (non-associative)
... is not number (non-associative)
... is time (non-associative)
... is not time (non-associative)
... is time of day (non-associative)
... is not time of day (non-associative)
... is duration (non-associative)
... is not duration (non-associative)
... is string (non-associative)
... is not string (non-associative)
... is list (non-associative)
... is not list (non-associative)
... is object (non-associative)
... is fuzzy (non-associative)
... is crisp (non-associative)
... is linguistic variable (non-associative)
... is not object (non-associative)
... is not linguistic variable (non-associative)
... is <object-name> (non-associative)
... is not <object-name> (non-associative)

... || ... (left associative)
... formatted with ... (non- associative)
uppercase ...(right associative)
lowercase ...(right associative)
trim ...(right associative)
trim left ...(right associative)
trim right ...(right associative)
substring ...characters from ...(right associative)
substring ...characters from ...starting at ...(right associative)
localized ...(non-associative)
localized ...by ...(right associative)

+ ... (non-associative)
... + ... (left associative)
- ... (non-associative)
... - ... (left associative)

... * ... (left associative)
... / ... (left associative)

... ** ... (non-associative)

... round ... (non-associative)

... before ... (non-associative)
... after ... | ...from ...(non-associative)

... ago (non-associative)

... year | ... years (non-associative)
... month | ... months (non-associative)
... week | ... weeks (non-associative)
... day | ... days (non-associative)
... hour | ... hours (non-associative)
... minute | ... minutes (non-associative)
... second | ... seconds (non-associative)
...matches pattern ...(non-associative)
find ...[in] ...(right associative)
find ...[in] ...starting at ...(right associative)

count [of] ... (right associative)
exist [of] ... (right associative)
avg [of] ... | average [of] ... (right associative)
median [of] ... (right associative)
sum [of] ... (right associative)
stddev [of] ... (right associative)
```

variance [of] ... (right associative)

any [of] ... (right associative)

all [of] ... (right associative)

no [of] ... (right associative)

slope [of] ... (right associative)

min ... from | minimum ... from ... (right associative)

min [of] ... | minimum [of] ... (right associative)

max ... from ... | maximum ... from ... (right associative)

max [of] ... | maximum [of] ... (right associative)

index min ... from | index minimum ... from ... (right associative)

index min [of] ... | index minimum [of] ... (right associative)

index max ... from ... | index maximum ... from ... (right associative)

index max [of] ... | index maximum [of] ... (right associative)

at least ... from ... (right associative)

at most ... from ... (right associative)

last ... from ... (right associative)

last [of] ... (right associative)

first ... from ... (right associative)

first [of] ... (right associative)

latest ... from ... (right associative)

latest [of] ... (right associative)

earliest ... from ... (right associative)

earliest [of] ... (right associative)

nearest ... from ... (right associative)

index nearest ... from ... (right associative)

increase [of] ... (right associative)

decrease [of] ... (right associative)

percent increase [of] ... |

percent decrease [of] ... |

interval [of] ... (right associative)

time [of] ... (right associative)

applicability [of] ... (right associative)

defuzzified ... (right associative)

time of day [of] ... (right associative)

day of week [of] ... (right associative)

arcos [of] ... (right associative)

arcsin [of] ... (right associative)

arctan [of] ... (right associative)

cos [of] ... | cosine [of] ... (right associative)

sin [of] ... | sine [of] ... (right associative)

```
tan [of] ... | tangent [of] ... (right associative)
exp [of] ... (right associative)
floor [of] ... (right associative)
ceiling [of] ... (right associative)
truncate [of] ... (right associative)
log [of] ... (right associative)
log10 [of] ... (right associative)
abs [of] ... (right associative)
sqrt [of] ... (right associative)
extract year [of] ... (right associative)
extract month [of] ... (right associative)
extract hour [of] ... (right associative)
extract minute [of] ... (right associative)
extract second [of] ... (right associative)
reverse [of] ... (right associative)
extract characters [of] ... (right associate)
string [of] ... (right associative)
length [of] ...(right associative)
... . ...(right associative)
attribute ...from ...(right associative)
extract attribute names ...(right associative)
clone ...(right associative)
```

... as truth value (left associative)