Free Fuzzy Logic Library

FFLL and FCL

Minimum FCL Compliance

FFLL is not fully compliant with the FCL standards set forth in IEC 61131-7. To reach the "basic level" compliance (as stated in Table 6.1-1 of IEC 61131-7) the following elements are mandatory:

Language Element	Keyword	Details
function block declaration	VAR_INPUT, VAR_OUTPUT	contains input and output variables
membership function	input variable: TERM	maximum of three points (degree of membership co-ordinate = 0 or 1)
	output variable: TERM	only singletons
conditional aggregation	operator: AND	algorithm: MIN
activation		Not relevant because singletons are used only
accumulation (result aggregation)	operator: ACCU	algorithm: MAX
defuzzification	METHOD	algorithm: COGS
condition	*IF ... IS ...*	*n subconditions*
conclusion	*THEN*	*only one subconclusion*
weighting factor	*WITH*	*value only*

FFLL complies with all these "minimum standards".

Production Rules

The following are the "production rules" defined for the Fuzzy Control Language (FCL). Elements of the FCL in GREY are not (currently) supported by FFLL. Elements in RED are functionality added by FFLL.

```
function_block_declaration ::= 'FUNCTION_BLOCK'
function_block_name
                                    {fb_io_var_declarations}
                                    {other_var_declarations}
```

function_block_body
'END_FUNCTION_BLOCK'

fb_io_var_declarations ::= input_declarations | output_declarations

other_var_declarations ::= var_declarations

function_block_body ::= {fuzzify_block}
{defuzzify_block}
{rule_block}
{option_block}

fuzzify_block ::= 'FUZZIFY' variable_name
{linguistic_term}
'END_FUZZIFY

defuzzify_block ::= 'DEFUZZIFY' f_variable_name
{linguistic_term}
defuzzification_method
default_value
[range]
'END_FUZZIFY'

rule_block ::= 'RULEBLOCK' rule_block_name
operator_definition
[activation_method]
accumulation_method
{rule}
'END_RULEBLOCK'

option_block ::= 'OPTION'
any manufacturere specific parameter
'END_OPTION'

linguistic_term ::= 'TERM' term_name ':=' membership_function ';'

membership_function ::= singleton | points

singleton ::= numeric_literal | variable_name

points ::= {'(' numeric_literal | variable_name ',' numeric_literal □)'}

defuzzification_method ::= 'METHOD' ':' 'COG' | 'COGS' | 'COA' | 'LM' | 'RM' | MOM';'

default_value ::= 'DEFAULT' ':=' numeric_literal | 'NC' ';'

range ::= 'RANGE' ':=' '('numeric_literal '..' numeric_literal')' ';'

operator_definition ::= (‘OR’ ‘:’ ‘MAX’ | ‘ASUM’ | ‘BSUM’) | (‘AND’ ‘:’ ‘MIN’ | ‘PROD’ | ‘BDIF’) ‘;’

activation_method ::= ‘ACT’ ‘:’ ‘PROD’ | ‘MIN’ ‘;’

accumulation_method ::= ‘ACCU’ ‘:’ ‘MAX’ | ‘BSUM’ | ‘NSUM’ ‘;’

rule ::= ‘RULE’ integer_literal ‘:’
‘IF’ condition ‘THEN’ conclusion [WITH weighting_factor] ‘;’

condition ::= (subcondition | variable_name) {AND’ | ‘OR’ (subcondition | variable_name)}

subcondition ::= (‘NOT’ ‘(‘ variable_name ‘IS’ [‘NOT’]) term_name ‘)’) | (variable_name ‘IS’ [‘NOT’] term_name)

FFLL shorthand: subcondition ::= term_name

Note: This is one area where FFLL uses a shorthand of just using the term name. To adhere to the standard it should use subconditions of (variable_name IS term_name). FFLL assumes the term_name part of each rule is specified in the order that the variables are declared so it "knows" which variable_name the term_name belongs to.

Note: Regardless if the strict FCL syntax or the FFLL shorthand is used, FFLL assumes the variables are "AND"ed in the order they are declared.

conclusion ::= { (variable_name | (variable_name ‘IS’ term_name)) ‘,’}

weighting_factor ::= variable | numeric_literal

function_block_name ::= identifier

rule_block_name ::= identifier

term_name ::= identifier

variable_name ::= identifier

numeric_literal ::= integer_literal | real_literal

letter ::= 'A' | 'B' | <...> | 'Z' | 'a' | 'b' | <...> | 'z'

digit ::= '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'

identifier ::= (letter | ('_' (letter | digit))) {['_'] (letter | digit)}

input_declarations ::=
'VAR_INPUT' ['RETAIN' | 'NON_RETAIN']
input_declaration ';'
{input_declaration ';'}
'END_VAR'

input_declaration ::= var_init_decl | edge_declaration

var_init_decl ::= var1_init_decl | array_var_init_decl |
structured_var_init_decl | fb_name_decl | string_var_declaration

var1_init_decl ::= var1_list ':'
(simple_spec_init | subrange_spec_init | enumerated_spec_init)

var1_list ::= variable_name {',' variable_name}

array_var_init_decl ::= var1_list ':' array_spec_init

output_declarations ::=
'VAR_OUTPUT' ['RETAIN' | 'NON_RETAIN']
var_init_decl ';'
{var_init_decl ';'}
'END_VAR'

real_type_name ::= 'REAL' | 'LREAL'

numeric_type_name ::= integer_type_name | real_type_name

elementary_type_name ::= numeric_type_name | date_type_name |
bit_string_type_name | 'STRING' | 'WSTRING' | 'TIME'

simple_type_name ::= identifier

simple_type_declaration ::= simple_type_name ':' simple_spec_init

simple_specification ::= elementary_type_name | simple_type_name

simple_spec_init := simple_specification [':=' constant]