ANNEX A - Specification method for textual languages (normative)

Programming languages are specified in terms of a syntax, which specifies the allowable combinations of symbols which can be used to define a program; and a set of semantics, which specify the relationship between programmed operations and the symbol combinations defined by the syntax.

A.1 Syntax

A syntax is defined by a set of terminal symbols to be utilized for program specification; a set of non-terminal symbols defined in terms of the terminal symbols; and a set of production rules specifying those definitions.

A.1.1 Terminal symbols

The terminal symbols for textual programmable controller programs shall consist of combinations of the characters in the ISO/IEC 646 character set. For interchange of programs between systems, these characters shall be represented by the seven-bit character codes defined in ISO 646. For the purposes of this part, terminal textual symbols consist of the appropriate character string enclosed in paired single or double quotes. For example, a terminal symbol consisting of the double quote itself would be represented by "".

A special terminal symbol utilized in this syntax is the end-of-line delimiter, which is represented by the unquoted character string EOL. This symbol shall normally consist of the FE5 (CR = carriage return) character defined by ISO/IEC 646. Language implementors shall specify any deviation from this usage; in any case, no characters other than those in ISO/IEC 646 are allowed.

A second special terminal symbol utilized in this syntax is the "null string", that is, a string containing no characters. This is represented by the terminal symbol NIL.

A.1.2 Non-terminal symbols

Non-terminal textual symbols shall be represented by strings of lower-case letters, numbers, and the underline character (_), beginning with a lower-case letter. For instance, the strings

nonterm1

and

non_term_2

are valid non-terminal symbols, while the strings

3nonterm

and

_nontermm

are not.
A.1.3 Production rules

The production rules for textual programmable controller programming languages shall form an extended grammar in which each rule has the form

\[
\text{non\_terminal\_symbol ::= extended\_structure}
\]

This rule can be read as:

"A non_terminal_symbol can consist of an extended_structure."

Extended structures can be constructed according to the following rules:

1) The null string, NIL, is an extended structure.
2) A terminal symbol is an extended structure.
3) A non-terminal symbol is an extended structure.
4) If S is an extended structure, then the following expressions are also extended structures:
   - (S), meaning S itself.
   - \{S\}, closure, meaning zero or more concatenations of S.
   - [S], option, meaning zero or one occurrence of S.
5) If S1 and S2 are extended structures, then the following expressions are extended structures:
   - S1 | S2, alternation, meaning a choice of S1 or S2.
   - S1 S2, concatenation, meaning S1 followed by S2.
   - Concatenation precedes alternation, that is, S1 | S2 S3 is equivalent to S1 | (S2 S3), and S1 S2 | S3 is equivalent to (S1 S2) | S3.

A.2 Semantics

Programmable controller textual programming language semantics are defined in this Part by appropriate natural language text, accompanying the production rules, which references the descriptions provided in the appropriate clauses. Standard options available to the user and manufacturer are specified in these semantics. In some cases it is more convenient to embed semantic information in an extended structure. In such cases, this information is delimited by paired angle brackets, for example, <semantic information>. 

ANNEX B - Formal specifications of language elements (normative)

B.0 Programming model

The contents of this annex are normative in the sense that a compiler which is capable of recognizing all the syntax in this annex shall be capable of recognizing the syntax of any textual language implementation complying with this standard.

PRODUCTION RULES:

\[
\text{library_element_name} ::= \text{data_type_name} \mid \text{function_name} \\
\quad \quad \quad \quad \mid \text{function_block_type_name} \mid \text{program_type_name} \\
\quad \quad \quad \quad \mid \text{resource_type_name} \mid \text{configuration_name}
\]

\[
\text{library_element_declaration} ::= \text{data_type_declaration} \\
\quad \quad \quad \quad \mid \text{function_declaration} \mid \text{function_block_declaration} \\
\quad \quad \quad \quad \mid \text{program_declaration} \mid \text{configuration_declaration}
\]

SEMANTICS: These productions reflect the basic programming model defined in 1.4.3, where *declarations* are the basic mechanism for the production of named *library elements*. The syntax and semantics of the non-terminal symbols given above are defined in the subclauses listed below.

<table>
<thead>
<tr>
<th>Non-terminal symbol</th>
<th>Syntax</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>data_type_name</td>
<td>B.1.3</td>
<td>2.3</td>
</tr>
<tr>
<td>data_type_declaration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>function_name</td>
<td>B.1.5.1</td>
<td>2.5.1</td>
</tr>
<tr>
<td>function_declaration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>function_block_type_name</td>
<td>B.1.5.2</td>
<td>2.5.2</td>
</tr>
<tr>
<td>function_block_declaration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>program_type_name</td>
<td>B.1.5.3</td>
<td>2.5.3</td>
</tr>
<tr>
<td>program_declaration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>resource_type_name</td>
<td>B.1.7</td>
<td>2.7</td>
</tr>
<tr>
<td>configuration_name</td>
<td></td>
<td></td>
</tr>
<tr>
<td>configuration_declaration</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
B.1 Common elements

B.1.1 Letters, digits and identifiers

PRODUCTION RULES:

letter ::= 'A' | 'B' | <...> | 'Z' | 'a' | 'b' | <...> | 'z'
digit ::= '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'
octal_digit ::= '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7'
hex_digit ::= digit | 'A' | 'B' | 'C' | 'D' | 'E' | 'F' | 'a' | 'b' | 'c' | 'd' | 'e' | 'f'
identifier ::= (letter | '(' letter | digit)) \{'_\}' (letter | digit)

SEMANTICS:
The ellipsis <...> here indicates the ISO/IEC 646 sequence of 26 letters.
Characters from national character sets can be used; however, international portability
of the printed representation of programs cannot be guaranteed in this case.
The case of letters shall be significant in terminal symbols, but not in other syntactic
elements.

B.1.2 Constants

PRODUCTION RULE:

constant ::= numeric_literal | character_string | time_literal

SEMANTICS:
The external representations of data described in 2.2 are designated as "constants" in
this annex.

B.1.2.1 Numeric literals

PRODUCTION RULES:

numeric_literal ::= integer_literal | real_literal | 'TRUE' | 'FALSE'
integer_literal ::= signed_integer | binary_integer | octal_integer | hex_integer
signed_integer ::= ['+' | '-' ] integer
integer ::= digit \{'_\}' integer
binary_integer ::= '2#' bit \{'_\}' bit
bit ::= '1' | '0'
octal_integer ::= '8#' octal_digit \{'_\}' octal_digit
hex_integer ::= '16#' hex_digit \{'_\}' hex_digit
real_literal ::= signed_integer '.' integer \{exponent\}
exponent ::= ('E' | 'e') \{['+' | '-']\} integer

SEMANTICS: See 2.2.1.
B.1.2.2 Character strings

PRODUCTION RULES:
character_string ::= " ' " {character_representation} " ' "
character_representation ::= <any printable character except '$'> | ' $' hex_digit
hex_digit ::= ' 0' | ' 1' | ' 2' | ' 3' | ' 4' | ' 5' | ' 6' | ' 7' | ' 8' | ' 9'
| ' $' | ' $L' | ' $N' | ' $P' | ' $R' | ' $T' | ' $l' | ' $n' | ' $p' | ' $r' | ' $t'

SEMANTICS: See 2.2.2.

B.1.2.3 Time literals

PRODUCTION RULE:
time_literal ::= duration | time_of_day | date | date_and_time

SEMANTICS: See 2.2.3.

B.1.2.3.1 Duration

PRODUCTION RULES:
duration ::= ('T' | 't' | 'TIME' | 'time') '#' [' - '] interval
interval ::= days | hours | minutes | seconds | milliseconds
days ::= fixed_point ('d' | 'D') | integer ('d' | 'D') ['_.'] hours
fixed_point ::= integer [ '.' integer]
hours ::= fixed_point ('h' | 'H') | integer ('h' | 'H') ['_.'] minutes
minutes ::= fixed_point ('m' | 'M') | integer ('m' | 'M') ['_.'] seconds
seconds ::= fixed_point ('s' | 'S') | integer ('s' | 'S') ['_.'] milliseconds
milliseconds ::= fixed_point ('ms' | 'MS')

SEMANTICS: See 2.2.3.1.
NOTE - The semantics of 2.2.3.1 impose additional constraints on the allowable values of hours, minutes, seconds, and milliseconds.

B.1.2.3.2 Time of day and date

PRODUCTION RULES:
time_of_day ::= ('TIME_OF_DAY' | 'time_of_day' | 'TOD' | 'tod') '#' daytime
daytime ::= day_hour ':' day_minute ':' day_second
day_hour ::= integer
day_minute ::= integer
day_second ::= fixed_point
date ::= ('DATE' | 'date' | 'D' | 'd') '#' date_literal
date_literal ::= year '-' month '-' day
year ::= integer
month ::= integer
day ::= integer
date_and_time ::= ('DATE_AND_TIME' | 'date_and_time' | 'DT' | 'dt') '# date_literal '-' daytime
SEMANTICS: See 2.2.3.2.
NOTE - The semantics of 2.2.3.2 impose additional constraints on the allowable values of day_hour, day_minute, day_second, year, month, and day.

B.1.3 Data types

PRODUCTION RULES:

data_type_name ::= non_generic_type_name | generic_type_name
non_generic_type_name ::= elementary_type_name | derived_type_name

SEMANTICS: See 2.3.

B.1.3.1 Elementary data types

PRODUCTION RULES:

elementary_type_name ::= numeric_type_name | date_type_name | bit_string_type_name
| 'STRING' | 'TIME'
numeric_type_name ::= integer_type_name | real_type_name
integer_type_name ::= signed_integer_type_name | unsigned_integer_type_name
signed_integer_type_name ::= 'SINT' | 'INT' | 'DINT' | 'LINT'
unsigned_integer_type_name ::= 'USINT' | 'UINT' | 'UDINT' | 'ULINT'
real_type_name ::= 'REAL' | 'LREAL'
date_type_name ::= 'DATE' | 'TIME_OF_DAY' | 'TOD' | 'DATE_AND_TIME' | 'DT'
bit_string_type_name ::= 'BOOL' | 'BYTE' | 'WORD' | 'DWORD' | 'LWORD'

SEMANTICS: See 2.3.1.

B.1.3.2 Generic data types

PRODUCTION RULE:

generic_type_name ::= 'ANY' | 'ANY_NUM' | 'ANY_REAL' | 'ANY_INT' | 'ANY_BIT' | 'ANY_DATE'

SEMANTICS: See 2.3.2.

B.1.3.3 Derived data types

PRODUCTION RULES:

derived_type_name ::= single_element_type_name | array_type_name | structure_type_name | string_type_name
single_element_type_name ::= simple_type_name | subrange_type_name | enumerated_type_name
simple_type_name ::= identifier
subrange_type_name ::= identifier
enumerated_type_name ::= identifier
array_type_name ::= identifier
structure_type_name ::= identifier
data_type_declaration ::= 'TYPE' type_declaration ';';

't' END_TYPE

type_declaration ::= single_element_type_declaration | array_type_declaration
structure_type_declaration | string_type_declaration

single_element_type_declaration ::= simple_type_declaration | array_type_declaration

| enumerated_type_declaration

| subrange_type_declaration

| simple_type_declaration ::= simple_type_name ':' simple_spec_init

simple_spec_init ::= simple_specification ['=' constant]

simple_specification ::= elementary_type_name | simple_type_name

subrange_type_declaration ::= subrange_type_name ':' subrange_spec_init

| subrange_specification ::= integer_type_name '(' subrange')' | subrange_type_name

subrange ::= signed_integer '..' signed_integer

erenumerated_type_declaration ::= enumerated_type_name ':' enumerated_spec_init

enumerated_spec_init ::= enumerated_specification ['=' enumerated_value]

enumerated_specification ::= ( '(' enumerated_value {',' enumerated_value} ')' )

| enumerated_type_name

enumerated_value ::= identifier

array_type_declaration ::= array_type_name ':' array_spec_init

array_spec_init ::= array_specification ['=' array_initialization]

array_specification ::= array_type_name

| 'ARRAY' '[' subrange {','} subrange ']' 'OF' non_generic_type_name

array_initialization ::= [array_initial_elements {','} array_initialization]

array_initial_elements ::= array_initial_element | integer '(' array_initial_element ')

array_initial_element ::= constant | enumerated_value | structure_initialization | array_initialization

structure_type_declaration ::= structure_type_name ':' structure_specification

structure_specification ::= structure_declaration | initialized_structure

initialized_structure ::= structure_type_name [structure_initialization]

structure_declaration ::= 'STRUCT' structure_element_declaration ';';

{structure_element_declaration ';'} 'END_STRUCT'

structure_element_declaration ::= structure_element_name ':'

| simple_spec_init | subrange_spec_init | enumerated_spec_init | array_spec_init

| initialized_structure

| enumerated_type_name ::= identifier
structure_initialization ::= ( '(' structure_element_initialization {','}

structure_element_initialization ::= structure_element_name '::='

| constant | enumerated_value | array_initialization | structure_initialization

string_type_name ::= identifier

string_type_declaration ::= string_type_name '::=' 'STRING' '[' integer ']' ['='

| character_string]

SEMANTICS: See 2.3.3.
B.1.4 Variables

PRODUCTION RULES:

variable ::= direct_variable | symbolic_variable
symbolic_variable ::= variable_name | multi_element_variable
variable_name ::= identifier

SEMANTICS: See 2.4.1.

B.1.4.1 Directly represented variables

PRODUCTION RULES:

direct_variable ::= '%%' location_prefix size_prefix integer {'.' integer}
location_prefix ::= 'I' | 'Q' | 'M'
size_prefix ::= NIL | 'X' | 'B' | 'W' | 'D' | 'L'

SEMANTICS: See 2.4.1.1.

B.1.4.2 Multi-element variables

PRODUCTION RULES:

multi_element_variable ::= array_variable | structured_variable
array_variable ::= subscripted_variable subscript_list
subscripted_variable ::= symbolic_variable
subscript_list ::= '[' subscript {',' subscript} ']'
subscript ::= expression
structured_variable ::= record_variable '.' field_selector
record_variable ::= symbolic_variable
field_selector ::= identifier

SEMANTICS: See 2.4.1.2.

B.1.4.3 Declaration and initialization

PRODUCTION RULES:

input_declarations ::= 'VAR_INPUT' input_declaration '"' {input_declaration '"'} 'END_VAR'
input_declaration ::= var_init_decl | edge_declaration
edge_declaration ::= var1_list ':' 'BOOL' ['R_EDGE' | 'F_EDGE']
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
structured_var_init_decl ::= var1_list ':' initialized_structure
fb_name_decl ::= fb_name_list ':' function_block_type_name
fb_name_list ::= fb_name {',' fb_name}
fb_name ::= identifier
output_declarations ::= 'VAR_OUTPUT' ['RETAIN'] var_init_decl ';' {var_init_decl ';'}
'END_VAR'
in_input_output_declarations ::= 'VAR_IN_OUT' var_declaration ';' {var_declaration ';'}
'END_VAR'
var_declaration ::= var1_declaration | array_var_declaration |
structured_var_declaration
var1_declaration ::= var1_list ':' (simple_specification | subrange_specification | enumerated_specification)
array_var_declaration ::= var1_list ':' array_specification
structured_var_declaration ::= var1_list ':' structure_type_name
var_declarations ::= 'VAR' ['CONSTANT'] var_init_decl ';' {var_init_decl ';'}
'END_VAR'
retentive_var_declarations ::= 'VAR' 'RETAIN' var_init_decl ';' {var_init_decl ';'}
'END_VAR'
located_var_declarations ::= 'VAR' ['CONSTANT' | 'RETAIN']
located_var_decl ';' {located_var_decl ';'} 'END_VAR'
located_var_decl ::= [variable_name] location ':' located_var_spec_init
external_var_declarations ::= 'VAR_EXTERNAL' external_declaration ';'}
'END_VAR'

external_declaration ::= = global_var_name ':' (simple_specification |
simple_specification | subrange_specification | enumerated_specification | array_specification | structure_type_name |
function_block_type_name)
global_var_name ::= identifier
global_var_declarations ::= = 'VAR_GLOBAL' ['CONSTANT' | 'RETAIN']
global_var_dec ';' {global_var_dec ';' } 'END_VAR'
global_var_dec ::= global_var_spec ':' [located_var_spec_init
    | function_block_type_name ]
global_var_spec ::= global_var_list | [global_var_name] location
located_var_spec_init ::= simple_spec_init | subrange_spec_init |
    enumerated_spec_init | array_spec_init | initialized_structure

location ::= 'AT' direct_variable
global_var_list ::= global_var_name {',' global_var_name}
string_var_declaration ::= variable_name ':' 'STRING' [' INTEGER ' | ':' character_string ]

SEMANTICS: See 2.4.2. The non-terminal "function_block_type_name" is defined in B.1.5.2.
B.1.5 Program organization units

B.1.5.1 Functions

PRODUCTION RULES:

function_name ::= standard_function_name | derived_function_name
standard_function_name ::= <as defined in 2.5.1.5>
derived_function_name ::= identifier

function_declaration ::= 'FUNCTION' derived_function_name ':' (elementary_type_name | derived_type_name)
input_declarations
['VAR' ['CONSTANT']function_var_decls 'END_VAR']
function_body
'END_FUNCTION'

function_var_decls ::= function_var_decl ';' {function_var_decl ';'}
function_var_decl ::=
var1_declaration | array_var_declaration | structured_var_declaration

function_body ::= ladder_diagram | function_block_diagram | instruction_list | statement_list

SEMANTICS: See 2.5.1.

NOTES

1  This syntax does not reflect the fact that function block references and invocations are not allowed in function bodies.

2  Ladder diagrams and function block diagrams are graphically represented as defined in clause 4. The non-terminals instruction_list and statement_list are defined in B.2.1 and B.3.2, respectively.

B.1.5.2 Function blocks

PRODUCTION RULES:

function_block_type_name ::= standard_function_block_name | derived_function_block_name
standard_function_block_name ::= <as defined in 2.5.2.3>
derived_function_block_name ::= identifier

function_block_declaration ::= 'FUNCTION_BLOCK' derived_function_block_name
{fb_io_var_declarations}
{other_var_declarations}
function_block_body
'END_FUNCTION_BLOCK'

fb_io_var_declarations ::= input_declarations | output_declarations |
input_output_declarations

other_var_declarations ::= external_var_declarations | var_declarations |
retentive_var_declarations
function_block_body ::= sequential_function_chart | ladder_diagram |
function_block_diagram |
instruction_list | statement_list

SEMANTICS: See 2.5.2.

NOTES

1 Ladder diagrams and function block diagrams are graphically represented as defined in clause 4.

2 The non-terminals sequential_function_chart, instruction_list, and statement_list are defined in B.1.6, B.2, and B.3.2, respectively.

B.1.5.3 Programs

PRODUCTION RULES:

program_type_name ::= identifier

program_declaration ::= 
'PROGRAM' program_type_name
{fib_io_var_declarations}
{other_var_declarations | located_var_declarations | global_var_declarations}
[program_access_decls]
function_block_body
'END_PROGRAM'

program_access_decls ::= 
'VAR_ACCESS' program_access_decl ';'
{program_access_decl ';' }
'END_VAR'

program_access_decl ::= access_name ':' symbolic_variable ':'
non_generic_type_name [direction]

SEMANTICS: See 2.5.3.

B.1.6 Sequential function chart elements

PRODUCTION RULES:

sequential_function_chart ::= sfc_network {sfc_network}
sfc_network ::= initial_step {step | transition | action}
initial_step ::= 'INITIAL_STEP' step_name ':' {action_association ';'} 'END_STEP'
step ::= 'STEP' step_name ':' {action_association ';'} 'END_STEP'
step_name ::= identifier
action_association ::= action_name '(' action_qualifier {',' indicator_name} ')
action_name ::= identifier
action_qualifier ::= 'N' | 'R' | 'S' | 'P' | timed_qualifier ',', action_time
timed_qualifier ::= 'L' | 'D' | 'SD' | 'DS' | 'SL'
action_time ::= duration | variable_name
indicator_name ::= variable_name
transition ::= named_transition | unnamed_transition
transition_name ::= identifier
steps ::= step_name | '(' step_name ',' step_name {',' step_name} ')' transition_condition ::= ':' instruction_list | ':' expression ';' | ':' (fbd_network | rung)
unnamed_transition ::= ‘TRANSITION’ ‘FROM’ steps ‘TO’ steps transition_condition ‘END_TRANSITION’
named_transition ::= ‘TRANSITION’ transition_name transition_condition
action ::= ‘ACTION’ action_name ':'
  function_block_body
  'END_ACTION'

SEMANTICS: See 2.6. The use of function block diagram networks and ladder diagram rungs, denoted by the non-terminals fbd_network and rung, respectively, for the expression of transition conditions shall be as defined in 2.6.3.

NOTES

1 The non-terminals instruction_list and expression are defined in B.2.1 and B.3.1, respectively.

2 The construction named_transition can only be used when feature No.7 of table 41 is supported.

B.1.7 Configuration elements

PRODUCTION RULES:

configuration_name ::= identifier
resource_type_name ::= identifier
configuration_declaration ::= ‘CONFIGURATION’ configuration_name [global_var_declarations]
  (single_resource_declaration | (resource_declaration {resource_declaration})) ] [access_declarations]
  'END_CONFIGURATION'
resource_declaration ::= ‘RESOURCE’ resource_name ‘ON’ resource_type_name [global_var_declarations]
  single_resource_declaration
  'END_RESOURCE'
single_resource_declaration ::= {task_configuration ';' | program_configuration ';' | program_configuration ';
  {program_configuration ';
resource_name ::= identifier
access_declarations ::= ‘VAR_ACCESS’ access_declaration ';' {accessDeclaration ';
  'END_VAR'
access_declaration ::= access_name ':' access_path ':' non_generic_type_name [direction]
access_path ::= [resource_name '.' ] direct_variable | resource_name '.'
  program_io_reference
  | global_var_reference
global_var_reference ::= [resource_name '.' ] global_var_name ['.' structure_element_name]
access_name ::= identifier
program_io_reference ::= program_input_reference \| program_output_reference
program_output_reference ::= program_name '.' symbolic_variable
program_input_reference ::= program_name '.' symbolic_variable
program_name ::= identifier
direction ::= 'READ_WRITE' \| 'READ_ONLY'
task_configuration ::= 'TASK' task_name task_initialization
task_name ::= identifier
task_initialization ::= '(' ['SINGLE' ':' = data_source ','] ['INTERVAL' ':' = data_source ',']
'PRIORITY' ':' = integer ')
data_source ::= constant | global_var_reference | program_output_reference | direct_variable
program_configuration ::= 'PROGRAM' program_name ['WITH' task_name] ':'
program_type_name
['(' prog_conf_elements ')']
prog_conf_elements ::= prog_conf_element {',' prog_conf_element}
prog_conf_element ::= fb_task | prog_cnxn
fb_task ::= fb_name 'WITH' task_name
prog_cnxn ::= symbolic_variable ':' = prog_data_source | symbolic_variable '=>' data_sink
prog_data_source ::= constant | global_var_reference | direct_variable
data_sink ::= global_var_reference | direct_variable

SEMANTICS: See 2.7.

B.2 Language IL (Instruction List)

B.2.1 Instructions and operands

PRODUCTION RULES:

instruction_list ::= instruction {instruction}
instruction ::= [[label ':' ] (il_operation | il_fb_call)] EOL {EOL}
label ::= identifier
il_operation ::= il_operator [' ' il_operand_list]
il_operand_list ::= il_operand {',' il_operand}
il_operand ::= [identifier '='] (constant | variable)
il_fb_call ::= 'CAL' ['C' ['N']] fb_name ['(' il_operand_list ')']

SEMANTICS: See 3.2.

NOTE - The form of subscripts in the IL language is restricted to single-element variables or integer literals.

B.2.2 Operators

PRODUCTION RULES:
il_operator ::= ('LD' | 'ST' ) ['N'] | 'S' | 'R'
| ('AND' | 'OR' | 'XOR') ['N'] ['(']
| ('ADD' | 'SUB' | 'MUL' | 'DIV') ['(']
| ('GT' | 'GE' | 'EQ' | 'NE' | 'LT' | 'LE') ['(']
| ('JMP' | 'RET') ['C' ['N']] | 'S1' | 'R1' | 'CLK' | 'CU' | 'CD' | 'PV' | 'IN' | 'PT' | ' )'
| function_name

SEMANTICS: See 3.2.

B.3 Language ST (Structured Text)

B.3.1 Expressions

PRODUCTION RULES:

expression ::= xor_expression {'OR' xor_expression}
xor_expression ::= and_expression {'XOR' and_expression}
and_expression ::= comparison {'&' | 'AND') comparison
comparison ::= equ_expression { ('=' | '<>') equ_expression}
equ_expression ::= add_expression {comparison_operator add_expression}
comparison_operator ::= '<' | '>' | '<=' | '>='
add_expression ::= term {add_operator term}
add_operator ::= '+' | '-'
term ::= power_expression {multiply_operator power_expression}
multiply_operator ::= '*' | '/' | 'MOD'
power_expression ::= unary_expression {'**' unary_expression}
unary_expression ::= [unary_operator] primary_expression
unary_operator ::= '-' | 'NOT'
primary_expression ::= constant | variable | '(' expression ')
| function_name '(' [st_function_inputs] ')
st_function_inputs ::= st_function_input { ',' st_function_input}
st_function_input ::= [variable_name ':='] expression

SEMANTICS: These definitions have been arranged to show a top-down derivation of expression structure. The precedence of operations is then implied by a "bottom-up" reading of the definitions of the various kinds of expressions. Further discussion of the semantics of these definitions is given in 3.3.1.

B.3.2 Statements

PRODUCTION RULE:

statement_list ::= statement ';' {statement ';'}
statement ::= NIL | assignment_statement | subprogram_control_statement |
| selection_statement
| iteration_statement

SEMANTICS: See 3.3.2.
B.3.2.1 Assignment statements

PRODUCTION RULE:

assignment_statement ::= variable ':=' expression

SEMANTICS: See 3.3.2.1.

B.3.2.2 Subprogram control statements

PRODUCTION RULES:

subprogram_control_statement ::= fb_invocation | 'RETURN'

fb_invocation ::= fb_name '(' [fb_input_assignment {',' fb_input_assignment}] ')'  

fb_input_assignment ::= variable_name ':=' expression

SEMANTICS: See 3.3.2.2.

B.3.2.3 Selection statements

PRODUCTION RULES:

selection_statement ::= if_statement | case_statement

if_statement ::= 'IF' expression 'THEN' statement_list  

{ 'ELSIF' expression 'THEN' statement_list }  

{ 'ELSE' statement_list }  

'END_IF'

case_statement ::= 'CASE' expression 'OF'  

case_element { case_element }  

{ 'ELSE' statement_list }  

'END_CASE'

case_element ::= case_list ':' statement_list

case_list ::= case_list_element {',' case_list_element}

case_list_element ::= subrange | signed_integer

SEMANTICS: See 3.3.2.3.

B.3.2.4 Iteration statements

PRODUCTION RULES:

iteration_statement ::= for_statement | while_statement | repeat_statement |  

exit_statement

for_statement ::= 'FOR' control_variable ':=' for_list 'DO' statement_list 'END_FOR'

control_variable ::= identifier

for_list ::= expression 'TO' expression ['BY' expression]

while_statement ::= 'WHILE' expression 'DO' statement_list 'END_WHILE'

repeat_statement ::= 'REPEAT' statement_list 'UNTIL' expression 'END_REPEAT'

exit_statement ::= 'EXIT'

SEMANTICS: See 3.3.2.4.
ANNEX C - Delimiters and Keywords (normative)

The usages of delimiters and keywords in IEC 1131-3 is summarized in tables C.1 and C.2. National standards organizations can publish tables of translations for the textual portions of the delimiters listed in table C.1 and the keywords listed in table C.2.

Table C.1 - Delimiters

<table>
<thead>
<tr>
<th>Delimiters</th>
<th>Clause</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Space</td>
<td>2.1.4</td>
<td>As specified in 2.1.4.</td>
</tr>
<tr>
<td>(*)</td>
<td>2.1.5</td>
<td>Begin comment</td>
</tr>
<tr>
<td></td>
<td></td>
<td>End comment</td>
</tr>
<tr>
<td>+</td>
<td>2.2.1</td>
<td>Leading sign of decimal literal</td>
</tr>
<tr>
<td></td>
<td>3.3.1</td>
<td>Addition operator</td>
</tr>
<tr>
<td>–</td>
<td>2.2.1</td>
<td>Leading sign of decimal literal</td>
</tr>
<tr>
<td></td>
<td>2.2.3.2</td>
<td>Year-month-day separator</td>
</tr>
<tr>
<td></td>
<td>3.3.1</td>
<td>Subtraction, negation operator</td>
</tr>
<tr>
<td></td>
<td>4.1.1</td>
<td>Horizontal line</td>
</tr>
<tr>
<td>#</td>
<td>2.2.1</td>
<td>Based number separator</td>
</tr>
<tr>
<td></td>
<td>2.2.3</td>
<td>Time literal separator</td>
</tr>
<tr>
<td>.</td>
<td>2.2.1</td>
<td>Integer/fraction separator</td>
</tr>
<tr>
<td></td>
<td>2.4.1.1</td>
<td>Hierarchical address separator</td>
</tr>
<tr>
<td></td>
<td>2.4.1.2</td>
<td>Structure element separator</td>
</tr>
<tr>
<td></td>
<td>2.5.2.1</td>
<td>Function block structure separator</td>
</tr>
<tr>
<td>e or E</td>
<td>2.2.1</td>
<td>Real exponent delimiter</td>
</tr>
<tr>
<td>'</td>
<td>2.2.2</td>
<td>Start and end of character string</td>
</tr>
<tr>
<td>$</td>
<td>2.2.2</td>
<td>Start of special character in strings</td>
</tr>
</tbody>
</table>

2.2.3 - Time literal delimiters, including:
- t#, T#, d, D, h, H, m, M, s, S, ms, MS
- DATE#, date#, D#, d#, TIME_OF_DAY#, time_of_day#
- TOD#, tod#, DATE_AND_TIME#, date_and_time#, DT#, dt#

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2.2.3.2</td>
<td>Time of day separator</td>
<td></td>
</tr>
<tr>
<td>2.3.3.1</td>
<td>Type name/specification separator</td>
<td></td>
</tr>
<tr>
<td>2.4.2</td>
<td>Variable/type separator</td>
<td></td>
</tr>
<tr>
<td>2.6.2</td>
<td>Step name terminator</td>
<td></td>
</tr>
<tr>
<td>:</td>
<td>2.7</td>
<td>RESOURCE name/type separator</td>
</tr>
<tr>
<td></td>
<td>2.7</td>
<td>PROGRAM name/type separator</td>
</tr>
<tr>
<td></td>
<td>2.7</td>
<td>Access name/path/type separator</td>
</tr>
<tr>
<td></td>
<td>3.2.1</td>
<td>Instruction label terminator</td>
</tr>
<tr>
<td></td>
<td>4.1.2</td>
<td>Network label terminator</td>
</tr>
</tbody>
</table>

(continued on following page)
<table>
<thead>
<tr>
<th>Delimiters</th>
<th>Clause</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>:=</td>
<td>2.3.3.1</td>
<td>Initialization operator</td>
</tr>
<tr>
<td></td>
<td>2.7.1</td>
<td>Input connection operator</td>
</tr>
<tr>
<td></td>
<td>3.3.2.1</td>
<td>Assignment operator</td>
</tr>
</tbody>
</table>

- () 2.3.3.1 Enumeration list delimiters
- () 2.3.3.1 Subrange delimiters
- [] 2.4.1.2 Array subscript delimiters
- [] 2.4.2 String length delimiters
- () 2.4.2 Multiple initialization
- () 3.2.2 Instruction List modifier/operator
- () 3.3.1 Function arguments
- () 3.3.1 Subexpression hierarchy
- () 3.3.2.2 Function block input list delimiters

<table>
<thead>
<tr>
<th>Delimiters</th>
<th>Clause</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2.3.3.1</td>
<td>Enumeration list separator</td>
</tr>
<tr>
<td></td>
<td>2.3.3.2</td>
<td>Initial value separator</td>
</tr>
<tr>
<td></td>
<td>2.4.1</td>
<td>Array subscript separator</td>
</tr>
<tr>
<td></td>
<td>2.4.2</td>
<td>Declared variable separator</td>
</tr>
<tr>
<td>,</td>
<td>2.5.2.1</td>
<td>Function block initial value separator</td>
</tr>
<tr>
<td></td>
<td>2.5.2.1</td>
<td>Function block input list separator</td>
</tr>
<tr>
<td></td>
<td>3.2.1</td>
<td>Operand list separator</td>
</tr>
<tr>
<td></td>
<td>3.3.1</td>
<td>Function argument list separator</td>
</tr>
<tr>
<td></td>
<td>3.3.2.3</td>
<td>CASE value list separator</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Delimiters</th>
<th>Clause</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2.3.3.1</td>
<td>Type declaration separator</td>
</tr>
<tr>
<td></td>
<td>3.3</td>
<td>Statement separator</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Delimiters</th>
<th>Clause</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2.3.3.1</td>
<td>Subrange separator</td>
</tr>
<tr>
<td></td>
<td>3.3.2.3</td>
<td>CASE range separator</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Delimiters</th>
<th>Clause</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2.4.1.1</td>
<td>Direct representation prefix</td>
</tr>
<tr>
<td></td>
<td>2.7.1</td>
<td>Output connection operator</td>
</tr>
</tbody>
</table>

3.3.1 - Infix operators, including:

***, NOT, *, /, MOD, +, -, <, >, <=, =>, =, <>, &., AND, XOR, OR

<table>
<thead>
<tr>
<th>Delimiters</th>
<th>Clause</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>or</td>
<td>4.1.1</td>
</tr>
</tbody>
</table>

NOTE - “!” is only allowed when “|” does not exist in a national character set.
<table>
<thead>
<tr>
<th>Keywords</th>
<th>Clause</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTION...END_ACTION</td>
<td>2.6.4.1</td>
</tr>
<tr>
<td>ARRAY...OF</td>
<td>2.3.3.1</td>
</tr>
<tr>
<td>AT</td>
<td>2.4.3</td>
</tr>
<tr>
<td>CASE...OF...ELSE...END_CASE</td>
<td>3.3.2.3</td>
</tr>
<tr>
<td>CONFIGURATION...END_CONFIGURATION</td>
<td>2.7.1</td>
</tr>
<tr>
<td>CONSTANT</td>
<td>2.4.3</td>
</tr>
<tr>
<td>Data type names</td>
<td>2.3</td>
</tr>
<tr>
<td>EN, ENO</td>
<td>2.5.1.2</td>
</tr>
<tr>
<td>EXIT</td>
<td>3.3.2.4</td>
</tr>
<tr>
<td>FALSE</td>
<td>2.2.1</td>
</tr>
<tr>
<td>F_EDGE</td>
<td>2.5.2.2</td>
</tr>
<tr>
<td>FOR...TO...BY...DO...END_FOR</td>
<td>3.3.2.4</td>
</tr>
<tr>
<td>FUNCTION...END_FUNCTION</td>
<td>2.5.1.3</td>
</tr>
<tr>
<td>Function names</td>
<td>2.5.1</td>
</tr>
<tr>
<td>FUNCTION_BLOCK...END_FUNCTION_BLOCK</td>
<td>2.5.2.2</td>
</tr>
<tr>
<td>Function Block names</td>
<td>2.5.2</td>
</tr>
<tr>
<td>IF...THEN...ELSIF...ELSE...END_IF</td>
<td>3.3.2.3</td>
</tr>
<tr>
<td>INITIAL_STEP...END_STEP</td>
<td>2.6.2</td>
</tr>
<tr>
<td>PROGRAM...WITH...</td>
<td>2.7.1</td>
</tr>
<tr>
<td>PROGRAM...END_PROGRAM</td>
<td>2.5.3</td>
</tr>
</tbody>
</table>

(continued on following page)
<table>
<thead>
<tr>
<th>Keywords</th>
<th>Clause</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_EDGE</td>
<td>2.5.2.2</td>
</tr>
<tr>
<td>READ_ONLY, READ_WRITE</td>
<td>2.7.1</td>
</tr>
<tr>
<td>REPEAT...UNTIL...END_REPEAT</td>
<td>3.3.2.4</td>
</tr>
<tr>
<td>RESOURCE...ON...END_RESOURCE</td>
<td>2.7.1</td>
</tr>
<tr>
<td>RETAIN</td>
<td>2.4.3</td>
</tr>
<tr>
<td>RETURN</td>
<td>3.3.2.2</td>
</tr>
<tr>
<td>STEP...END_STEP</td>
<td>2.6.2</td>
</tr>
<tr>
<td>STRUCT...END_STRUCT</td>
<td>2.3.3.1</td>
</tr>
<tr>
<td>TASK</td>
<td>2.7.2</td>
</tr>
<tr>
<td>Textual operators (IL language)</td>
<td>3.2.2</td>
</tr>
<tr>
<td>(ST language)</td>
<td>3.3.1</td>
</tr>
<tr>
<td>TRANSITION...FROM...TO...END_TRANSITION</td>
<td>2.6.3</td>
</tr>
<tr>
<td>TRUE</td>
<td>2.2.1</td>
</tr>
<tr>
<td>TYPE...END_TYPE</td>
<td>2.3.3.1</td>
</tr>
<tr>
<td>VAR...END_VAR</td>
<td>2.4.2</td>
</tr>
<tr>
<td>VAR_INPUT...END_VAR</td>
<td></td>
</tr>
<tr>
<td>VAR_OUTPUT...END_VAR</td>
<td></td>
</tr>
<tr>
<td>VAR_IN_OUT...END_VAR</td>
<td></td>
</tr>
<tr>
<td>VAR_EXTERNAL...END_VAR</td>
<td></td>
</tr>
<tr>
<td>VAR_ACCESS...END_VAR</td>
<td>2.7.1</td>
</tr>
<tr>
<td>VAR_GLOBAL...END_VAR</td>
<td>2.7.1</td>
</tr>
<tr>
<td>WHILE...DO...END_WHILE</td>
<td>3.3.2.4</td>
</tr>
<tr>
<td>WITH</td>
<td>2.7.1</td>
</tr>
</tbody>
</table>

NOTE - The use of these keywords is restricted as defined in subclause 2.1.3 only within program organization units programmed in the respective languages.
ANNEX D - Implementation-dependent parameters (normative)

The implementation-dependent parameters defined in IEC 1131-3, and the primary reference clause for each, are listed in table D.1.

Table D.1 - Implementation-dependent parameters

<table>
<thead>
<tr>
<th>Clause</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.5.1</td>
<td>Error handling procedures</td>
</tr>
<tr>
<td>2.1.1</td>
<td>National characters used</td>
</tr>
<tr>
<td></td>
<td># or „pounds Sterling“ sign</td>
</tr>
<tr>
<td></td>
<td>$ or „currency“ sign</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>2.1.2</td>
<td>Maximum length of identifiers</td>
</tr>
<tr>
<td>2.1.5</td>
<td>Maximum comment length</td>
</tr>
<tr>
<td>2.2.3.1</td>
<td>Range of values of duration</td>
</tr>
<tr>
<td>2.3.1</td>
<td>Range of values for variables of type TIME</td>
</tr>
<tr>
<td></td>
<td>Precision of representation of seconds in types TIME_OF_DAY and DATE_AND_TIME</td>
</tr>
<tr>
<td>2.3.3</td>
<td>Maximum number of array subscripts</td>
</tr>
<tr>
<td></td>
<td>Maximum array size</td>
</tr>
<tr>
<td></td>
<td>Maximum number of structure elements</td>
</tr>
<tr>
<td></td>
<td>Maximum structure size</td>
</tr>
<tr>
<td></td>
<td>Maximum number of variables per declaration</td>
</tr>
<tr>
<td>2.3.3.1</td>
<td>Maximum number of enumerated values</td>
</tr>
<tr>
<td>2.3.3.2</td>
<td>Default maximum length of STRING variables</td>
</tr>
<tr>
<td></td>
<td>Maximum allowed length of STRING variables</td>
</tr>
<tr>
<td>2.4.1.1</td>
<td>Maximum number of hierarchical levels</td>
</tr>
<tr>
<td></td>
<td>Logical or physical mapping</td>
</tr>
<tr>
<td>2.4.1.2</td>
<td>Maximum number of subscripts</td>
</tr>
<tr>
<td></td>
<td>Maximum range of subscript values</td>
</tr>
<tr>
<td></td>
<td>Maximum number of levels of structures</td>
</tr>
<tr>
<td>2.4.2</td>
<td>Initialization of system inputs</td>
</tr>
<tr>
<td>2.4.3</td>
<td>Maximum number of variables per declaration</td>
</tr>
<tr>
<td>2.5</td>
<td>Information to determine execution times of program organization units</td>
</tr>
<tr>
<td>2.5.1.1</td>
<td>Method of function representation (names or symbols)</td>
</tr>
<tr>
<td>2.5.1.3</td>
<td>Maximum number of function specifications</td>
</tr>
<tr>
<td>2.5.1.5</td>
<td>Maximum number of inputs of extensible functions</td>
</tr>
<tr>
<td>2.5.1.5.1</td>
<td>Effects of type conversions on accuracy</td>
</tr>
</tbody>
</table>

(continued on following page)
<table>
<thead>
<tr>
<th>Clause</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.5.1.5.2</td>
<td>Accuracy of functions of one variable</td>
</tr>
<tr>
<td></td>
<td>Implementation of arithmetic functions</td>
</tr>
<tr>
<td>2.5.2</td>
<td>Maximum number of function block specifications and instantiations</td>
</tr>
<tr>
<td>2.5.2.3.3</td>
<td>( P_{\text{vmin}}, P_{\text{vmax}} ) of counters</td>
</tr>
<tr>
<td>2.5.2.3.4</td>
<td>Effect of a change in the value of a PT input during a timing operation</td>
</tr>
<tr>
<td>2.5.3</td>
<td>Program size limitations</td>
</tr>
<tr>
<td>2.6</td>
<td>Timing and portability effects of execution control elements</td>
</tr>
<tr>
<td>2.6.2</td>
<td>Precision of step elapsed time</td>
</tr>
<tr>
<td></td>
<td>Maximum number of steps per SFC</td>
</tr>
<tr>
<td>2.6.3</td>
<td>Maximum number of transitions per SFC and per step</td>
</tr>
<tr>
<td>2.6.4</td>
<td>Action control mechanism</td>
</tr>
<tr>
<td>2.6.4.2</td>
<td>Maximum number of action blocks per step</td>
</tr>
<tr>
<td>2.6.5</td>
<td>Graphic indication of step state</td>
</tr>
<tr>
<td></td>
<td>Transition clearing time</td>
</tr>
<tr>
<td></td>
<td>Maximum width of diverge/converge constructs</td>
</tr>
<tr>
<td>2.7.1</td>
<td>Contents of RESOURCE libraries</td>
</tr>
<tr>
<td>2.7.2</td>
<td>Maximum number of tasks</td>
</tr>
<tr>
<td></td>
<td>Task interval resolution</td>
</tr>
<tr>
<td></td>
<td>Pre-emptive or non-pre-emptive scheduling</td>
</tr>
<tr>
<td>3.3.1</td>
<td>Maximum length of expressions</td>
</tr>
<tr>
<td></td>
<td>Partial evaluation of Boolean expressions</td>
</tr>
<tr>
<td>3.3.2</td>
<td>Maximum length of statements</td>
</tr>
<tr>
<td>3.3.2.3</td>
<td>Maximum number of CASE selections</td>
</tr>
<tr>
<td>3.3.2.4</td>
<td>Value of control variable upon termination of FOR loop</td>
</tr>
<tr>
<td>4.1.1</td>
<td>Graphic/semigraphic representation</td>
</tr>
<tr>
<td></td>
<td>Restrictions on network topology</td>
</tr>
<tr>
<td>4.1.3</td>
<td>Evaluation order of feedback loops</td>
</tr>
<tr>
<td>4.3.3</td>
<td>Means for specifying order of network execution</td>
</tr>
</tbody>
</table>
ANNEX E - Error Conditions (normative)

The error conditions defined in IEC 1131-3, and the primary reference clause for each, are listed in table E.1. These errors may be detected during preparation of the program for execution or during execution of the program. The manufacturer shall specify the disposition of these errors according to the provisions of subclause 1.5.1 of this part.

Table E.1 - Error conditions

<table>
<thead>
<tr>
<th>Clause</th>
<th>Error conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.3.3.1</td>
<td>Value of a variable exceeds the specified subrange</td>
</tr>
<tr>
<td>2.4.2</td>
<td>Length of initialization list does not match number of array entries</td>
</tr>
<tr>
<td>2.5.1</td>
<td>Improper use of directly represented or external variables in functions</td>
</tr>
<tr>
<td>2.5.1.5.1</td>
<td>Type conversion errors</td>
</tr>
<tr>
<td>2.5.1.5.2</td>
<td>Numerical result exceeds range for data type Division by zero</td>
</tr>
<tr>
<td>2.5.1.5.4</td>
<td>Mixed input data types to a selection function Selector (K) out of range for MUX function</td>
</tr>
<tr>
<td>2.5.1.5.5</td>
<td>Invalid character position specified Result exceeds maximum string length</td>
</tr>
<tr>
<td>2.5.1.5.5</td>
<td>CONCAT result too long</td>
</tr>
<tr>
<td>2.5.1.5.6</td>
<td>Result exceeds range for data type</td>
</tr>
<tr>
<td>2.5.2.2</td>
<td>No parameter value specified for a function block instance used as input parameter</td>
</tr>
<tr>
<td>2.5.2.2</td>
<td>No parameter value specified for a VAR_IN_OUT parameter</td>
</tr>
<tr>
<td>2.6.2</td>
<td>Zero or more than one initial steps in SFC network User program attempts to modify step state or time</td>
</tr>
<tr>
<td>2.6.2.5</td>
<td>Simultaneously true, non-prioritized transitions in a selection divergence</td>
</tr>
<tr>
<td>2.6.3</td>
<td>Side effects in evaluation of transition condition</td>
</tr>
<tr>
<td>2.6.4.5</td>
<td>Action control contention error</td>
</tr>
<tr>
<td>2.6.5</td>
<td>Unsafe or unreachable SFC</td>
</tr>
<tr>
<td>2.7.1</td>
<td>Data type conflict in VAR_ACCESS</td>
</tr>
<tr>
<td>2.7.2</td>
<td>Tasks require too many processor resources Execution deadline not met Other task scheduling conflicts</td>
</tr>
<tr>
<td>3.2.2</td>
<td>Numerical result exceeds range for data type</td>
</tr>
<tr>
<td>3.2.2</td>
<td>Current result and operand not of same data type</td>
</tr>
<tr>
<td>3.3.1</td>
<td>Division by zero Invalid data type for operation</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>-------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>3.3.2.1</td>
<td>Return from function without value assigned</td>
</tr>
<tr>
<td>3.3.2.4</td>
<td>Iteration fails to terminate</td>
</tr>
<tr>
<td>4.1.1</td>
<td>Same identifier used as connector label and element name</td>
</tr>
<tr>
<td>4.1.3</td>
<td>Uninitialized feedback variable</td>
</tr>
</tbody>
</table>