2.6 Sequential Function Chart (SFC) elements

2.6.1 General

This subclause defines *sequential function chart* (SFC) elements for use in structuring the internal organization of a programmable controller program organization unit, written in one of the languages defined in this standard, for the purpose of performing *sequential control* functions. The definitions in this subclause are derived from IEC 848, with the changes necessary to convert the representations from a *documentation standard* to a set of *execution control elements* for a programmable controller program organization unit.

The SFC elements provide a means of partitioning a programmable controller program organization unit into a set of *steps* and *transitions* interconnected by *directed links*. Associated with each step is a set of *actions*, and with each transition is associated a *transition condition*.

Since SFC elements require storage of state information, the only program organization units which can be structured using these elements are *function blocks* and *programs*.

If any part of a program organization unit is partitioned into SFC elements, the entire program organization unit shall be so partitioned. If no SFC partitioning is given for a program organization unit, the entire program organization unit shall be considered to be a single *action* which executes under the control of the invoking entity.

2.6.2 Steps

A *step* represents a situation in which the behavior of a program organization unit with respect to its inputs and outputs follows a set of rules defined by the associated *actions* of the step. A step is either *active* or *inactive*. At any given moment, the state of the program organization unit is defined by the set of active steps and the values of its internal and output variables.

As shown in table 40, a step shall be represented graphically by a block containing a *step name* in the form of an identifier as defined in 2.1.2, or textually by a STEP...END_STEP construction. The directed link(s) into the step can be represented graphically by a vertical line attached to the top of the step. The directed link(s) out of the step can be represented by a vertical line attached to the bottom of the step. Alternatively, the directed links can be represented textually by the TRANSITION... END_TRANSITION construction defined in 2.6.3.
The *step flag* (active or inactive state of a step) can be represented by the logic value of a Boolean structure element ***.X, where *** is the step name, as shown in table 40. This Boolean variable has the value "1" when the corresponding step is active, and "0" when it is inactive. The state of this variable is available for graphical connection at the right side of the step as shown in table 40.
Similarly, the elapsed time, ***.T, since initiation of a step can be represented by a structure element of type TIME, as shown in table 40. When a step is deactivated, the value of the step elapsed time shall remain at the value it had when the step was deactivated. When a step is activated, the value of the step elapsed time shall be reset to t#0s.
The *scope* of step names, step flags, and step times shall be *local* to the program organization unit in which the steps appear.

The initial state of the program organization unit is represented by the initial values of its internal and output variables, and by its set of *initial steps*, i.e., the steps which are initially active. Each SFC *network*, or its textual equivalent, shall have exactly one initial step.
An initial step can be drawn graphically with double lines for the borders, and with the ISO 646 character set shall be drawn as shown in table 40.
For system initialization as defined in 2.4.2, the default initial elapsed time for steps is t#0s, and the default initial state is Boolean 0 for ordinary steps and Boolean 1 for initial steps. However, when an instance of a function block or a program is declared to be *retentive* (for instance, as in feature 3 of table 33), the states and (if supported) elapsed times of all steps contained in the program or function block shall be treated as retentive for system initialization as defined in 2.4.2.

Table 40 - Step features

No.	REPRESENTATION	DESCRIPTION

1	`|`
	`+-----+`
	`| *** |`
	`+-----+`
	`|`
		Step - Graphical form
		with directed links
		"***" = step name

	`|`
	`+=======+`
	`|| *** ||`
	`||     ||`
	`+=======+`
	`|`
		Initial step - Graphical form with directed links
		"***" = Name of initial step
		(note 2)

2	`STEP *** :`
	`  (* Step body *)`
	`END_STEP`
		Step - Textual form
		without directed links (see 2.6.3)
		"***" = Step name

	`INITIAL_STEP *** :`
	`  (* Step body *)`
	`END_STEP`
		Initial step - Textual form
		without directed links (see 2.6.3)
		"***" = Name of initial step

3a	`***.X`	Step flag - General form "***" = Step name ***.X = Boolean 1 when *** is active, Boolean 0 otherwise

3b	`|`
	`+-----+`
	`| *** |----`
	`+-----+`
	`|`
		Step flag - Direct connection
		of Boolean variable ***.X to
		right side of step "***"

4	`***.T`	Step elapsed time - General form "***" = Step name ***.T = A variable of type TIME (See 2.6.2)

NOTES

1. When feature 3a, 3b, or 4 is supported, it shall be an *error* if the user program attempts to modify the associated variable. For example, if S4 is a step name, then the following statements would be *errors* in the ST language defined in 3.3:

```
S4.X := 1 ; (* ERROR *)
S4.T := t#100ms ; (* ERROR *)
```

2. The upper directed link is not required if the initial step has no predecessors.

2.6.3 Transitions

A *transition* represents the condition whereby control passes from one or more steps preceding the transition to one or more successor steps along the corresponding directed link. The transition shall be represented by a horizontal line across the vertical directed link.
The direction of evolution following the directed links shall be from the bottom of the predecessor step(s) to the top of the successor step(s).
Each transition shall have an associated *transition condition* which is the result of the evaluation of a single Boolean expression. A transition condition which is always true shall be represented by the symbol "1" or the keyword TRUE.
A transition condition can be associated with a transition by one of the following means, as shown in table 41:

1) By placing the appropriate Boolean expression in the ST language defined in 3.3 to the right of the vertical directed link.

2) By a ladder diagram network in the LD language defined in 4.2, whose output intersects the vertical directed link instead of a right rail.

3) By a network in the FBD language defined in 4.3, whose output intersects the vertical directed link.

4) By a LD or FBD network whose output intersects the vertical directed link via a *connector* as defined in 4.1.1.

5) By a TRANSITION...END_TRANSITION construct using the ST language. This shall consist of:

- The keywords TRANSITION FROM followed by the step name of the predecessor step (or, if there is more than one predecessor, by a parenthesized list of predecessor steps);
- The keyword TO followed by the step name of the successor step (or, if there is more than one successor, by a parenthesized list of successor steps);
- The assignment operator (:=), followed by a Boolean expression in the ST language, specifying the transition condition;
- The terminating keyword END_TRANSITION.

6) By a TRANSITION...END_TRANSITION construct using the IL language defined in 3.2. This shall consist of:

- The keywords TRANSITION FROM followed by the step name of the predecessor step (or, if there is more than one predecessor, by a parenthesized list of predecessor steps), followed by a colon (':');
- The keyword TO followed by the step name of the successor step (or, if there is more than one successor, by a parenthesized list of successor steps);
- Beginning on a separate line, a list of instructions in the IL language, the result of whose evaluation determines the transition condition;
- The terminating keyword END_TRANSITION on a separate line.

7) By the use of a *transition name* in the form of an identifier to the right of the directed link. This identifier shall refer to a TRANSITION...END_TRANSITION construction defining one of the following entities, whose evaluation shall result in the assignment of a Boolean value to the variable denoted by the transition name:

- A network in the LD or FBD language;

- A list of instructions in the IL language;
- An assignment of a Boolean expression in the ST language.

The *scope* of a transition name shall be *local* to the program organization unit in which the transition is located.
It shall be an *error* in the sense of 1.5.1 if any "side effect" (for instance, the assignment of a value to a variable other than the transition name) occurs during the evaluation of a transition condition.

Table 41 - Transitions and transition conditions

No.	EXAMPLE	DESCRIPTION

1	`      |`
	`  +------+`
	`  |STEP7|`
	`  +------+`
	`      |`
	`      + %IX2.4 & %IX2.3`
	`      |`
	`  +------+`
	`  |STEP8|`
	`  +------+`
	`      |`
		Predecessor step
		
		Transition condition using ST language (see 3.3)
		
		Successor step

2	`                          |`
	`                      +------+`
	`                      |STEP7|`
	`                      +------+`
	`| %IX2.4  %IX2.3         |`
	`+---||-----||---------+`
	`|                        |`
	`                      +------+`
	`                      |STEP8|`
	`                      +------+`
	`                          |`
		Predecessor step
		
		Transition condition using LD language (see 4.2)
		
		Successor step

3	`                          |`
	`                      +------+`
	`                      |STEP7|`
	`         +--------+   +------+`
	`         |   &    |       |`
	`%IX2.4---|        |------+`
	`%IX2.3---|        |       |`
	`         +--------+   +------+`
	`                      |STEP8|`
	`                      +------+`
	`                          |`
		Predecessor step
		
		Transition condition using FBD language (see 4.3)
		
		Successor step

(continued on following page)

Table 41 - Transitions and transition conditions *(continued)*

No.	**Example**	**Description**

4	`                        |`
	`                     +------+`
	`                     |STEP7|`
	`                     +------+`
	`                        |`
	`    >TRANX>-------------+`
	`                        |`
	`                     +------+`
	`                     |STEP8|`
	`                     +------+`
	`                        |`
		Use of connector:
		
		Predecessor step
		
		Transition connector
		
		Successor step

4a
 	
4b
	`    | %IX2.4  %IX2.3`
	`    +---| |-----| |---->TRANX>`
	`    |`
	`           +--------+`
	`           |   &    |`
	`  %IX2.4---|        |-->TRANX>`
	`  %IX2.3---|        |`
	`           +--------+`
		Transition condition:
		Using LD language
		(see 4.2)
		
		Using FBD language
		(see 4.3)

5	`STEP STEP7: END_STEP`
	`TRANSITION FROM STEP7 TO STEP8`
	`  := %IX2.4 & %IX2.3 ;`
	`END_TRANSITION`
	`STEP STEP8: END_STEP`
		Textual equivalent
		of feature 1
		using ST language
		(see 3.3)

6	`STEP STEP7: END_STEP`
	`TRANSITION FROM STEP7 TO STEP 8:`
	`  LD  %IX2.4`
	`  AND %IX2.3`
	`END_TRANSITION`
	`STEP STEP8: END_STEP`
		Textual equivalent
		of feature 1
		using IL language
		(see 3.2)

(continued on following page)

Table 41 - Transitions and transition conditions *(continued)*

No.	Example	Description

7	`    |`
	`+------+`
	`|STEP7|`
	`+------+`
	`    |`
	`    + TRAN78`
	`    |`
	`+------+`
	`|STEP8|`
	`+------+`
	`    |`
		Use of transition name:
		
		Predecessor step
		
		Transition name
		
		Successor step

7a	`TRANSITION TRAN78:`
	`  |                         |`
	`  | %IX2.4  %IX2.3  TRAN78 |`
	`  +---||-----||------( )---+`
	`  |                         |`
	` END_TRANSITION`
		Transition condition using LD language (see 4.2)

7b	`TRANSITION TRAN78:`
	`          +--------+`
	`          |   &    |`
	` %IX2.4---|        |--TRAN78`
	` %IX2.3---|        |`
	`          +--------+`
	`END_TRANSITION`
		Transition condition using FBD language (see 4.3)

7c	`TRANSITION TRAN78:`
	`      LD   %IX2.4`
	`      AND  %IX2.3`
	`END_TRANSITION`
		Transition condition using IL language (see 3.2)

7d	`TRANSITION TRAN78`
	`  := %IX2.4 & %IX2.3 ;`
	`END_TRANSITION`
		Transition condition using ST language (see 3.3)

NOTES 1. If feature 1 of table 40 is supported, then one or more of features 1, 2, 3, 4, or 7 of this table shall be supported. 2. If feature 2 of table 40 is supported, then feature 5 or 6 of this table, or both, shall be supported.		

2.6.4 Actions

Zero or more *actions* shall be associated with each step. A step which has zero associated actions shall be considered as having a WAIT function, that is, waiting for a successor transition condition to become true.
An action can be a Boolean variable, a collection of *instructions* in the IL language defined in 3.2, a collection of *statements* in the ST language defined in 3.3, a collection of *rungs* in the LD language defined in 4.2, a collection of *networks* in the FBD language defined in 4.3, or a *sequential function chart* (SFC) organized as defined in this subclause (2.6).
Actions shall be declared via one or more of the mechanisms defined in 2.6.4.1, and shall be associated with steps via textual *step bodies* or graphical *action blocks,* as defined in 2.6.4.2. The details of action block representation are defined in 2.6.4.3. Control of actions shall be expressed by *action qualifiers* as defined in 2.6.4.4.

2.6.4.1 Declaration

A programmable controller implementation which supports SFC elements shall provide one or more of the mechanisms defined in table 42 for the declaration of actions. The *scope* of the declaration of an action shall be *local* to the program organization unit containing the declaration.

Table 42 - Declaration of actions

No.	Feature
1	Any Boolean variable declared in a VAR or VAR_OUTPUT block, or their graphical equivalents, can be an action.

	Example	**Feature**

2l

```
+-----------------------------------------+
|                ACTION_4                 |
+-----------------------------------------+
|     |  %IX1   %MX3  S8.X  %QX17  |      |
|     +---||-----||----||-----()---+      |
|     |                            |      |
|     |    +------+                |      |
|     +----|EN ENO|         %MX10  |      |
|     | C--|  LT  |----------(S)---+      |
|     | D--|      |                |      |
|     |    +------+                |      |
+-----------------------------------------+
```

Graphical declaration in LD language (see 4.2)

2s

```
+-----------------------------------------+
|             OPEN_VALVE_1                |
+-----------------------------------------+
|          | ...                          |
| +==================+                    |
| || VALVE_1_READY ||                     |
| +==================+                    |
|          |                              |
|          + STEP8.X                      |
|          |                              |
| +------------------+  +---+-----------+ |
| | VALVE_1_OPENING  |--| N |VALVE_1_FWD| |
| +------------------+  +---+-----------+ |
|          | ...                          |
+-----------------------------------------+
```

Inclusion of SFC elements in action

2f

```
+-----------------------------------------+
|                ACTION_4                 |
+-----------------------------------------+
|                +---+                    |
|          %IX1--| & |                    |
|          %MX3--|   |--%QX17             |
|   S8.X---------|   |                    |
|                +---+    FF28            |
|                        +----+           |
|                        | SR |           |
|              +------+  |  Q1|-%MX10     |
|           C--|  LT  |--|S1  |           |
|           D--|      |  +----+           |
|              +------+                   |
+-----------------------------------------+
```

Graphical declaration in FBD language (see 4.3)

(continued on following page)

Table 42 - Declaration of actions *(continued)*

No.	Example	Feature

3s	
	```
	ACTION ACTION_4:
	  %QX17 := %IX1 & %MX3 & S8.X ;
	  FF28(S1 := (C<D));
	  %MX10 := FF28.Q;
	END_ACTION
	```
		Textual declaration in ST language (see 3.3)

3i	
	```
	ACTION     ACTION_4:
	LD         S8.X
	AND        %IX1
	AND        %MX3
	ST         %QX17
	LD         C
	LT         D
	S1         FF28
	LD         FF28.Q
	ST         %MX10
	END_ACTION
	```
		Textual declaration in IL language (see 3.2)

NOTES

1 - The step flag S8.X is used in these examples to obtain the desired result that, when S8 is deactivated, %QX17 := 0.

2 - If feature 1 of table 40 is supported, then one or more of the features in this table, or feature 4 of table 43, shall be supported.

3 - If feature 2 of table 40 is supported, then one or more of features 1,3s, or 3i of this table shall be supported.

2.6.4.2 Association with steps

A programmable controller implementation which supports SFC elements shall provide one or more of the mechanisms defined in table 43 for the association of actions with steps.

Table 43 - Step/action association

No.	Example	Feature

1	`    |`
	`  +----+  +-----+----------+---+`
	`  | S8 |--|  L  | ACTION_1 |DN1|`
	`  +----+  |t#10s|          |   |`
	`    |     +-----+----------+---+`
	`    + DN1`
	`    |`
		Action block (see 2.6.4.3)

2	`    |`
	`  +----+  +-----+----------------------+---`
	`+`
	`  | S8 |--|  L  |      ACTION_1`
	`|DN1|`
	`  +----+  |t#10s|                      |`
	`|`
	`    |     +-----+----------------------+---`
	`+`
	`   +DN1   |  P  |      ACTION_2        |`
	`|`
	`    |     +-----+----------------------+---`
	`+`
	`    |     |  N  |      ACTION_3        |`
	`|`
	`    |     +-----+----------------------+---`
	`+`
		Concatenated action blocks

3	`STEP S8:`
	`  ACTION_1(L,t#10s,DN1) ;`
	`  ACTION_2(P) ;`
	`  ACTION_3(N) ;`
	`END_STEP`
		Textual step body

4	`    +-----+----------------------+---+`
	`----| N   |      ACTION_4        |   |---`
	`    +-----+----------------------+---+`
	`    |  %QX17 := %IX1 & %MX3 & S8.X ; |`
	`    |  FF28 (S1 := (C<D));            |`
	`    |  %MX10 := FF28.Q;               |`
	`    +--------------------------------+`
		Action block "d" Field (see 2.6.4.3)

NOTE - When feature 4 is used, the corresponding action name cannot be used in any other action block.		

2.6.4.3 Action blocks

As shown in table 44, an *action block* is a graphical element for the combination of a Boolean variable with one of the *action qualifiers* specified in subclause 2.6.4.4 to produce an enabling condition, according to the rukes given in subclause 2.6.4.5, par an associated action.
The action block provides a means of optionally specifying Boolean "indicator" variables, indicated by the "c" field in table 44, which can be set by the specified action to indicate its completion, timeout, error conditions, etc. If the "c" field is not present, and the "b" field specifies that the action shall be a Boolean variable, then this variable shall be interpreted as the "c" variable when required.
When action blocks are concatenated graphically as illustrated in table 43, such concatenations can have multiple indicator variables, but shall have only a single common Boolean input variable, which shall act simultaneously upon all the concat-enated blocks.
As well as being associated with a step, an action block can be used as a graphical element in the LD or FBD languages specified in clause 4. In this case, signal or power flow through an action block shall follow the rules specified in 4.1.1.

Table 44 - Action block features

No.	**Feature**	**Graphical form**

1
2
3
 	
4
5
6
7
	"a" : Qualifier as per 2.6.4.4
	"b" : Action name
	"c" : Boolean "indicator" variables
	"d" : Action using:
	IL language (3.2)
	ST language (3.3)
	LD language (4.2)
	FBD language (4.3)
		`+-----+---------------+-----+`
		`---| "a" |      "b"      | "c"`
		`|---`
		`+-----+---------------+-----+`
		`|            "d"            |`
		`|                           |`
		`+---------------------------+`

No.	**Feature/Example**	

8	Use of action blocks in ladder diagrams (see 4.2):
	`|   S8.X   %IX7.5   +---+------+---+   OK1   |`
	`+--| |----| |----| N | ACT1 |DN1|--( )--+`
	`|                   +---+------+---+         |`	

9	Use of action blocks in function block diagrams (see 4.3):
	`          +---+      +---+------+-----+`
	`   S8.X---| & |-----| N | ACT1 | DN1 |---OK1`
	`%IX7.5---|   |      +---+------+-----+`
	`          +---+`	

NOTES 1. Field "a" can be omitted when the qualifier is "N". 2. Field "c" can be omitted when no feedback variable is used.		

2.6.4.4 Action qualifiers

Associated with each step/action association defined in 2.6.4.2, or each occurrence of an action block as defined in 2.6.4.3, shall be an *action qualifier*. The value of this qualifier shall be one of the values listed in table 45. In addition, the qualifiers L, D, SD, DS, and SL shall have an associated duration of type TIME.

NOTE - IEC 848 gives informal definitions and examples of the use of these qualifiers. This standard formalizes these definitions, redefining the "S" qualifier and introducing the "R" qualifier. The control of actions using these qualifiers is defined in the following subclause, and additional examples of their use are given in annex F.

Table 45 - Action qualifiers

No.	Qualifier	Explanation
1	None	Non-stored (null qualifier)
2	N	**N**on-stored
3	R	overriding **R**eset
4	S	**S**et (**S**tored)
5	L	time **L**imited
6	D	time **D**elayed
7	P	**P**ulse
8	SD	**S**tored and time **D**elayed
9	DS	**D**elayed and **S**tored
10	SL	**S**tored and time **L**imited

2.6.4.5 Action control

The control of actions shall be functionally equivalent to the application of the following rules:

1) Associated with each action shall be the functional equivalent of an instance of the ACTION_CONTROL function block defined in figures 14 and 15. If the action is declared as a Boolean variable, as defined in 2.6.4.1, the "Q" output of this block shall be the state of this Boolean variable. If the action is declared as a collection of statements or networks, as defined in 2.6.4.1, then this collection shall be executed upon each invocation of the program organisation unit (POU) in which the action is contained while the "Q" output of the ACTION_CONTROL function block stands at Boolean 1. The statements or networks shall be executed one final time after the falling edge of "Q".

2) A Boolean input to the ACTION_CONTROL block for an action shall be said to have an *association* with a step as defined in 2.6.4.2, or with an action block as defined in 2.6.4.3, if the corresponding qualifier is equivalent to the input name (N, R, S, L, D, P, SD, DS, or SL). The association shall be said to be *active* if the associated step is active, or if the associated action block's input has the value Boolean 1. The *active associations* of an *action* are equivalent to the set of *active associations* of all inputs to its ACTION_CONTROL function block.

A Boolean input to an ACTION_CONTROL block shall have the value Boolean 1 if it has at least one active association, and the value Boolean 0 otherwise.

3) The value of the T input to an ACTION_CONTROL block shall be the value of the duration portion of a time-related qualifier (L, D, SD, DS, or SL) of an active association. If no such association exists, the value of the T input shall be t#0s.

4) It shall be an *error* in the sense of subclause 1.5.1 if one or more of the following conditions exist:

a) More than one *active association* of an action has a time-related qualifier (L, D, SD, DS, or SL).

b) The SD input to an ACTION_CONTROL block has the Boolean value 1 when the Q1 output of its SL_FF block has the Boolean value 1.

c) The SL input to an ACTION_CONTROL block has the Boolean value 1 when the Q1 output of its SD_FF block has the Boolean value 1.

5) It is not required that the ACTION_CONTROL block itself be implemented, but only that the control of actions be equivalent to the preceding rules. Only those portions of the action control appropriate to a particular action need be instantiated, as illustrated in figure 16. In particular, note that simple MOVE (:=) and Boolean OR functions suffice for control of Boolean variable actions if the latter's associations have only "N" qualifiers.

```
       +----------------+
       | ACTION_CONTROL |
BOOL---|N              Q|---
BOOL
BOOL---|R               |
BOOL---|S               |
BOOL---|L               |
BOOL---|D               |
BOOL---|P               |
BOOL---|SD              |
BOOL---|DS              |
BOOL---|SL              |
TIME---|T               |
       +----------------+
```

Figure 14 - ACTION_CONTROL function block - External interface (Not visible to the user)

```
+----+
   +-----------------------------------------------------------
O| & |---Q
   |                                                 +-----+
|    |
N--|---------------------------------------------| >=1 |-
-|    |
   |                        S_FF                  |     |
+----+
R--+                       +----+                 |     |
   |                       | RS |                 |     |
S--|-----------------------|S Q1|-----------------|     |
   +-----------------------|R1  |                 |     |
   |                       +----+  +---+          |     |
L--|----------+--------------------| & |----------|     |
   |          |        L_TMR    +--O|   |          |     |
   |          |       +-----+   |   +---+          |     |
   |          |       | TON |   |                  |     |
   |          +-------|IN  Q|---+       D_TMR      |     |
   |  +---------------|PT   |          +-----+     |     |
   |  |               +-----+          | TON |     |     |
D--|--|--------------------------------|IN  Q|-----|     |
   |  +--------------------------------|PT   |     |     |
   |  |             P_TRIG             +-----+     |     |
   |  |           +---------+                      |     |
   |  |           | R_TRIG |                       |     |
P--|--|-----------|CLK    Q|-----------------------|     |
   |  |     SD_FF   +---------+   SD_TMR            |     |
   |  |    +----+                +-----+           |     |
   |  |    | RS |                | TON |           |     |
SD-|--|----|S Q1|----------------|IN  Q|-----------|     |
   +--|----|R1  |    +-----------|PT   |           |     |
   |  |    +----+    |   DS_TMR  +-----+  DS_FF    |     |
   |  +--------------+   +-----+          +----+   |     |
   |  |                  | TON |          | RS |   |     |
DS-|--|------------------|IN  Q|----------|S Q1|---|     |
   |  +------------------|PT   |     +----|R1  |   |     |
   |  |                  +-----+     |    +----+   |     |
   +--|------------------------------+             |     |
   |  |          SL_FF                              |     |
   |  |         +----+                              |     |
   |  |         | RS |                   +---+      |     |
SL-|--|---------|S Q1|--+----------------| & |--|   |
   +--|---------|R1  |  |     SL_TMR  +--O|   |  +-----+
      |         +----+  |    +-----+    |   +---+
      |                 |    | TON |    |
      |                 +----|IN  Q|----+
T-----+----------------------|PT   |
                             +-----+
```

Figure 15 - ACTION_CONTROL function block body (not visible to the user)

```
        |
     +-----+   +---+------------+----------------+
     | S22 |---| N | HV_BREAKER | HV_BRKR_CLOSED |
     +-----+   +---+------------+----------------+
        |      | S | START_INDICATOR              |
        |      +---+-----------------------------+
        + HV_BRKR_CLOSED
        |
     +-----+   +----+---------------+
     | S23 |---| SL | RUNUP_MONITOR |
     +-----+   |t#1m|               |
        |      +----+---------------+
        |      | D  | START_WAIT    |
        |      |t#1s|               |
        |      +----+---------------+
        + START_WAIT
        |
     +-----+   +-----+------------------+-----------------
-+
     | S24 |---| N   | ADVANCE_STARTER | STARTER_ADVANCED
|
     +-----+   +-----+------------------+-----------------
-+
        |      | L   | START_MONITOR
|
        |      |t#30s|
|
        |      +-----+------------------------------------
-+
        + STARTER_ADVANCED
        |
     +-----+   +-----+------------------+-----------------
--+
     | S26 |---| N   | RETRACT_STARTER |
STARTER_RETRACTED |
     +-----+   +-----+------------------+-----------------
--+
        |
        |
        + STARTER_RETRACTED
        |
     +-----+   +-----+-----------------+
     | S27 |---| R   | START_INDICATOR |
     +-----+   +-----+-----------------+
        |      | R   | RUNUP_MONITOR   |
        |      +-----+-----------------+
```

NOTE - The complete SFC network and its associated declarations are not shown in this example.

Figure 16a - Action control example - SFC representation

```
S22.X---------------------------------------------------
HV_BREAKER
S24.X----------------------------------------------
ADVANCE_STARTER
S26.X----------------------------------------------
RETRACT_STARTER

                      START_INDICATOR_S_FF
                            +----+
                            | RS |
S22.X-----------------------|S Q1|-----------------
START_INDICATOR
S27.X-----------------------|R1  |
                            +----+

                       START_WAIT_D_TMR
                            +-----+
                            | TON |
S23.X-----------------------|IN  Q|---------------------
START_WAIT
t#1s------------------------|PT   |
                            +-----+
RUNUP_MONITOR_SL_FF
        +----+
        | RS |                                +---+
S23.X---|S Q1|--+-----------------------------| & |--
RUNUP_MONITOR
S27.X---|R1  |  |   RUNUP_MONITOR_SL_TMR  +--O|   |
        +----+  |         +-----+         |   +---+
                |         | TON |         |
                +---------|IN  Q|---------+
t#1m----------------------|PT   |
                          +-----+
                                             +---+
S24.X------------+---------------------------| & |---
START_MONITOR
                 | START_MONITOR_L_TMR  +---O|   |
                 |        +-----+       |    +---+
                 |        | TON |       |
                 +--------|IN  Q|-------+
t#30s---------------------|PT   |
                          +-----+
```

Figure 16b - Action control example - functional equivalent

2.6.5 Rules of evolution

The *initial situation* of a SFC network is characterized by the *initial step* which is in the active state upon initialization of the program or function block containing the network.

Evolutions of the active states of steps shall take place along the *directed links* when caused by the *clearing* of one or more *transitions*.

A transition is *enabled* when all the preceding steps, connected to the corresponding transition symbol by directed links, are active. The clearing of a transition occurs when the transition is enabled and when the associated transition condition is true.

The clearing of a transition causes the *deactivation* (or "resetting") of all the immediately preceding steps connected to the corresponding transition symbol by directed links, followed by the *activation* of all the immediately following steps.

The alternation Step/Transition and Transition/Step shall always be maintained in SFC element connections, that is:

- Two steps shall never be directly linked; they shall always be separated by a transition.
- Two transitions shall never be directly linked; they shall always be separated by a step.

When the clearing of a transition leads to the activation of several steps at the same time, the sequences to which these steps belong are called *simultaneous sequences*. After their simultaneous activation, the evolution of each of these sequences becomes independent. In order to emphasize the special nature of such constructs, the divergence and convergence of simultaneous sequences shall be indicated by a double horizontal line.

Table 46 defines the syntax and semantics of the allowed combinations of steps and transitions.

The clearing time of a transition may theoretically be considered as short as one may wish, but it can never be zero. In practice, the clearing time will be imposed by the programmable controller implementation. For the same reason, the duration of a step activity can never be considered to be zero.

Several transitions which can be cleared simultaneously shall be cleared simultaneously, within the timing constraints of the particular programmable controller implementation and the priority constraints defined in table 46.

Testing of the successor transition condition(s) of an active step shall not be performed until the effects of the step activation have propagated throughout the program organization unit in which the step is declared.

Figure 17 illustrates the application of these rules. In this figure, the active state of a step is indicated by the presence of an asterisk (*) in the corresponding block. This notation is used for illustration only, and is not a required language feature.
The application of the rules given in this subclause cannot prevent the formulation of "unsafe" SFCs, such as the one shown in figure 18a, which may exhibit uncontrolled proliferation of tokens. Likewise, the application of these rules cannot prevent the formulation of "unreachable" SFCs, such as the one shown in figure 18b, which may exhibit "locked up" behavior. The programmable controller system shall treat the existence of such conditions as *errors* as defined in 1.5.1.

Table 46 - Sequence evolution

No.	Example	Rule

1	`   |`
	`+-----+`
	`| S3  |`
	`+-----+`
	`   |`
	`   +  c`
	`   |`
	`+-----+`
	`| S4  |`
	`+-----+`
	`   |`
		Single sequence:
		The alternation step-transition is repeated in series.
		Example:
		An evolution from step S3 to step S4 shall take place if and only if step S3 is in the active state and the transition condition c is true.

2a	`         |`
	`      +-----+`
	`      | S5  |`
	`      +-----+`
	`         |`
	`   +-----*----+--...`
	`   |          |`
	`   +  e       +  f`
	`   |          |`
	`+-----+    +-----+`
	`| S6  |    | S8  |`
	`+-----+    +-----+`
	`   |          |`
		Divergence of sequence selection:
		A selection between several sequences is represented by as many transition symbols, *under* the horizontal line, as there are different possible evolutions. The asterisk denotes left-to-right priority of transition evaluations.
		Example:
		An evolution shall take place from S5 to S6 only if S5 is active and the transition condition "e" is true, or from S5 to S8 only if S5 is active and "f" is true and "e" is false.

2b	`         |`
	`      +-----+`
	`      | S5  |`
	`      +-----+`
	`         |`
	`   +-----*-----+--...`
	`   |2          |1`
	`   +  e        +  f`
	`   |           |`
	`+-----+     +-----+`
	`| S6  |     | S8  |`
	`+-----+     +-----+`
	`   |           |`
		Divergence of sequence selection:
		The asterisk, followed by numbered branches, indicates a user-defined priority of transition evaluation, with the lowest-numbered branch having the highest priority.
		Example:
		An evolution shall take place from S5 to S8 only if S5 is active and the transition condition "f" is true, or from S5 to S6 only if S5 is active, and "e" is true, and "f" is false.

2c	`            |`
	`         +----+`
	`         | S5 |`
	`         +----+`
	`            |`
	`  +------+----+--...`
	`  |           |`
	`  +e          +NOT e`
	`& f`
	`  |           |`
	`+-----+     +-----+`
	`| S6 |     | S8 |`
	`+-----+     +-----+`
	`   |           |`
		Divergence of sequence selection: The connection of the branch indicates that the user must assure that transitionconditions are mutually exclusive, asspecified by IEC 848.
		Example:
		S6 only if S5 is active and the transition condition "e" is true, or from S5 to S8 only if S5 is active and "e" is false and "f" is true.

Table 46 - Sequence evolution *(continued)*

3

```
   |           |
+-----+     +-----+
| S7  |     | S9  |
+-----+     +-----+
   |           |
   + h         + j
   |           |
   +-----+-----+--...
         |
      +-----+
      |S10  |
      +-----+
         |
```

Convergence of sequence selection:
The end of a sequence selection is represented by as many transition symbols, *above* the horizontal line, as there are selection paths to be ended.

Example:
An evolution shall take place from S7 to S10 only if S7 is active and the transition condition "h" is true, or from S9 to S10 only if S9 is active and "j" is true.

4

```
         |
      +-----+
      |S11  |
      +-----+
         |
         + b
         |
 ==+=====+=====+==...
   |           |
+-----+     +-----+
| S12 |     | S14 |
+-----+     +-----+
   |           |
```

Simultaneous sequences - divergence:
Only one common transition symbol shall be possible, immediately *above* the double horizontal line of synchronization.

Example:
An evolution shall take place from S11 to S12, S14,... only if S11 is active and the transition condition "b" associated to the common transition is true. After the simultaneous activation of S12, S14, etc., the evolution of each sequence proceeds independently.

```
   |           |
+-----+     +-----+
| S13 |     | S15 |
+-----+     +-----+
   |           |
 ==+=====+=====+==...
         |
         + d
         |
      +-----+
      |S16  |
      +-----+
         |
```

Simultaneous sequences - convergence:
Only one common transition symbol shall be possible, immediately *under* the double horizontal line of synchronization.

Example:
An evolution shall take place from S13, S15,... to S16 only if all steps above and connected to the double horizontal line are active and the transition condition "d" associated to the common transition is true.

Table 46 - Sequence evolution *(continued)*

No.	Example	Rule

5a
5b
5c
	`       |`
	`    +-----+`
	`    | S30 |`
	`    +-----+`
	`       |`
	`   +---*---+`
	`   |       |`
	`   + a     +d`
	`   |       |`
	`+-----+    |`
	`| S31 |    |`
	`+-----+    |`
	`   |       |`
	`   + b     |`
	`   |       |`
	`+-----+    |`
	`| S32 |    |`
	`+-----+    |`
	`   |       |`
	`   + c     |`
	`   |       |`
	`   +---+---+`
	`       |`
	`    +-----+`
	`    | S33 |`
	`    +-----+`
	`       |`
		Sequence skip:
		A "sequence skip" is a special case of sequence selection (Feature 2) in which one or more of the branches contain no steps. Features 5a, 5b, and 5c correspond to the representation options given in features 2a, 2b, and 2c, respectively.
		Example:
		(Feature 5a shown)
		An evolution shall take place from S30 to S33 if "a" is false and "d" is true, that is, the sequence (S31, S32) will be skipped.

6a
6b
6c
	`    |`
	`+-----+`
	`| S30 |`
	`+-----+`
	`    |`
	`    + a`
	`    |`
	`    +---------+`
	`    |         |`
	`+-----+       |`
	`| S31 |       |`
	`+-----+       |`
	`    |         |`
	`    + b       |`
	`    |         |`
	`+-----+       |`
	`| S32 |       |`
	`+-----+       |`
	`    |         |`
	`    *-----+   |`
	`    |     |   |`
	`    + c   + d |`
	`    |     |   |`
	`+-----+  +---+`
	`| S33 |`
	`+-----+`
	`    |`
		Sequence loop:
		A "sequence loop" is a special case of sequence selection (Feature 2) in which one or more of the branches returns to a preceding step. Features 6a, 6b, and 6c correspond to the representation options given in features 2a, 2b, and 2c, respectively.
		Example:
		(Feature 6a shown)
		An evolution shall take place from S32 to S31 if "c" is false and "d" is true, that is, the sequence (S31, S32) will be repeated.

Table 46 - Sequence evolution *(concluded)*

No.	Example	Rule

7

```
   |
+-----+
| S30 |
+-----+
   |
   + a
   |
   +----<----+
   |         |
+-----+      |
| S31 |      |
+-----+      |
   |         |
   + b       |
   |         |
+-----+      |
| S32 |      |
+-----+      |
   |         |
   *-----+   |
   |     |   |
   + c   + d |
   |     |   |
+-----+  +->-+
| S33 |
+-----+
   |
```

Directional arrows:
When necessary for clarity, the "less than" (<) character of the ISO 646 character set can be used to indicate right-to-left control flow, and the "greater than" (>) character to represent left-to-right control flow. When this feature is used, the corresponding character shall be located between two "-" characters, that is, in the character sequence "-<-" or "->-" as shown in the accompanying example.

a) Transition not enabled (X = Don’t care)

```
    |                  |        |         |
+-------+           +-----+  +-------+  +---
---+
|STEP10|            |STEP9|  |STEP13|
|STEP22|
|      |            |     |  |  *   |  |  *
|
+-------+           +-----+  +-------+  +---
---+
    |                  |        |         |
    + X
====+========+=========+====
    |                           |
+-------+                       + X
|STEP11|                        |
|      |               ====+====+===+====
+-------+                  |        |
    |                  +-------+ +-------+
                       |STEP15|  |STEP16|
                       |      |  |      |
                       +-------+ +-------+
                           |        |
```

b) Transition enabled but not cleared (X = 0)

```
    |                 |        |         |
+-------+          +-----+  +-------+  +----
--+
|STEP10|           |STEP9|  |STEP13|
|STEP22|
|  *   |           |  *  |  |  *   |  |  *
|
+-------+          +-----+  +-------+  +----
--+
    |                 |        |         |
    + X
===+========+=========+====
    |                          |
+-------+                      + X
|STEP11|                       |
|      |              ====+====+====+====
+-------+                 |         |
    |                  +-------+ +-------+
                       |STEP15|  |STEP16|
                       |      |  |      |
                       +-------+ +-------+
                          |         |
```

c) Transition cleared (X = 1)

```
    |                  |        |         |
+------+            +-----+  +------+  +---
---+
|STEP10|            |STEP9|  |STEP13|
|STEP22|
|      |            |     |  |      |  |
|
+------+            +-----+  +------+  +---
---+
    |                  |        |         |
    + X
====+========+========+====
    |                           |
+------+                        + X
|STEP11|                        |
|   *  |               ====+====+===+====
+------+                   |        |
    |                  +------+  +------+
                       |STEP15|  |STEP16|
                       |   *  |  |  *   |
                       +------+  +------+
                           |        |
```

NOTE - In this figure, the active state of a step is indicated by the presence of an asterisk (*) in the corresponding block. This notation is used for illustration only, and is not a required language feature.

Figure 17 - SFC evolution rules

```
+----------------------+
|                      |
|                   +=====+
|                   || A ||
|                   +=====+
|                      |
|                      + t1
|                      |
|     ======+==========+============+=======
|           |                       |
|        +------+                +------+
|        |  B   |                |  C   |
|        +------+                +------+
|           |                       |
|           |                       *--------
+
|           |                       |
|
|           |                       + t2
+ t3
|           |                       |
|
|           |                     +---+     +-
--+
|           |                     | D |     |
E |
|           |                     +---+     +-
--+
|           |                       |
|
|        ===+==========+============+===
|
|                      |
|
|                      + t4
+ t5
|                      |
|
|                    +---+                +-
--+
|                    | F |                |
G |
|                    +---+                +-
--+
|                      |
|
|                      + t6
+ t7
|                      |
|
+----------------------+--------------------
+
```

Figure 18a - SFC errors: an "unsafe" SFC (see 2.6.5)

```
+----------------------+
|                      |
|                   +=====+
|                   || A ||
|                   +=====+
|                      |
|                      + t1
|                      |
|     ======+==========+============+=======
|           |                       |
|        +------+                +------+
|        |  B   |                |  C   |
|        +------+                +------+
|           |                       |
|           |                       *--------
+
|           |                       |
|
|           |                       + t2
+ t3
|           |                       |
|
|           |                     +---+     +-
--+
|           |                     | D |     |
E |
|           |                     +---+     +-
--+
|           |                       |
|
|        ===+==========+============+===
|
|                      |
|
|                      + t4
+ t5
|                      |
|
|                    +---+                  +-
--+
|                    | F |                  |
G |
|                    +---+                  +-
--+
|                      |
|
|
====+==========+==========+===
|                                 |
|                                 + t6
|                                 |
+---------------------------------+
```

Figure 18b - SFC errors: an "unreachable" SFC (see 2.6.5)

2.6.6 Compatibility of SFC elements

SFCs can be represented graphically or textually, utilizing the elements defined above. Table 47 summarizes for convenience those elements which are mutually compatible for graphical and textual representation, respectively.

Table 47 - Compatible SFC features

Table	Graphical representation	Textual representation
40	1, 3a, 3b, 4	2, 3a, 4
41	1,2,3,4,4a,4b,7,7a,7b	5, 6, 7c, 7d
42	1, 2l, 2s, 2f	3s,3i
43	1, 2, 4	3
44	1 to 9	--
45	1 to 10	1 to 10 (textual equivalent)
46	1 to 7	1 to 6
57	All	--

2.6.7 Compliance requirements

In order to claim compliance with the requirements of 2.6, the elements shown in table 48 shall be supported and the compatibility requirements defined in 2.6.6 shall be observed.

Table 48 - SFC minimal compliance requirements

Table	Graphical representation	Textual representation
40	1	2
41	1 or 2 or 3 or (4 and (4a or 4b)) or (7 and (7a or 7b or 7c or 7d))	5 or 6
42	1 or 2l or 2f	3s or 3i
43	1 or 2 or 4	3
45	1 or 2	1 or 2
46	1 and (2a or 2b or 2c) and 3 and 4	Same (textual equivalent)
57	(1 or 2) and (3 or 4) and (5 or 6) and (7 or 8) and (9 or 10) and (11 or 12)	Not required