

1) IEC 61131-3 Introduction

La norme* IEC 61131-3 standardise la programmation des automates

*Nous allons aborder la description des fonctions logiques
en langage LD, FBD, ST et IL*

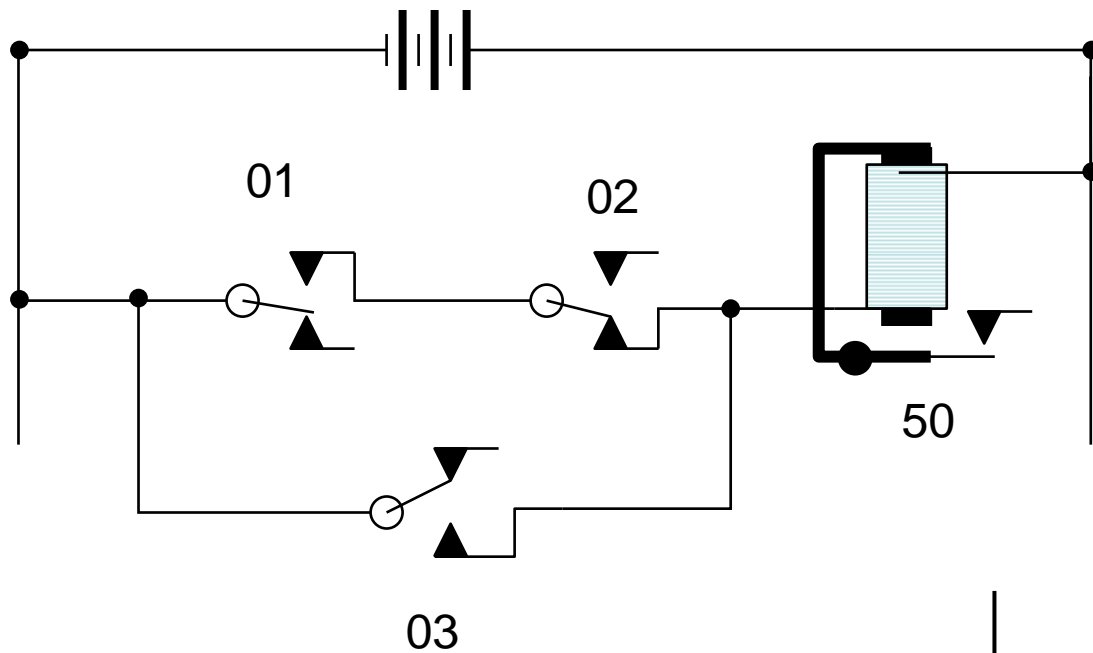
* Document établi par consensus et approuvé par un organisme de normalisation reconnu (ISO, CEI, NF ...).

Notes de cours

Philippe RAYMOND

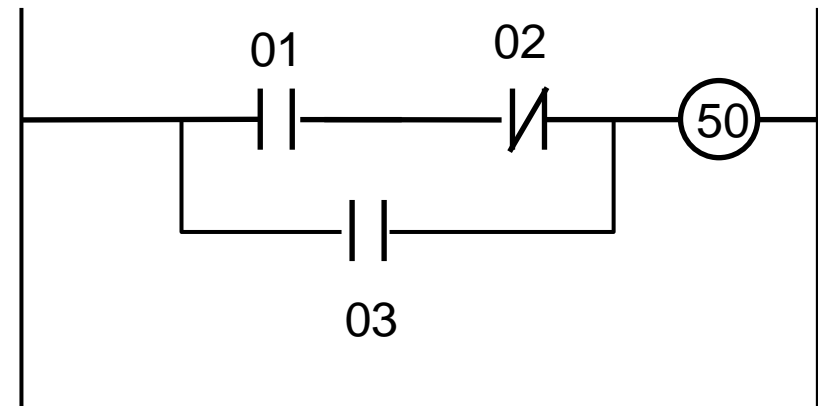
septembre 2005

Le Langage LD Ladder Diagram

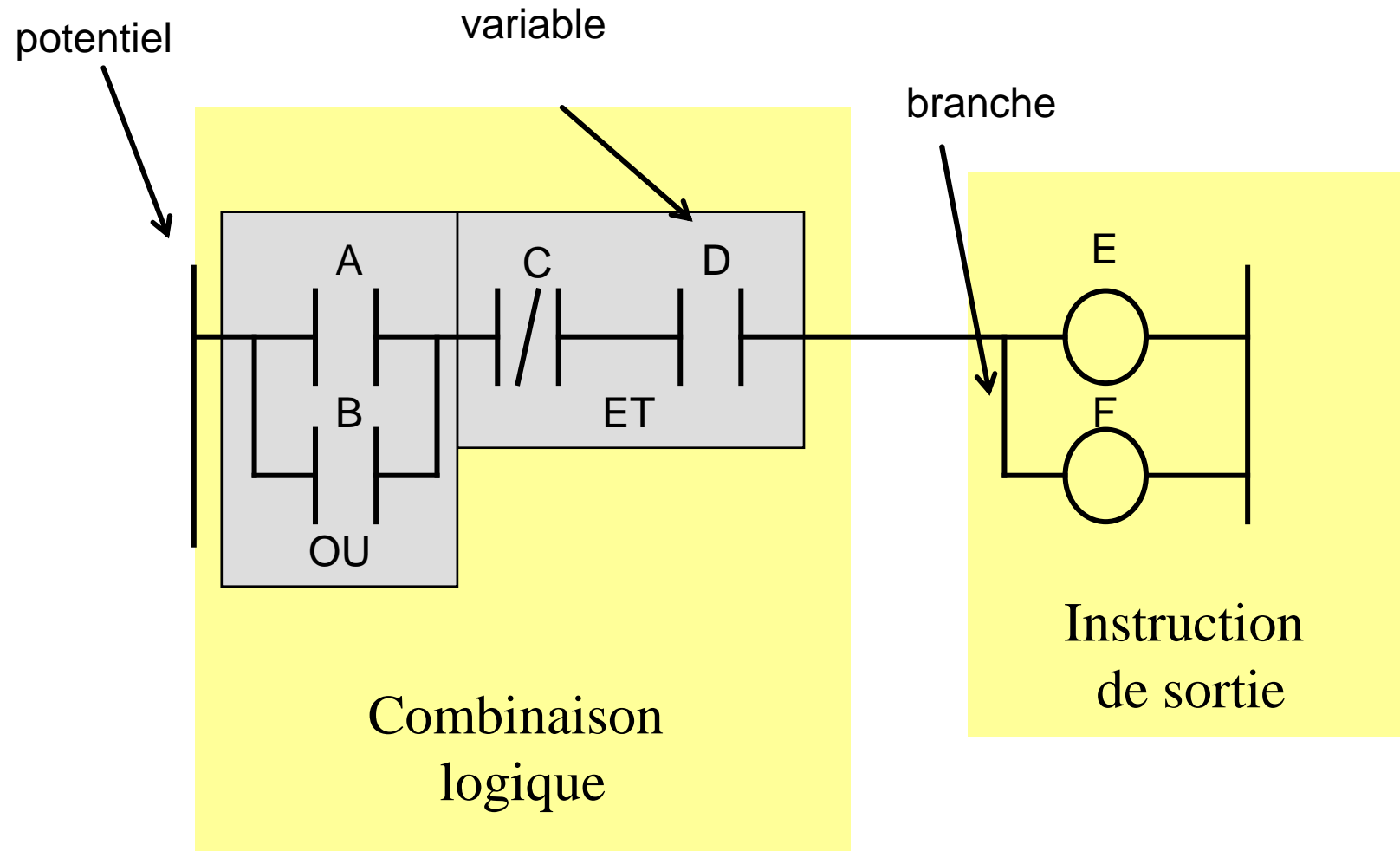


L'un des premiers langages pour les automates, reconnu depuis longtemps aux USA, en Amérique, et en Asie

...transition avec les diagrammes électriques...



Structure d'un RUNG

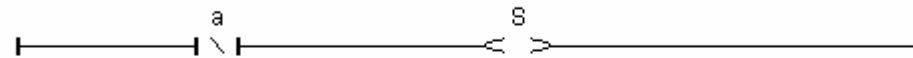


Eléments

(* OUI *)

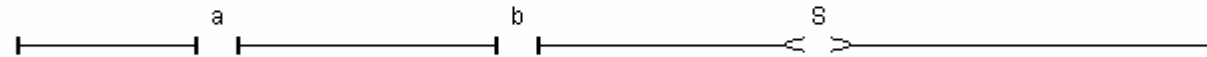


(* NON *)

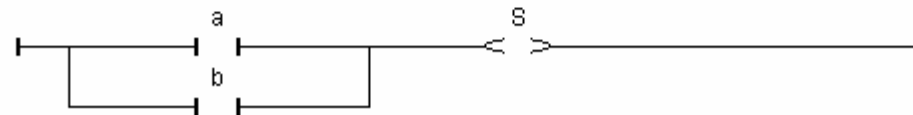


Symbole \ pour traduire le NON

(* ET *)



(* OU *)



contact NO -| |-

contact NF -|/|-

contact Front Montant -|P|-

contact Front Descendant -|N|-

bobine de sortie -()-

bobine Negative -(/)-

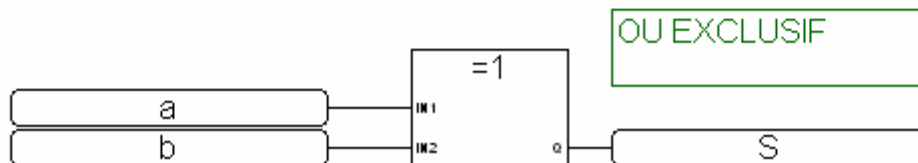
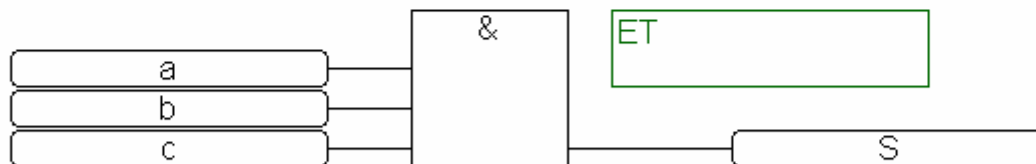
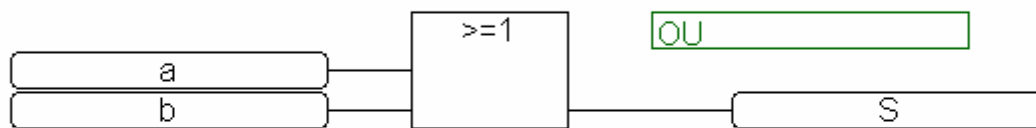
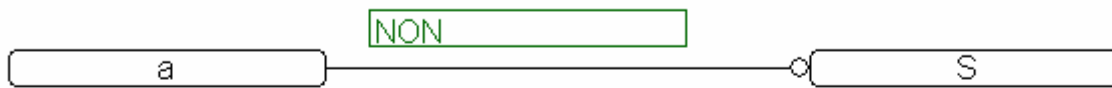
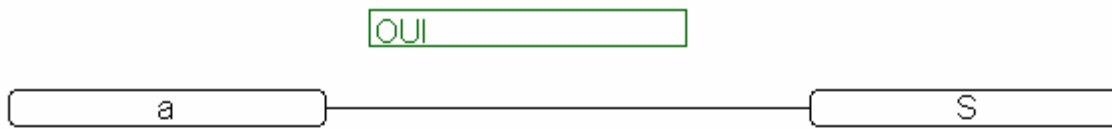
bobine SET -(S)-

bobine RESET -(R)-

Le Langage FBD

Function Block Diagram

Langage graphique, où les éléments du programme sont représentés par des blocs interconnectés.



liaison --o
Pour traduire le NON

Le Langage ST Structured Text

Langage haute niveau, structuré présentant une syntaxe qui ressemble au langage **PASCAL**

```
(*OUI*)  
S:=a;  
(*NON*)  
S:=NOT (a);  
(*OU*)  
S:=a OR b;  
(*AND*)  
S:= a AND b;  
S:= a XOR b;
```

**Le := correspond à
l'assignation**

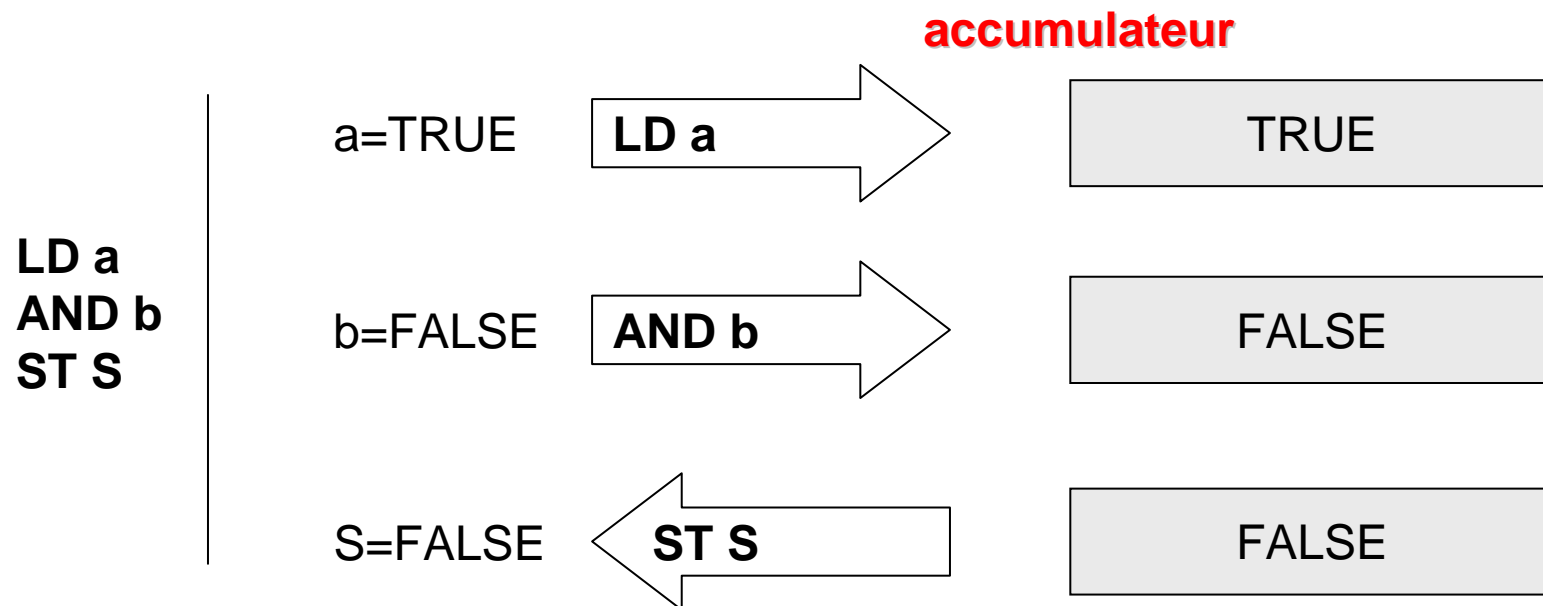
**L'instruction NOT au
complément logique**

LANGAGE IL

Instruction List

langage textuel de bas niveau, type langage assembleur basé sur le concept d'un **accumulateur**

Dans une ligne du programme, seulement **une opération** d'accès au registre d'accumulateur est autorisée



LANGAGE IL

(* OUI*)

LD a

ST S

(*NON*)

LDN a

ST S

(*NON*)

LD a

STN S

(*OU*)

LD a

OR b

ST S

(*ET*)

LD a

AND b

ST S

(*OU EXCLUSIF*)

LD a

XOR b

ST S

LD charge un opérande (variable ou constante).

ST stocke le résultat courant dans la variable.

N traduit le NON.

Re=ab+cd

(* Re = ((a.b)+c).d *)

ld a
and b
or c
and d
st Re

(* Re = a.b+c.d *)

ld a
and b
st Re
ld c
and d
or Re
st Re

(* Re = a.b+c.d *)

ld a
and b
st VAR_1

ld c
and d
st VAR_2

ld VAR_1
or VAR_2
st Re

INTERNATIONAL STANDARD

IEC 61131-3

Second edition
2003-01



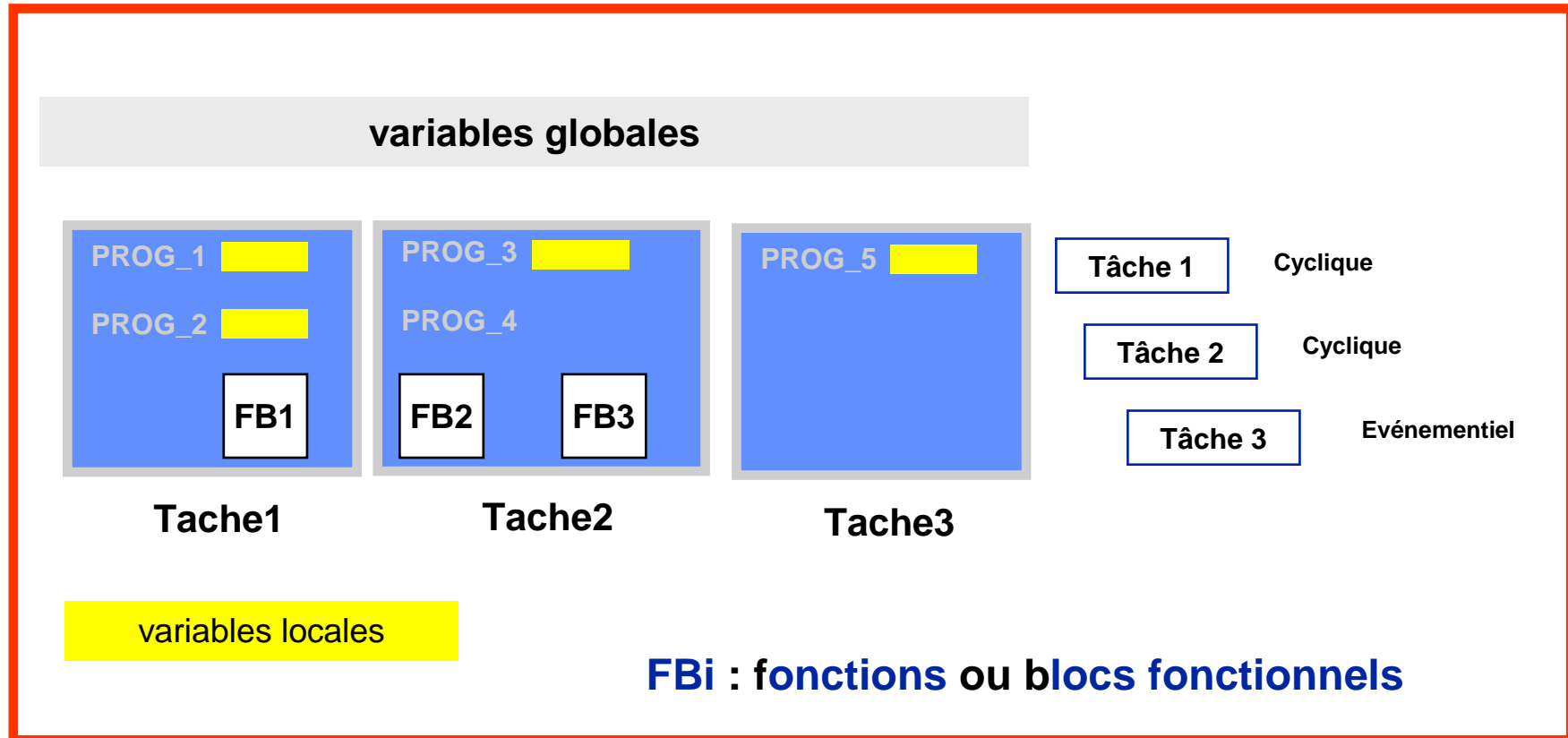
Notes de cours

Philippe RAYMOND

octobre 2005

Organisation Logicielle

La norme suppose l'existence d'un système (ressource) *multitâche et temps réel*.



Notion de POU*

Program Organization Unit

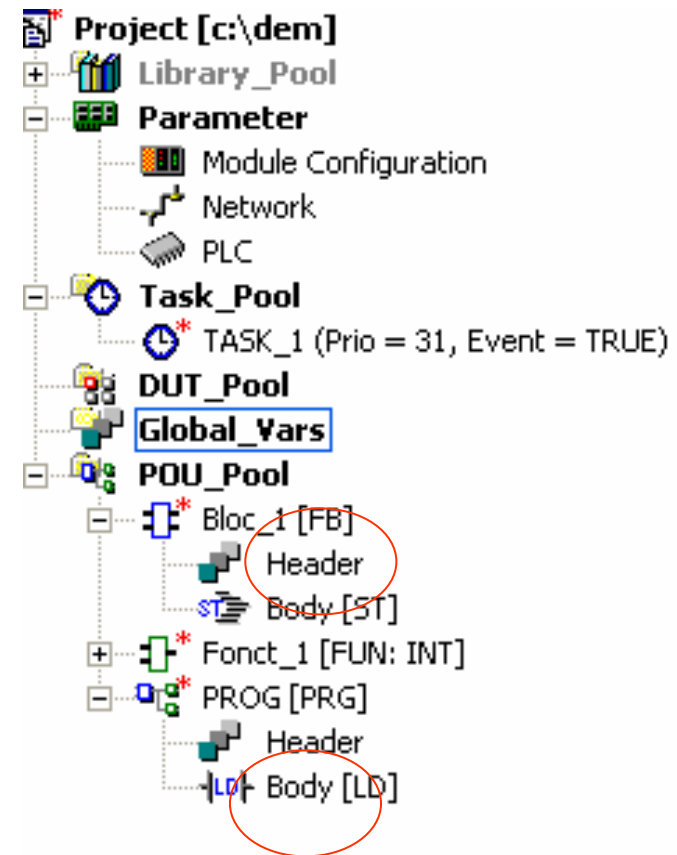
les Programmes, les Fonctions, les Blocs Fonctionnels forment les POU.

Le nom d'un POU doit être unique au sein d'un projet

Chaque POU se compose en deux parties différentes :

la partie « *Déclaration de Variables* », ou toutes les variables nécessaires sont déclarées (Header)

la partie « *Code du Programme* » qui contient les instructions dans le langage de programmation désiré (Body)



* UOP = Unité d'Organisation de Programmes

Exemple organisation sur Mitsubishi

The screenshot displays the Mitsubishi GX Developer software interface. On the left, a project tree for 'Project [c:\test]' is shown. The tree includes folders for 'Library_Pool', 'Parameter', 'Task_Pool', 'DUT_Pool', 'Global_Vars', and 'POU_Pool'. The 'Task_Pool' folder contains 'TASK_1 (Prio = 31, Event = TR)' and 'TASK_2 (Prio = 31, Event = TR)'. The 'POU_Pool' folder contains 'POU_1 [PRG]', 'POU_2 [PRG]', and 'POU_3 [PRG]'. The 'POU_1 [PRG]' folder is expanded, showing 'Header', 'Body [FBD]', and 'Body [SFC]'. The 'POU_2 [PRG]' folder is expanded, showing 'Header', 'Body [SFC]', and 'Action_Pool'. The 'POU_3 [PRG]' folder is expanded, showing 'Header' and 'Body [LD]'. On the right, two task configuration windows are shown. The top window is titled 'TASK_1 (Prio = 31, Event = TRUE)' and contains a table with the following data:

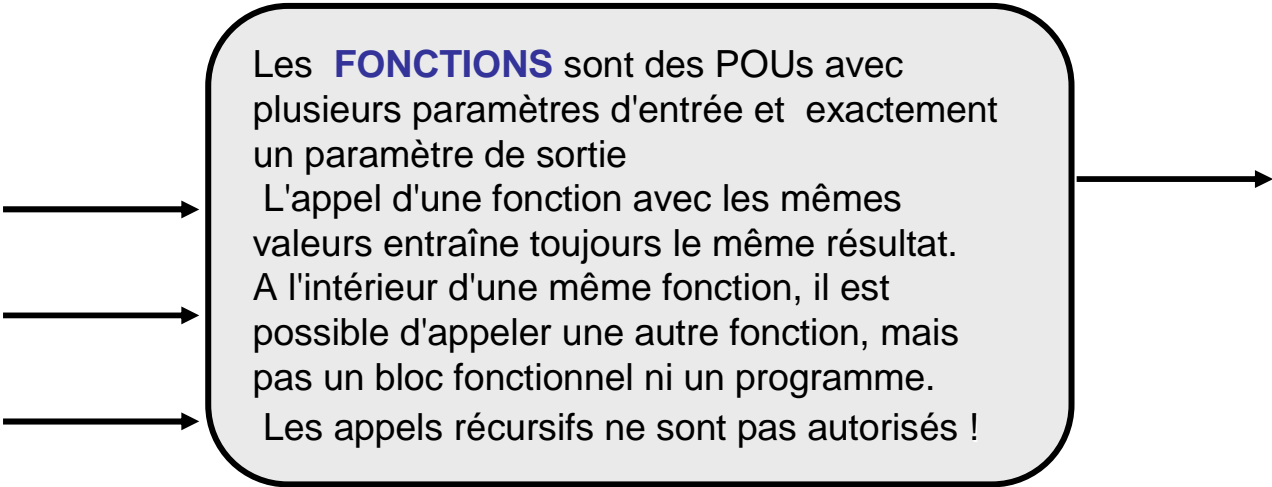
	POU name	Comment
0	POU_1	...

The bottom window is titled 'TASK_2 (Prio = 31, Event = TRUE)' and contains a table with the following data:

	POU name	Comment
0	POU_2	...
1	POU_3	...

Red circles highlight the task names and the POU names in the tables.

Notion de Fonction



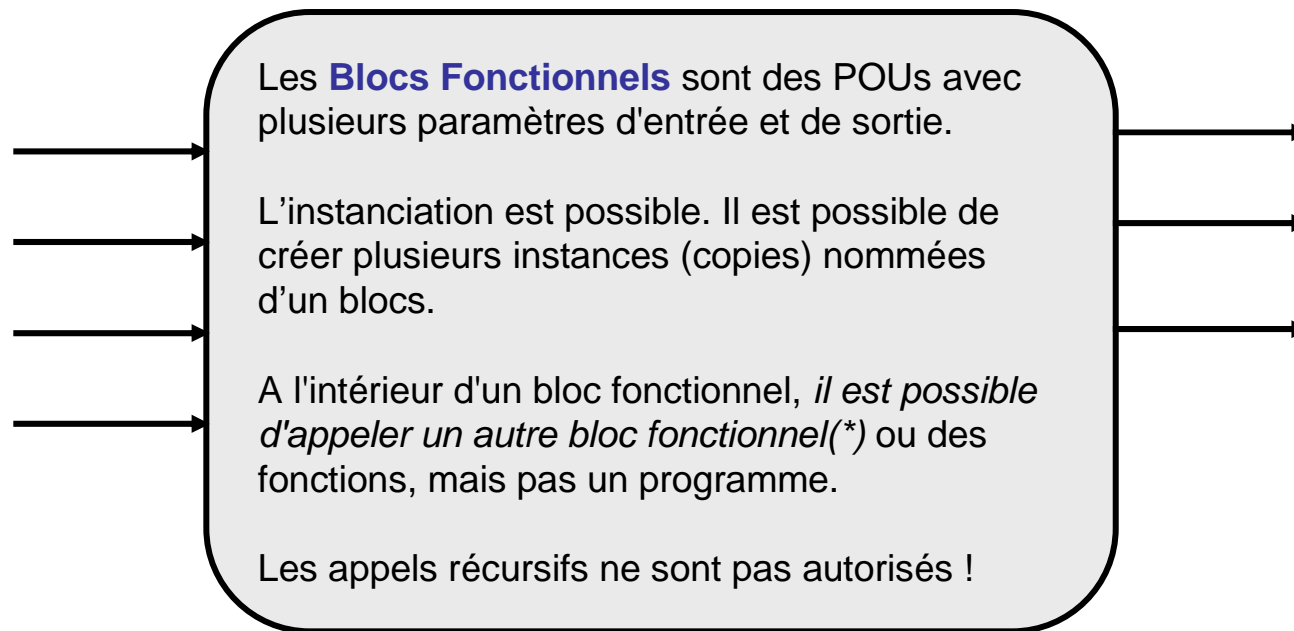
Les **FONCTIONS** sont des POUs avec plusieurs paramètres d'entrée et exactement un paramètre de sortie

L'appel d'une fonction avec les mêmes valeurs entraîne toujours le même résultat.

A l'intérieur d'une même fonction, il est possible d'appeler une autre fonction, mais pas un bloc fonctionnel ni un programme.

Les appels récursifs ne sont pas autorisés !

Notion de bloc fonctionnel



(*) pas toujours possible...

Variable Declarations

program

VAR_GLOBAL
VAR_GLOBAL_CONSTANT

HEADER

Interface variables

VAR_EXTERNAL
VAR_EXTERNAL_CONSTANT

Local variables

VAR
VAR_CONSTANT

BODY

instructions

Exemple sur Mitsubishi

The screenshot displays the Mitsubishi GX Developer interface. On the left is the Project Explorer showing a tree structure for 'Project [c:\dem]'. The main workspace is divided into three overlapping windows:

- Global Variable List:** A table listing global variables.
- PROG [PRG] Header:** A table listing variables declared in the program header.
- PROG [PRG] Body [LD]:** A ladder logic diagram for program 'PRG'.

Global Variable List Table:

	Class	Aut	Identifier	MIT-Addr.	IEC-Addr.
0	VAR_GLOBAL_CONSTANT		VolumeMax		
1	VAR_GLOBAL		VolumeCuve		

PROG [PRG] Header Table:

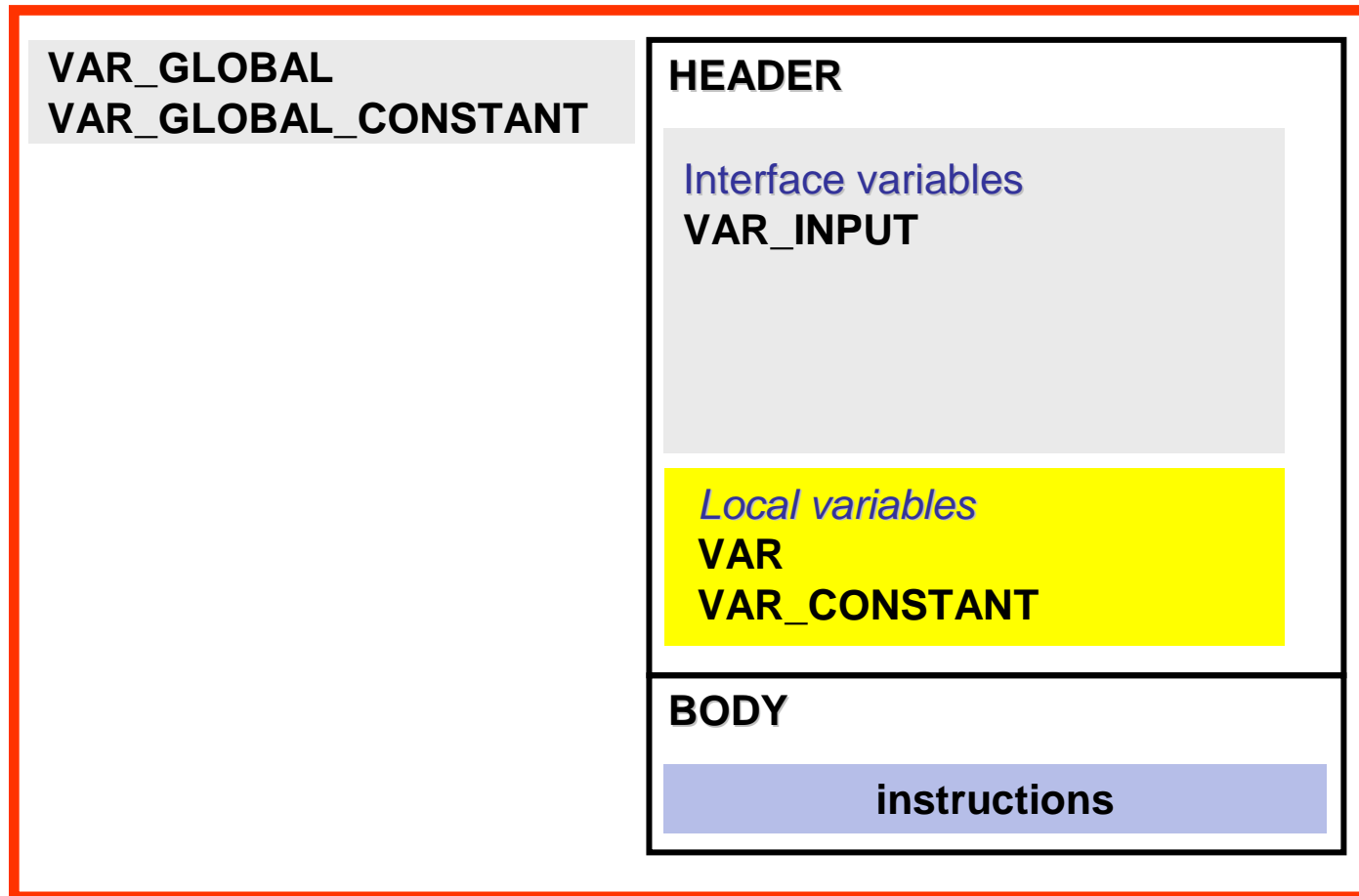
	Class	Identifier	Type	
0	VAR_EXTERNAL	VolumeCuve	INT	0
1	VAR_EXTERNAL_CONSTANT	VolumeMax	INT	10
2	VAR	POMPE	BOOL	F.

PROG [PRG] Body [LD] Diagram:

The diagram shows a single step (Step 1) in a normally closed contact configuration. The contacts are labeled 'VolumeCuve' and 'VolumeMax'. The coil is labeled 'EQ'. The output is labeled 'POMPE'.

Variable Declarations

function



Exemple sur Mitsubishi

The screenshot displays the Mitsubishi GX Developer software interface. On the left, a project tree for 'Project [c:\dem]' is visible, showing a hierarchy of components including Library_Pool, Parameter, Task_Pool (TASK_1), DUT_Pool, Global_Vars, and POU_Pool. Under POU_Pool, a function block 'circonference [FUN: INT]' is expanded, showing its Header and Body sections.

The main window shows the 'circonference [FUN: INT] Body [ST]' section with the following code:

```
circonference := PI*diametre;
```

Below the code, the 'circonference [FUN: INT] Header' section is open, displaying a table of variables:

	Class	Identifier	Type
0	VAR_CONSTANT	PI	REAL
1	VAR_INPUT	diametre	REAL

Variable Declarations block

VAR_GLOBAL
VAR_GLOBAL_CONSTANT

HEADER

Interface variables

VAR_EXTERNAL

VAR_EXTERNAL_CONSTANT

VAR_INPUT

VAR_OUTPUT

VAR_IN_OUT

Local variables

VAR

VAR_CONSTANT

BODY

instructions

Exemple Mitsubishi

The screenshot displays the Mitsubishi GX Developer interface with several windows open:

- Project [c:\dem]:** Shows a tree view with folders for Library_Pool, Parameter, Task_Pool, DUT_Pool, Global_Vars, and POU_Pool. Under POU_Pool, there is a folder for Bloc_1 [FB] containing a Header, Body [ST], and a sub-programme PROG [PRG] with its own Header and Body [FBD].
- Global Variable List:** A table listing global variables.

	Class	Aut	Identifier	MIT-Addr.	IEC-A	Type	Initial
0	VAR_GLOBAL		DEBIT_STATION_A			REAL	0.0
1	VAR_GLOBAL		VOLUME_STATION_A			REAL	0.0
2	VAR_GLOBAL_CONSTANT		Periode_Echantillon			REAL	0.5
3	VAR_GLOBAL_CONSTANT		VolumeMax			REAL	100.0
- Bloc_1 [FB] Header:** A table listing local variables for the function block.

	Class	Identifier	Type	Initial
0	VAR_INPUT	debit	REAL	0.0
1	VAR_OUTPUT	volume	REAL	0.0
2	VAR_EXTERNAL_CONSTANT	Periode_Echantillon	REAL	0.5
3	VAR_EXTERNAL_CONSTANT	VolumeMax	REAL	100.0
- PROG [PRG] Header:** A table listing variables for the program.

	Class	Identifier	Type	Initial
0	VAR_EXTERNAL	VOLUME_STATION_A	REAL	0.0
1	VAR_EXTERNAL	DEBIT_STATION_A	REAL	0.0
2	VAR	Station_A	Bloc_1	
- PROG [PRG] Body [FBD]:** A ladder logic diagram showing a network with three inputs: DEBIT_STATION_A, debit (from Bloc_1), and volume (from Bloc_1). The output is VOLUME_STATION_A.
- Bloc_1 [FB] Body [ST]:** A structured text block containing the assignment: `volume:=debit*periode_Echantillon;`

Exemple d'organisation B&R

B&R Automation Studio - [pp_24_v1] - [pp_24_v1 [Projet]]

Fichier Edition Affichage Insertion Ouvrir Projet Objet Outils Fenêtre Aide

Logiciel Permanent Série Ethernet Journal d'erreurs

Logiciel	Permanent	Série	Ethernet	Journal d'erreurs
Nom du module				
Version	Transfert vers	Taille (octets)	Description	
CPU				
Cyclique #1 - [11.2 ms]				
iqintc1	V1.58	ROM utilisateur	8620	
mc_man	V1.60.1	ROM utilisateur	578544	
confvar	V0.00	ROM utilisateur	24132	
pom	V0.00	ROM utilisateur	12512	
ma_move	V0.00	ROM utilisateur	7644	
telecmd	V0.00	ROM utilisateur	5032	
iqoutc1	V1.58	ROM utilisateur	8392	
Cyclique #2 - [20.8 ms]				
iqintc2	V1.58	ROM utilisateur	8356	
ecrans	V0.00	ROM utilisateur	4348	Gestion ecrans
gct	V0.00	ROM utilisateur	1996	Gestion des taches
tache_t1	V0.00	ROM utilisateur	3296	Amener produit
tache_t2	V0.00	ROM utilisateur	3224	Amener caisse
tache_t3	V0.00	ROM utilisateur	4896	Gestion camera
tache_t4	V0.00	ROM utilisateur	5792	Prehension produit
tache_t5	V0.00	ROM utilisateur	4048	Transfert produit
tache_t6	V0.00	ROM utilisateur	5548	Depose produit
tache_t7	V0.00	ROM utilisateur	5720	Retour bras
out_tor	V0.00	ROM utilisateur	852	
iqoutc2	V1.58	ROM utilisateur	8124	
Cyclique #3 - [51.2 ms]				
iqintc3	V1.58	ROM utilisateur	8120	
iqoutc3	V1.58	ROM utilisateur	7892	
Cyclique #4 - [100.8 ms]				
iqintc4	V1.58	ROM utilisateur	7888	
mc_slow	V1.60.1	ROM utilisateur	2544	
iqoutc4	V1.58	ROM utilisateur	7656	
Cyclique #5 - [200 ms]				

* Transfert de ihm OK

Messages Débugueur Recherche dans des fichiers Pile des appels

Ligne 12 de 75 Tcpip/DA=1 /DAIP=192.168.1.9 PP200 V2.68 RUN

Les variables

The screenshot displays the Control FPWIN Pro interface. The main window shows a ladder logic program with four rungs. The first rung contains a normally open contact labeled 'Sensor_start', followed by a normally closed contact labeled 'Limit_Switch_1', and a coil labeled 'Motor_up_down'. The second rung contains a normally open contact labeled 'Start_button' and a coil labeled 'Motor_up_down'. The third and fourth rungs are empty.

The Project Navigator on the left shows a tree view with the following items: Project [C:\Programme\NAIS Cont], Program Code, Library_Pool, PLC_Config (FP0,2.7k), Task_Pool, DUT_Pool, * Global Variables, POU_Pool, and * Program_1 [PRG]. The 'Global Variables' and '* Program_1 [PRG]' items are circled in red.

The Global Variables table is shown below the main window. It has the following columns: Class, Identifier, Matsushita, IEC_Address, Type, Initial, Au, and Com. The table contains four rows of data, with the first row highlighted. The 'Class' and 'Type' columns are circled in red.

	Class	Identifier	Matsushita	IEC_Address	Type	Initial	Au	Com
0	VAR_GLOBAL	Sensor_start	X0	%IX0.0	BOOL	FALSE		
1	VAR_GLOBAL	Limit_Switch_1	X1	%IX0.1	BOOL	FALSE		
2	VAR_GLOBAL	Start_button	X2	%IX0.2	BOOL	FALSE		
3	VAR_GLOBAL	Motor_up_down	Y0	%QX0.0	BOOL	FALSE		

Type de variables classique

Type de données	Taille	Etendue
Boolean BOOL	1	[0(false),1(true)]
Short integer SINT	8	[-128,127]
Integer INT	16	[-32768,32767]
Double integer DINT	32	[-2.147.483.648,2.147.483.647]
Unsigned short integer USINT	8	[0,255]
Unsigned integer UINT	16	[0,65535]
Unsigned double integer UDINT	32	[0,4.294.967.295]
Real numbers REAL	32	$[-2^{128}, 2^{128}]$
Bit string of length 8 BYTE	8	[16#00,16#FF]
Bit string of length 16 WORD	16	[16#0000,16#FFFF]
Bit string of length 32 DWORD	32	[16#0000,16#FFFFFFFF]
TIME(*)		T#0,00s à T#21 474 836,47s
TIME_OF_DAY, DATE_AND_TIME, DATE,		Exemple : DATE_AND_TIME#2001-04-02-10:15:29.00 DATE#2001-04-02
STRING	8 bits / ASCII	1 à 255 caractères ASCII

Adressage IEC

variables représentées directement

Les variables emplacements sont déclarées en utilisant un nom symbolique et une adresse logique

%	Attribut	type	i.j.k
	I: Entrée	X: boolean	i: <i>numéro de voie</i>
	Q : Sortie	B: 8 bits	j: <i>numéro de carte</i>
	M : mémoire	W: 16 bits	k: <i>numéro de rack</i>
	K : constante	D: 32 bits	
		F: flottant	

%IX1.5 Entrée TOR n°5 de la carte n°1 du rack par défaut n°0
%QX2.4.F Sortie TOR voie F du module 4 du rack 2
%MX128 bit interne n°128
%MW4 entier 16 bits n°4

Exemple Schneider

TABLE_1 (Animée)* 1216

Modification		Repère	Symbole / Nom	Valeur courante
F3	Modifier	%MB0		16#FE
F7	0	%MB1		16#CA
F8	1	%MB2		16#D0
		%MB3		16#D0
		%MW0		16#CAFE
		%MW1		16#D0D0
F4	Forcer 0	%MW2		16#BABA
F5	Forcer 1	%MW3		16#0000
F6	Déforger	%MD0		16#D0D0CAFE
		%MD1		16#BABAD0D0
		%MF0		-2.802371e10
		%MF1		-0.00142529

				%MD0 (16#D0D0CAFE)			
				%MW1(16#D0D0)		%MW0(16#CAFE)	
				%MB3(16#D0)	%MB2(16#D0)	%MB1(16#CA)	%MB0(16#FE)
				%MD1 (16#BABAD0D0)			
				%MW2(16#BABA)		%MW1(16#D0D0)	
%MB5(16#BA)	%MB4(16#BA)	%MB3(16#D0)	%MB2(16#D0)				

Exemple B&R

B&R Automation Studio - [pp_24_v1] - [ecrans.src [Declaration]]

Fichier Edition Insertion Affichage Ouvrir Projet Objet Outils Fenêtre Aide

Nom	Type	Domaine	Attribut	Valeur	Propriétaire	Description
disable_limit_master	BOOL	global	QE3.1.1.1.03.07			0,5 A, 24 VDC
fd1	BOOL	local	inteme			
fd_pilz	BOOL	local	inteme			
fm0	BOOL	local	inteme			
fm1	BOOL	local	inteme			
fm20	BOOL	local	inteme			
fm5	BOOL	local	inteme			
fm6	BOOL	local	inteme			
fm_button_GO	BOOL	local	inteme			
fm_button_home	BOOL	local	inteme			
fm_button_infos	BOOL	local	inteme			
fm_button_manuAxe	BOOL	local	inteme			
fm_button_manuMove	BOOL	local	inteme			
initGCT	BOOL	global	inteme			
msg_ecran0	STRING(50)	local	inteme			msg sur écran 0
msg_ecran3	STRING(50)	local	inteme			msg sur écran 3
num_ecranMoveAxes	UINT	global	constante	3		
num_ecranMoveXYZ	UINT	global	constante	5		
num_ecran_actif	UINT	global	inteme			retourne l'écran actif
num_ecran_infos	UINT	local	constante	10		
num_ecran_memorise	UINT	local	inteme			mémorise l'écran du niveau 0
password	USINT	local	inteme			mot de passe pour modes manu
pilz	BOOL	global	IE3.1.1.1.02.02	-----		24 VDC, 1 ms
pomRD47ok	BOOL	global	inteme			
ptG	ptXYZ	global	inteme			G et le pt produit dans Rep. Mac
pt_att	ptXYZ	global	inteme			
pt_encaissage	ptXYZ	global	inteme			
pt_origine	ptXYZ	global	inteme			
pt_pstactuMAC	ptXYZ	global	inteme			
pt_pstactuROB	ptROB	global	inteme			
pt_souhaiteeMAC	ptXYZ	global	inteme			
ream_pilz	BOOL	global	QE3.1.1.1.02.01			0,5 A, 24 VDC
reboot_cam	BOOL	global	QE3.1.1.1.02.02			0,5 A, 24 VDC
stopcame1	BOOL	global	inteme	1		
stopcame2	BOOL	global	inteme	1		

* Transfert de ihm OK

Messages Débugueur Recherche dans des fichiers Pile des appels

Notions de Enum, Struct et Array

(*Type de données ENUM *)

```
TYPE  
feux : (rouge,jaune,vert);  
END_TYPE
```

```
VAR  
signal : feux;  
END_VAR
```

```
...  
IF signal = rouge THEN...
```

(*Type de données ARRAY*) (* maximum 3 dimensions *)

```
TYPE  
Graphe : ARRAY [1..23] OF INT;  
Mesure : ARRAY[0..10,0..10,0..10] of REAL;  
END_TYPE
```

(*Type de données STRUCT *)

```
TYPE
```

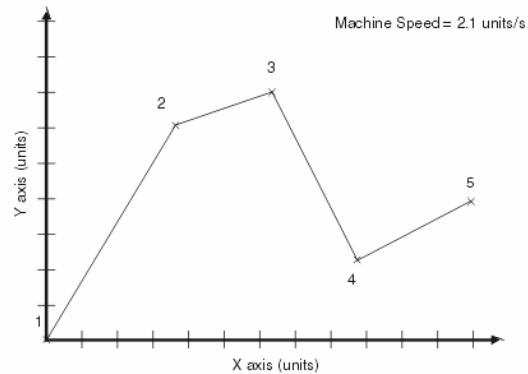
```
alesage : STRUCT  
pos_x : INT;  
pos_y: INT;  
profondeur:INT;  
END_STRUCT;
```

```
END_TYPE
```

```
VAR  
alesage_1 : alesage;  
END_VAR  
...  
alesage_1 .pos_x := 100;  
alesage_1 .pos_y := 200;  
alesage_1 .profondeur := 50;  
...
```

Exemple : calcul de distance

positioning chart



The distance between two points is calculated with the following formula:

$$\text{dist} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

```
TYPE
COORD :STRUCT
x : REAL;
y : REAL;
END_STRUCT;
END_TYPE
```

```
VAR
point : ARRAY [0..4] OF COORD;
dist, diff , x_diff , y_diff : REAL;
index : USINT
END_VAR
```

```
dist = 0.0;
FOR index := 0 TO 4 DO
  x_diff := point[index+1].x - point[index].x;
  y_diff := point[index+1].y - point[index].y;
  diff := SQRT(pow(x_diff,2) + pow(y_diff,2));
  dist := dist + diff;
END_FOR;
```

Exemple sur atelier B&R

B&R Automation Studio - [pp_24_v1] - [ConvCamMac.SRC [Texte structuré]]

Fichier Edition Affichage Insertion Ouvrir Projet Objet Outils Fenêtre Aide

```

0001 (* Implémentation de ConvCamMac *)
0002
0003 (* cette fonction réalise le passage du repère caméra *)
0004 (* vers le repère machine *)
0005
0006 n.x:=m.x*EX + J1X;
0007 n.y:=m.y*EY+J1Y;
0008 n.z:=EP+J1Z;
0009 n.o:=m.o;
0010
  
```

B&R Automation Studio - [pp_24_v1] - [pp_24_v1 [Structure de données]]

Fichier Edition Affichage Insertion Ouvrir Projet Objet Outils Fenêtre Aide

Nom	Type	Propriétaire
mcSPT_LATCH_typ		mc_lib
mcSPT_LOGIC_typ		mc_lib
mcSPT_MPGEN_typ		mc_lib
mcSPT_MUX_typ		mc_lib
mcSPT_PARAM_typ		mc_lib
mcSPT_PID_typ		mc_lib
mcSPT_VAR_typ		mc_lib
mcSignal_typ		mc_lib
offsetPar_typ		mc_lib
phasePar_typ		mc_lib
positionSwitch_typ		mc_lib
printMarkData_typ		mc_lib
ptROB		PP2005
ptXYZ		PP2005
o	REAL	PP2005
z	REAL	PP2005
y	REAL	PP2005
x	REAL	PP2005

B&R Automation Studio - [pp_24_v1] - [pp_24_v1 [Gestionnaire de bibliothèques]]

Fichier Edition Affichage Insertion Ouvrir Projet Bibliothèque Objet Outils Fenêtre Aide

Nom	Type	Version
Bibliothèques		
acp10man	Binaire	V1.14
acp10par	Binaire	V1.14
AsString	Binaire	V1.01.1
brsystem	Binaire	V1.11.3
CONVERT	Binaire	V1.10
DVFrame	Binaire	V3.42.5
mc_lib	Binaire	V1.60.1
ncglobal	Binaire	V0.37
OPERATOR	Binaire	V1.02
powerlnk	Binaire	V1.08.3
PP2005	Bibliothèque CEI	V0.00
ConvCamMac	Bloc de fonction	
ConCaisMac	Bloc de fonction	

Nom	Type	Domaine
m	ptXYZ	VAR_INPUT
n	ptXYZ	VAR_OUTPUT

Cinq Languages

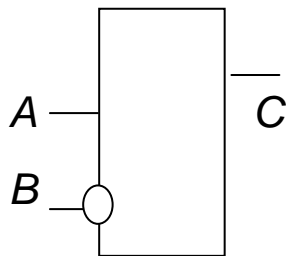
(*Instruction List*)

```
LD    A
ANDN  B
ST    C
```

(*Structured Text*)

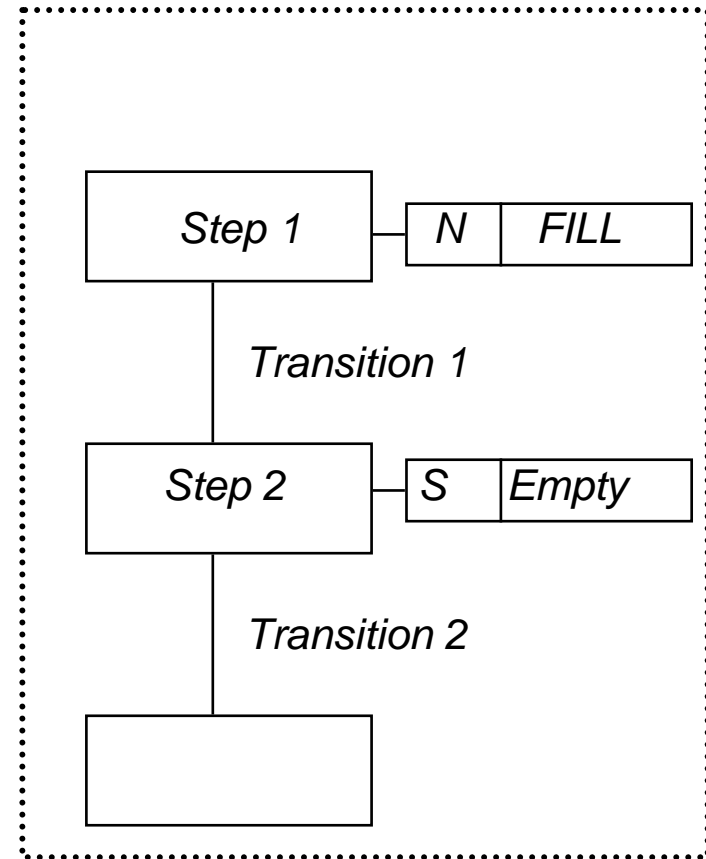
```
C := A AND NOT B
```

Function Block Diagram



Ladder Diagram

```
A B          C
-| |---|/|----- ( )
```



Fonctions Standards

La norme IEC 6 1131-3 décrit des fonctions standard qui peuvent être utilisées dans le programme API

Opérations sur les cordons de bits : **(AND, OR , XOR , NOT, SHL , SHR, ROL , ROR)**

Fonctions Numériques **(ADD , SUB , MUL , DIV , MOD , EXPT , ABS , SQRT , LN , LOG , EXP , SIN , COS , TAN , ASIN ,ACOS , ATAN)**

Conversions de types **(x_TO_y : ex. USINT_TO_DINT , BOOL_TO_BYTE)**

Fonctions de Sélections **(SEL , MIN , MAX , LIMIT , MUX)**

Fonctions de Comparaison **(GT, GE , EQ , LT , LE , NE)**

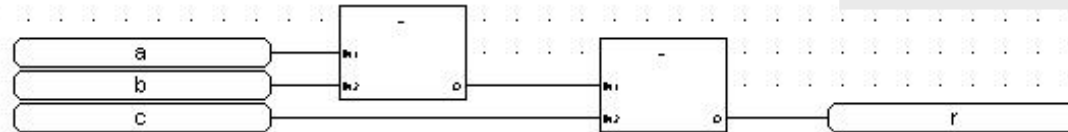
Fonctions de Chaînes de caractères **(LEN , LEFT, RIGHT, MID, CONCAT,INSERT, DELETE , REPLACE, FIND)**

Fonctions numériques

LD a
ADD b
ST r



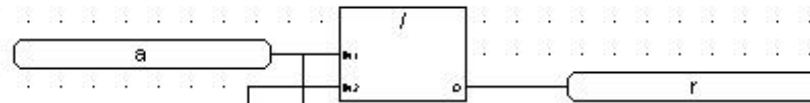
LD a
SUB b
SUB c
ST r



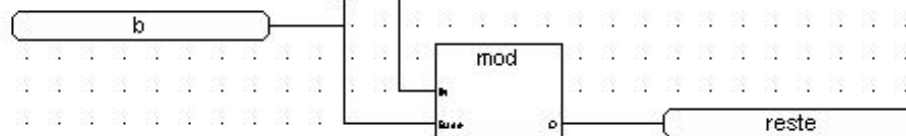
LD a
MUL b
ST r



LD a
DIV b
ST r



LD a
DIV b
ST r
LD a
MOD b
ST reste

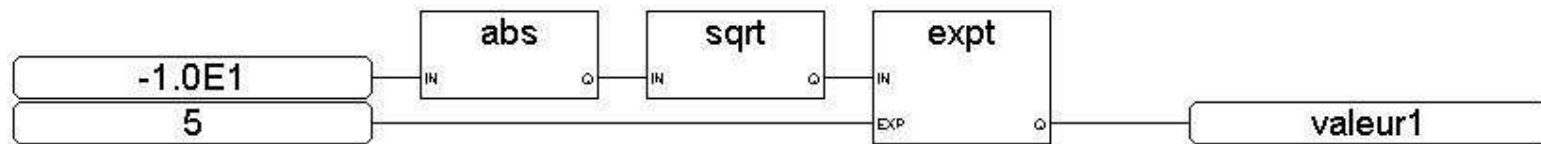


$r := a + b;$
 $r := a - b - c;$
 $r := a * b;$
 $r := a / b;$
(*division entière *)
 $r := (a / b);$
(* reste de la division *)
 $reste := MOD(a, b);$

Fonctions numériques

LD -1.0E1
ABS
SQRT
EXPT 5
ST Valeur1

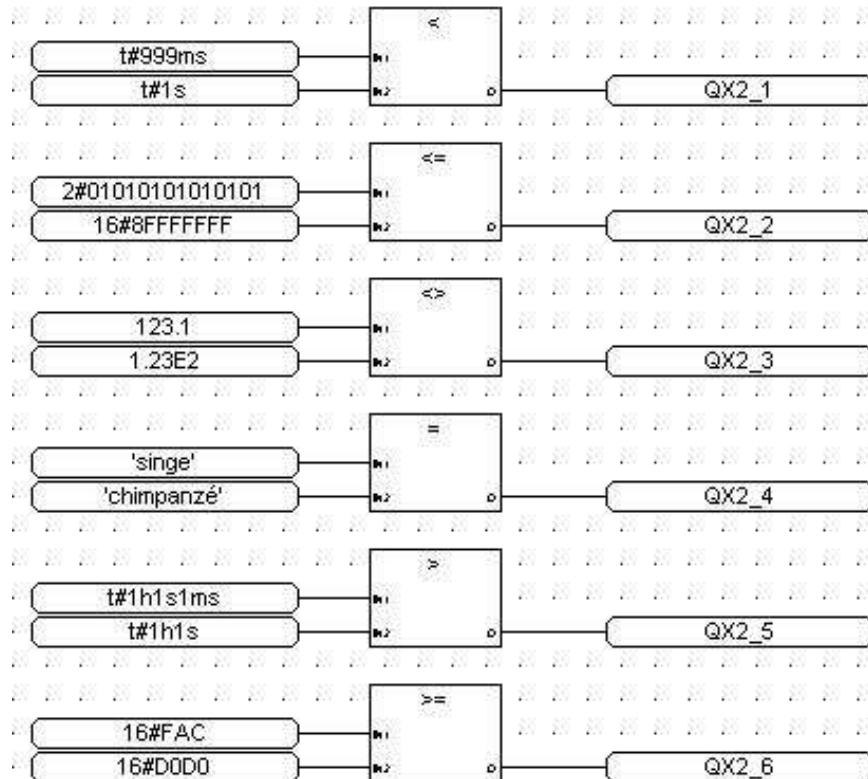
```
Valeur1:=expt(sqrt(abs(-1.0E1)),5);
```



ABS Valeur absolue
SQRT Racine carrée
EXPT Exponentiation
LOG Logarithme

Sortie dans l'intervalle [-1.0 ,+1.0]
COS Cosinus
SIN Sinus
TAN Tangente
Entrée dans l'intervalle [-1.0 ,+1.0]
Sortie dans l'intervalle [0.0, PI]
ACOS Arc cosinus
ASIN Arc sinus
ATAN Arc tangente

Fonctions de comparaisons



```

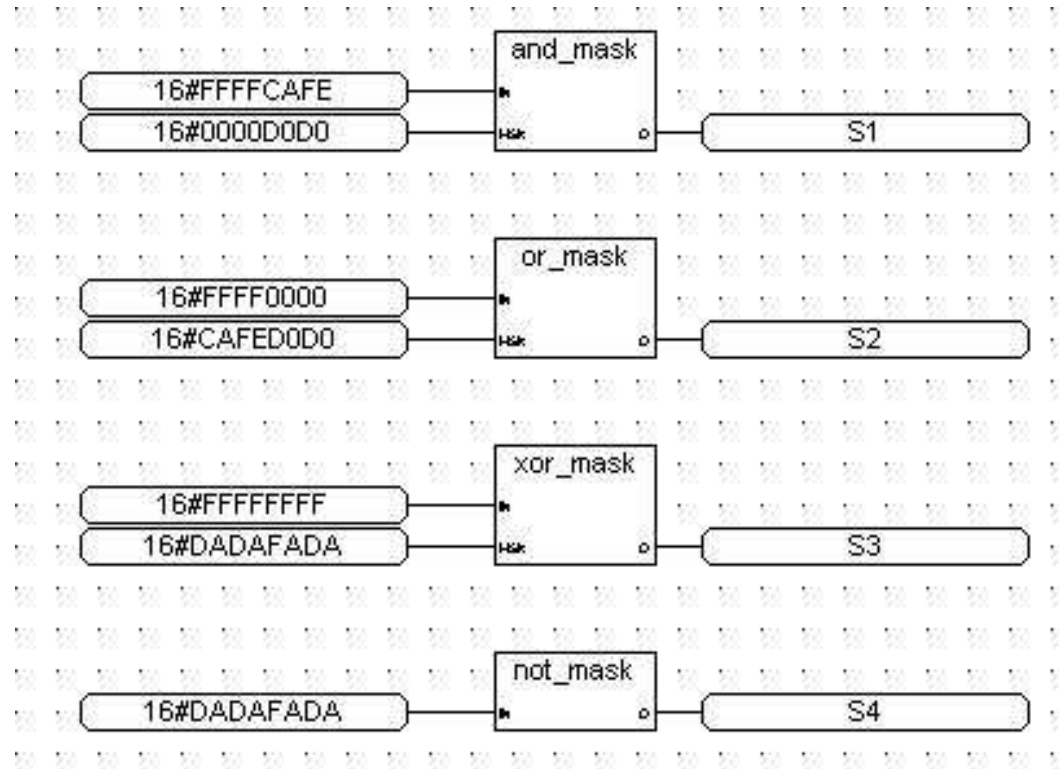
QX2_1 :=(t#999ms < t#1s) ;
QX2_2 :=(2#01010101010101 <= 16#8FFFFFFF) ;
QX2_3 :=(123.1 <> 1.23E2) ;
QX2_4 :=('singe' = 'chimpanzé') ;
QX2_5 :=(t#1h1s1ms > t#1h1s) ;
QX2_6 :=(16#FAC >=16#D0D0) ;
    
```

```

LD t#999ms
LT t#1s
ST QX2_1
LD 2#010101010101
LE 16#8FFFFFFF
ST QX2_2
LD 123.1
NE 1.23E2
ST QX2_3
LD 'singe'
EQ 'chimpanzé'
ST QX2_4
LD t#1h1s1ms
GT t#1h1sST QX2_5
LD 16#FACGE
16#D0D0
ST QX2_6
    
```



Opérations sur cordons de bits



```
S1 :=AND_MASK(16#FFFFFFCAFE,16#0000D0D0) ;  
S2 :=OR_MASK(16#FFFFFF0000,16#CAFED0D0) ;  
S3 :=XOR_MASK(16#FFFFFFFF,16#DADAFADA) ;  
S4 :=NOT_MASK(16#DADAFADA) ;
```

```
LD 16#FFFFFFCAFE  
AND_MASK 16#0000D0D0  
ST S1  
LD 16#FFFFFF0000  
OR_MASK 16#CAFED0D0  
ST S2  
LD 16#FFFFFFFF  
XOR_MASK  
16#DADAFADA  
ST S3  
LD 16#DADAFADA  
NOT_MASK S4
```

```
S1=  
S2=  
S3=  
S4=
```

Avertissement

Les exemples de masquage sont issus de Isagraf. Le `_mask` permettant de faire la différence avec les opérateurs logiques (sur bit). En fait, la norme ne fait pas la différence. Voir cet exemple sous Mitsubishi.

The screenshot displays two windows from a software development environment:

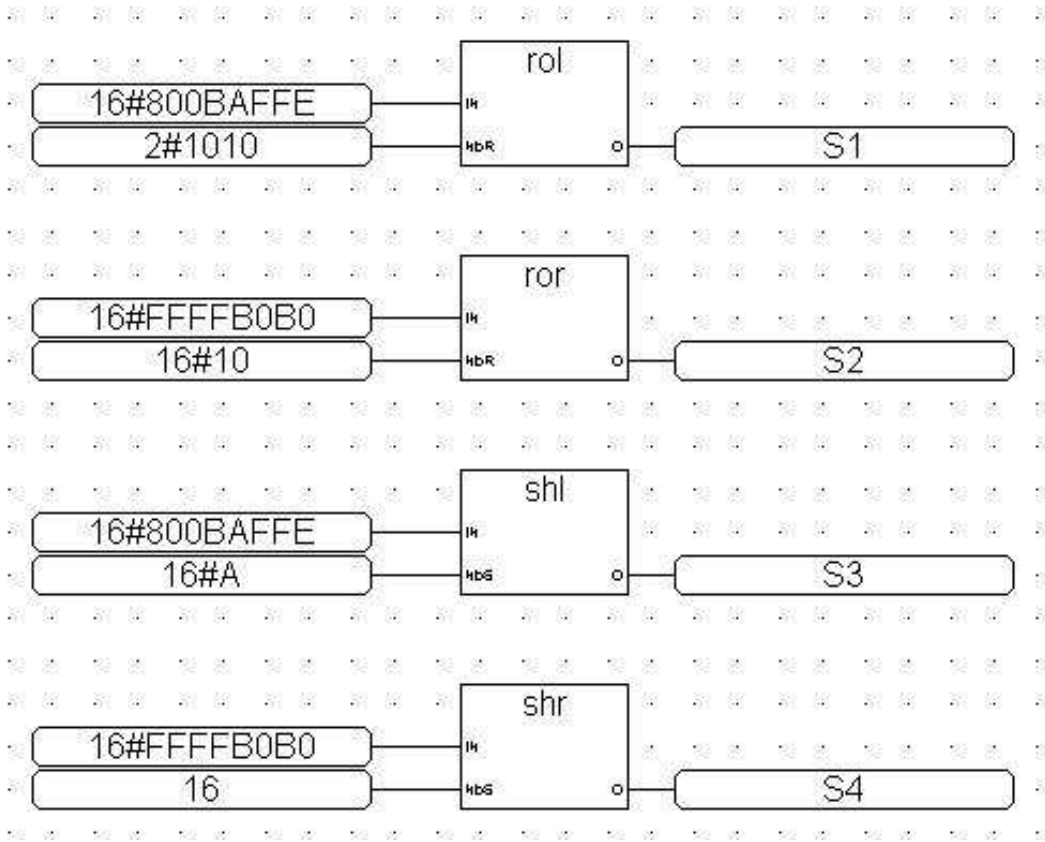
- POU_3 [PRG] Header:** A table listing variable declarations.
- POU_3 [PRG] Body [LD]:** A logic editor showing two XOR operations.

	Class	Identifier	Type	
0	VAR_EXTERNAL	WORD_1	WORD	0
1	VAR_EXTERNAL	INT_1	INT	0
2	VAR_EXTERNAL	BOOL_1	BOOL	F.

The logic editor shows two XOR operations:

- Top XOR: `WORD_1` and `16#CAFE` are inputs, and `WORD_1` is the output.
- Bottom XOR: `BOOL_1` and `TRUE` are inputs, and `BOOL_1` is the output.

Opérations sur cordons de bits



```
S1 :=rol(16#800BAFFE,2#1010) ;  
S2 :=ror(16#FFFFB0B0,16#10) ;  
S3 :=shl(16#800BAFFE,16#A) ;  
S4 :=shr(16#FFFFB0B0,16);
```

```
LD 16#800BAFFE  
ROL 2#1010  
ST S1  
LD 16#FFFFB0B0  
ROR 16#10  
ST S2  
LD 16#800BAFFE  
SHL 16#A  
ST S3  
LD 16#FFFFB0B0  
SHR 16  
ST S4
```

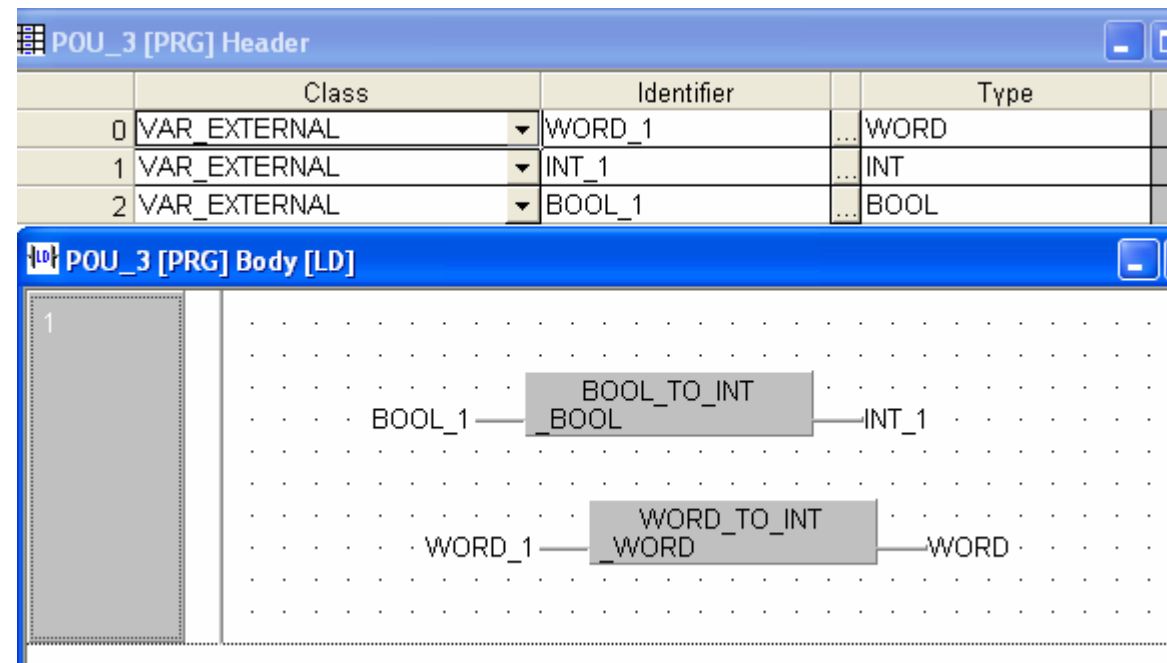
S1=
S2=
S3=
S4=

Conversion de type

Instruction de la forme `*_TO_*` permettant la conversion de type

Exemple

```
A := INT_TO_REAL (198);    (*A = 198.0 *)
```



Une liste exhaustive ne présente aucun intérêt.

Manipulation des chaînes de caractères

	Graphical form	Explanation/example
	<pre> +-----+ STRING--- LEN ---INT +-----+ </pre>	<p>String length function Example: A := LEN('ASTRING') is equivalent to A := 7;</p>
	<pre> +-----+ LEFT STRING--- IN ---STRING UINT----- L +-----+ </pre>	<p>Leftmost L characters of IN Example: A := LEFT('ASTR',3); is equivalent to A := 'AST' ;</p>
	<pre> +-----+ RIGHT STRING--- IN ---STRING UINT----- L +-----+ </pre>	<p>Rightmost L characters of IN Example: A := RIGHT('ASTR',3); is equivalent to A := 'STR' ;</p>
	<pre> +-----+ MID STRING--- IN ---STRING UINT----- L UINT----- P +-----+ </pre>	<p>L characters of IN, beginning at the P-th Example: A := MID('ASTR',2,2); is equivalent to A := 'ST' ;</p>

Manipulation des chaînes de caractères

```
      +-----+
      |   CONCAT   |
STRING---|         |---STRING
      |         |
      |         |
      |         |
      |         |
      |         |
      |         |
      +-----+
```

Extensible concatenation

Example:

```
A :=
CONCAT('AB','CD','E') ;
is equivalent to
A := 'ABCDE' ;
```

```
      +-----+
      |   INSERT   |
STRING---| IN1     |---STRING
STRING---| IN2     |
UINT-----| P      |
      |         |
      |         |
      |         |
      +-----+
```

Insert IN2 into IN1 after the P-th character position

Example:

```
A:=INSERT('ABC',XY',2);
is equivalent to
A := 'ABXYC' ;
```

```
      +-----+
      |   DELETE   |
STRING---| IN      |---STRING
UINT-----| L      |
UINT-----| P      |
      |         |
      |         |
      |         |
      +-----+
```

Delete L characters of IN, beginning at the P-th character position

Example:

```
A :=
DELETE('ABXYC',2,3) ;
is equivalent to
A := 'ABC' ;
```

Manipulation des chaînes de caractères

```
      +-----+
      | REPLACE |
STRING---| IN1   |---STRING
STRING---| IN2   |
UINT-----| L    |
UINT-----| P    |
      +-----+
```

Replace L characters of IN1 by IN2, starting at the P-th character position

Example:

```
A :=
REPLACE('ABCDE','X',2,
3);
is equivalent to
A := 'ABXE' ;
```

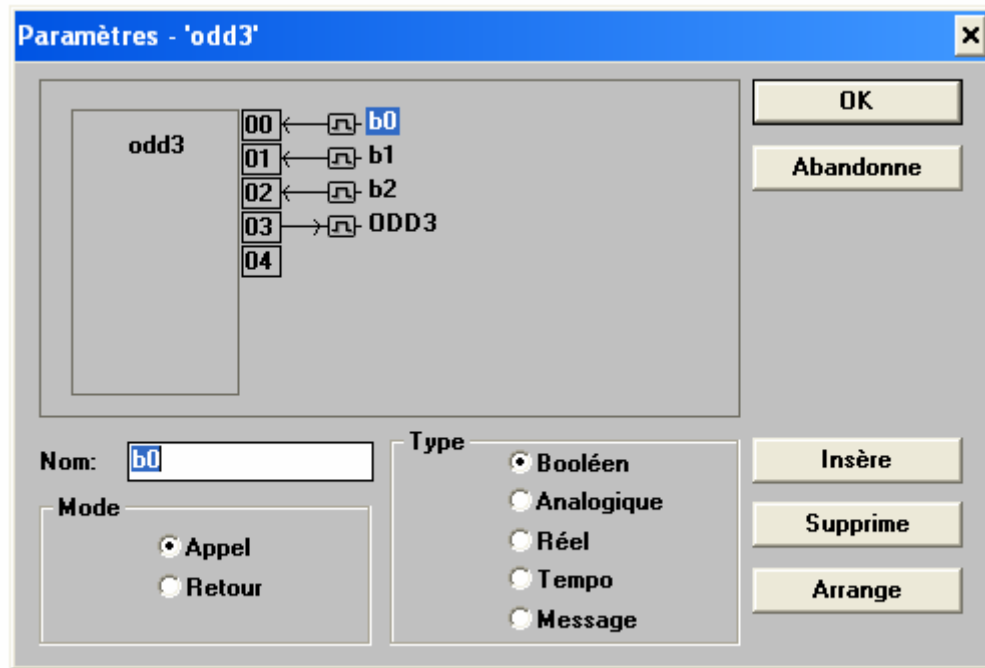
```
      +-----+
      | FIND    |
STRING---| IN1   |---INT
STRING---| IN2   |
      +-----+
```

Find the character position of the beginning of the first occurrence of IN2 in IN1. If no occurrence of IN2 is found, then OUT := 0.

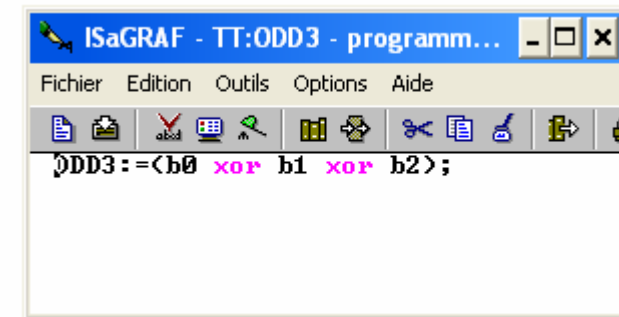
Example:

```
A :=
FIND('ABCBC','BC') ;
is equivalent to A := 2 ;
```

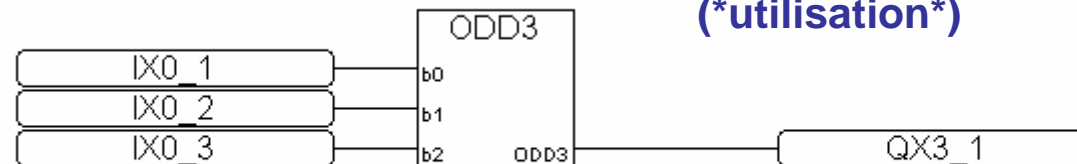
Exemple de fonction avec Isagraf



(*définition*)



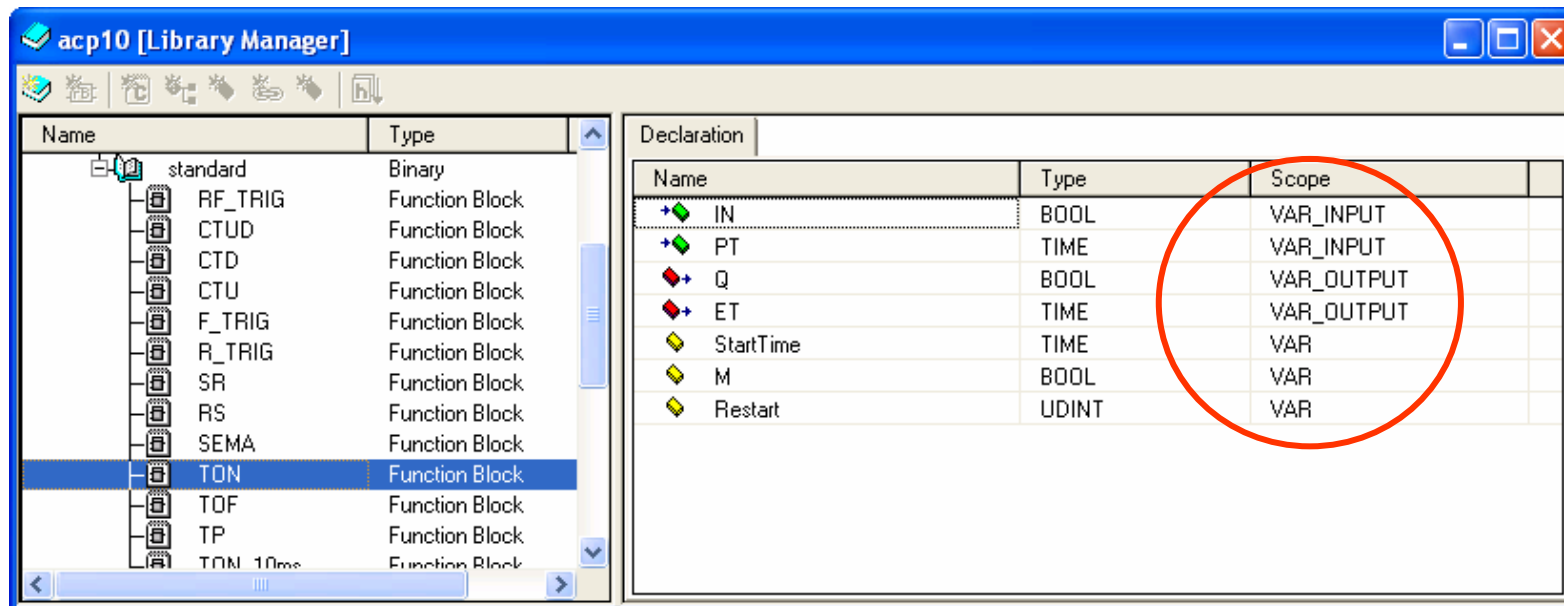
(*utilisation*)



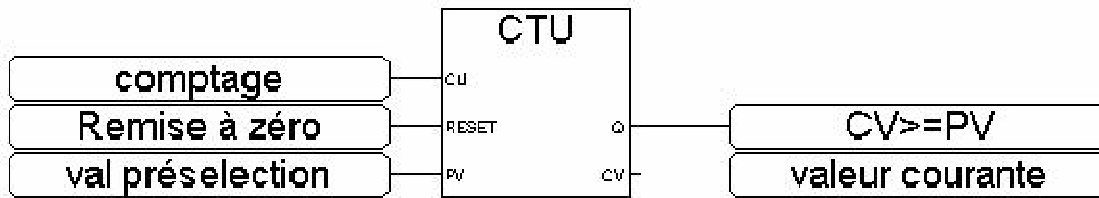
Blocs fonctionnels standards

La norme IEC 6 1131-3 décrit des blocs fonctionnels standard qui peuvent être utilisés dans le programme API

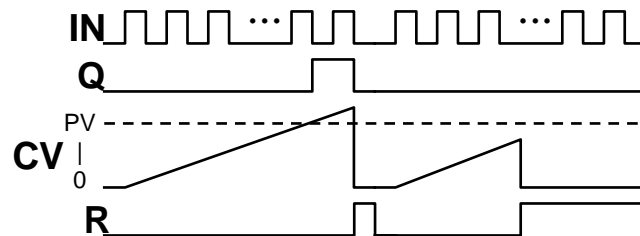
Compteurs **CTU, CTD, CTUD**
DéTECTEURS de fronts **R_TRIG, F_TRIG**
Temporisateurs **TP, TON, TOF, RTC**
Bascules **SR, RS**



CTU

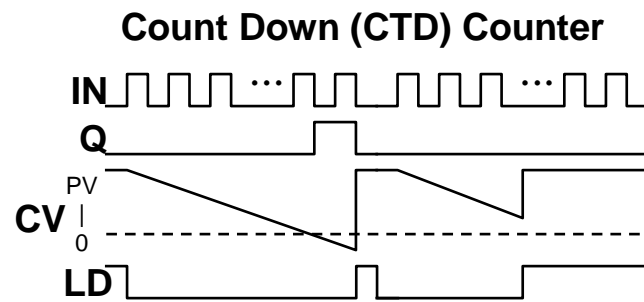
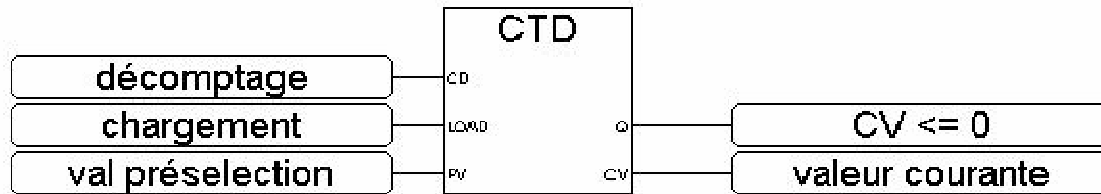


Count Up (CTU) Counter



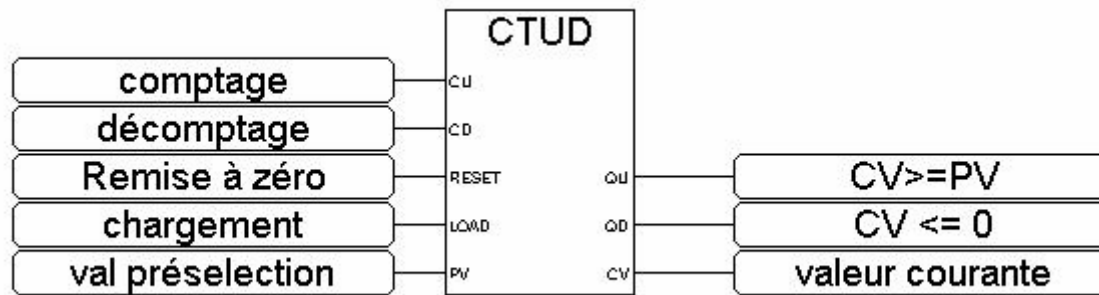
```
(*CTU*)  
IF R THEN CV := 0 ;  
ELSIF CU AND (CV < PVmax)  
  THEN CV := CV+1 ;  
END_IF ;  
Q := (CV >= PV) ;
```

CTD

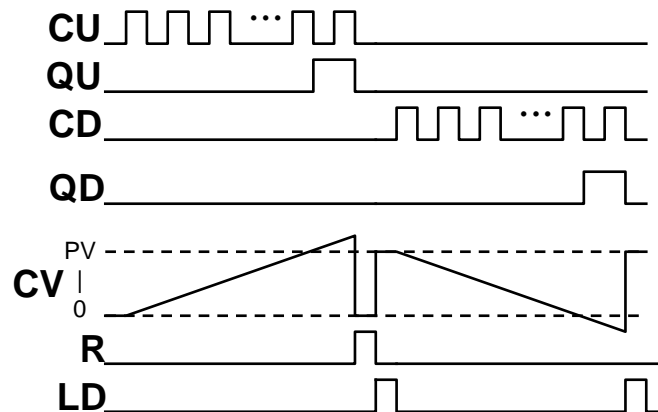


```
(*CTD*)  
IF LD THEN CV := PV ;  
ELSIF CD AND (CV > PVmin)  
  THEN CV := CV-1;  
END_IF ;  
Q := (CV <= 0) ;
```

CTUD



Count Up/Down (CTUD) Counter



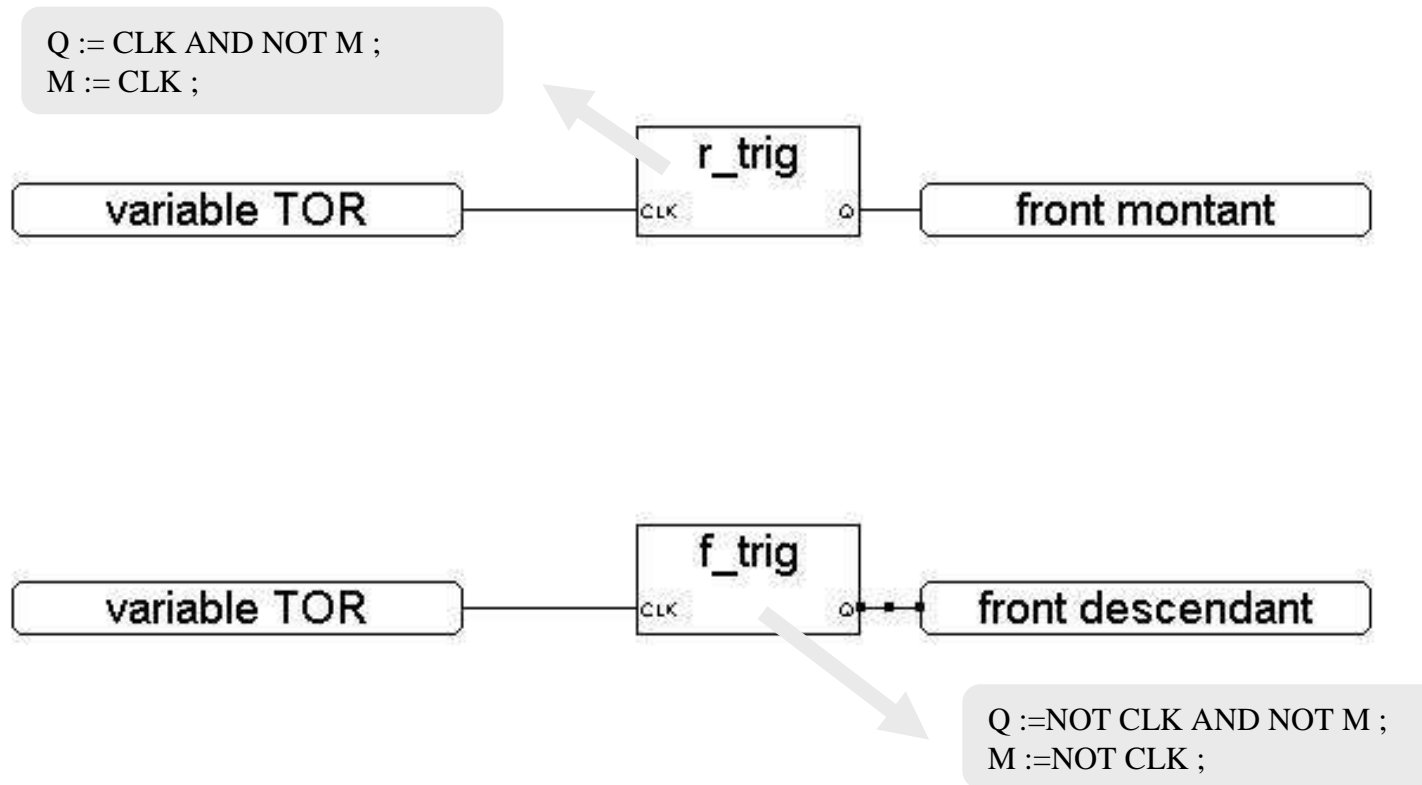
(*CTUD*)

```

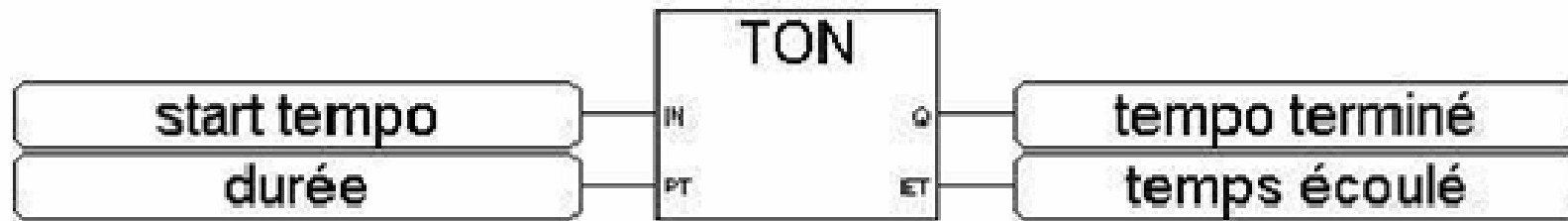
IF R THEN CV := 0 ;
ELSIF LD THEN CV := PV ;
ELSE
  IF NOT (CU AND CD) THEN
    IF CU AND (CV < PVmax)
      THEN CV := CV+1;
    ELSIF CD AND (CV > PVmin)
      THEN CV := CV-1;
    END_IF;
  END_IF;
END_IF ;
QU := (CV >= PV) ;
QD := (CV <= 0) ;

```

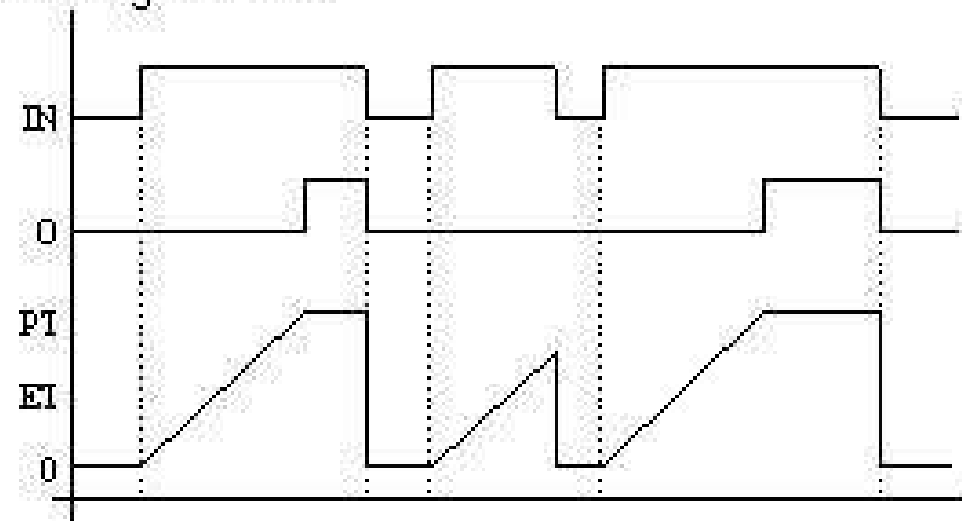
F_TRIG & R_TRIG



TON



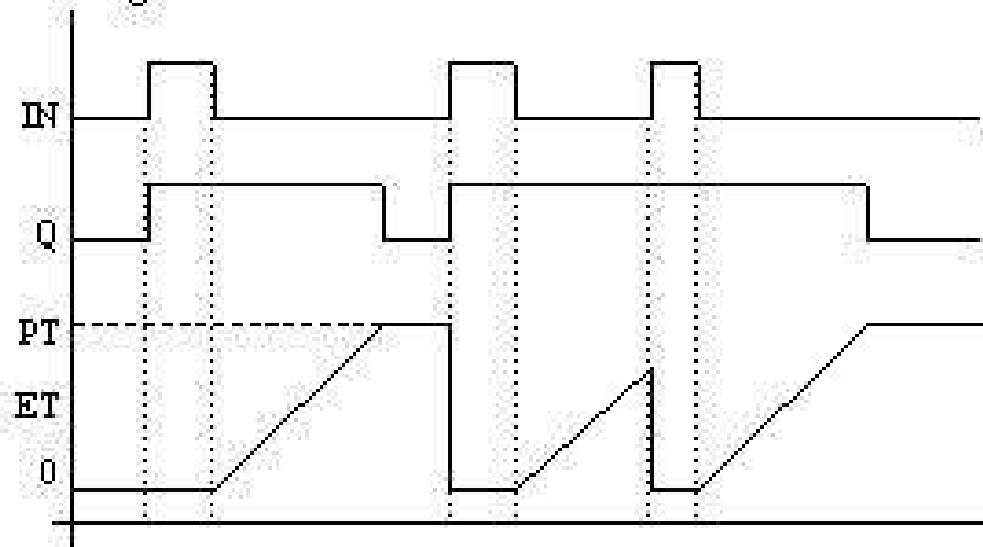
Chronogramme :



TOFF

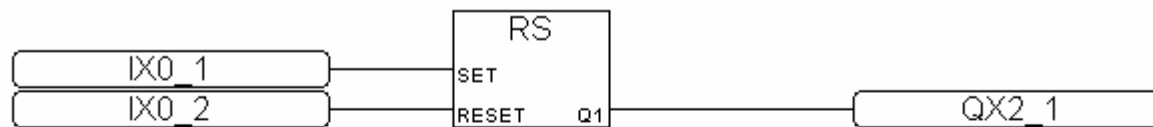


Chronogramme :



Bascule RS & SR

SET	BOO	si TRUE, force Q1 à TRUE
RESET1	BOO	si TRUE, force Q1 à FALSE (prioritaire)
Q1	BOO	état de la bascule



SET1	BOO	si TRUE, force Q1 à TRUE (prioritaire)
RESET	BOO	si TRUE, force Q1 à FALSE
Q1	BOO	état de la bascule

Exemple de block avec Siemens

The image displays the Siemens SIMATIC Manager interface for a project named 'Projet1 (CPU 221)'. The project tree on the left shows the structure of the project, including a code block 'PPAL (OB1)' containing a sub-block 'SBR_0 (SBR0)'. The main workspace is divided into two windows, both titled 'CONT SIMATIC'.

The top window shows a variable declaration table for the block:

	Nom	Type var.	Type donné~	Commentaire
LW0	A	TEMP	INT	
		TEMP		
		TEMP		
		TEMP		

The bottom window shows the ladder logic for 'Réseau 1' (Network 1) with the title 'TITRE DE RESEAU (ligne unique)'. It features a block call for 'SBR_0' with the following connections:

- EN: Connected to SM0.0
- Nb: Connected to 16#CAFE
- Q: Connected to M0.0

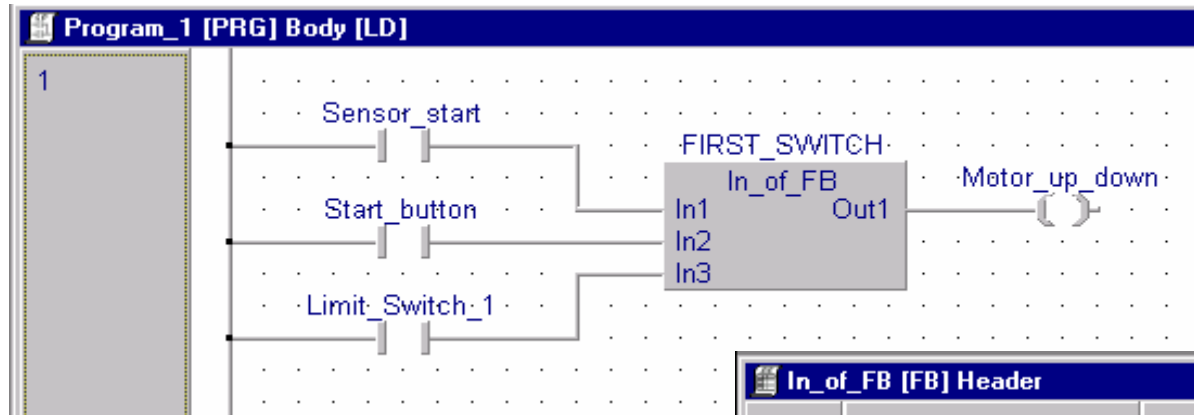
The bottom-left window shows a detailed view of the ladder logic for 'Réseau 1' and 'Réseau 2' within the 'SBR_0' block:

Réseau 1: A normally open contact labeled 'SM0.0' is connected to the 'EN' input of a 'DIV' (Divide) block. The 'ENO' output of the 'DIV' block is connected to a coil. The 'IN1' input is labeled '#Nb' and the 'IN2' input is labeled '+2'. The 'OUT' output is labeled '#Re'.

Réseau 2: A normally open contact labeled 'SM0.0' is connected to the 'EN' input of a 'SHR_DW' (Shift Right Double Word) block. The 'ENO' output of the 'SHR_DW' block is connected to a coil labeled '#Rd ==D +1'. The 'IN' input is labeled '#Re' and the 'OUT' output is labeled '#Rd'. The block size is indicated as '16-N'.

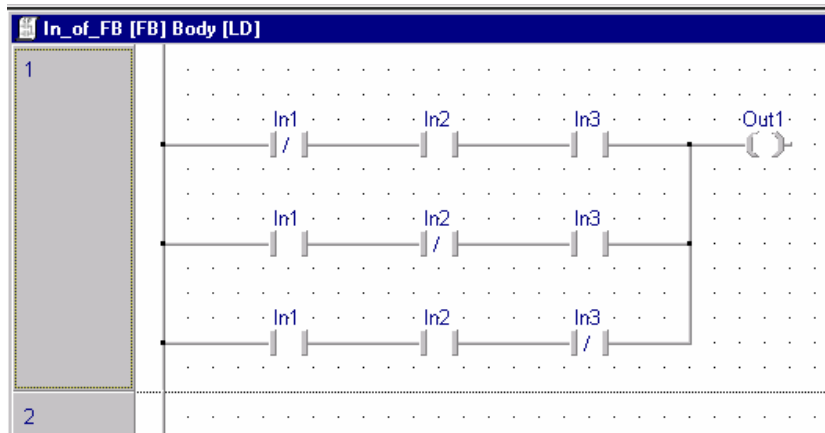
The bottom status bar shows the current network: 'PPAL \ SBR_0 /'.

Exemple de block avec NAIS



In_of_FB [FB] Header

	Class	Identifier	Type	Initial	C
0	VAR_INPUT	In1	BOOL	FALSE	
1	VAR_INPUT	In2	BOOL	FALSE	
2	VAR_INPUT	In3	BOOL	FALSE	
3	VAR_OUTPUT	Out1	BOOL	FALSE	



Exemple de block avec B&R

The screenshot displays the B&R Automation Studio interface. On the left, a tree view shows various libraries, with 'ConvMacRob' selected. The main window is divided into a declaration table and a code editor.

Nom	Type	Version	Déclaration
m	ptXYZ		VAR_INPUT
n	ptROB		VAR_OUTPUT

```
0001 (* Implémentation de ConvMacRob *)
0002
0003
0004 (* passage du repère machine vers le repère robot*)
0005
0006 (*calcul du déplacement suivant Xr *)
0007
0008 n.xr := SQRT(m.x*m.x+m.y*m.y)-X0;
0009
0010 (* calcul de l'angle de rotation sur 360 ° pour Rz*)
0011
0012 IF (m.x=0) AND (m.y=0) THEN n.rz:=0.0;
0013 ELSIF (m.x=0) AND (m.y <0) THEN n.rz:=-90.0;
0014 ELSIF (m.x=0) AND (m.y>0) THEN n.rz:=90.0;
0015 ELSIF (m.x>0) AND (m.y <>0) THEN n.rz:=(180.0/PI)*ATAN(m.y/m.x);
0016 ELSIF (m.x>0) AND (m.y=0) THEN n.rz:=0.0;
0017 ELSIF (m.x<0) AND (m.y<0) THEN n.rz:=- (180.0/PI)* (PI-ATAN(m.y/m.x));
0018 ELSIF (m.x<0) AND (m.y>0) THEN n.rz:=(180.0/PI)* (PI+ ATAN(m.y/m.x));
0019 ELSIF (m.x<0) AND (m.y=0) THEN n.rz:=-180.0;
0020 END_IF;
0021
0022 (*déplacement suivant Zr *)
0023
0024 n.zr:=m.z;
0025 (*déplacement suivant Oz *)
0026 n.oz:=m.o;
0027
```

Avantages

Standard International

Pratiquement tous les grands constructeurs des automates proposent des plateformes orientées vers IEC 6 1131 - 3

Structures de l'interface, langages et l'organisation des programmes sont uniformes

Economie du temps

Modèle unique du software et des données pour les différents types d'automates

Il suffit d'apprendre une fois pour programmer les différents API

Risque d'erreurs réduit

Fonctions et Blocs Fonctionnels standard

Réutilisation et portabilité du software

Avantages

Qualité de programmation

Indépendance du constructeur et du type de l'automate

Structuration des programmes et déclarations des variables comme dans des langages informatiques

Utilisation de données typées et programmation symbolique permettent d'éviter des erreurs de programmation

Le choix d'un meilleur langage pour chaque problème

5 langages de programmation (textuels et graphiques)

Possibilité de mixer différents langages dans le projet utilisateur

IEC 61131-3



www.plcopen.org

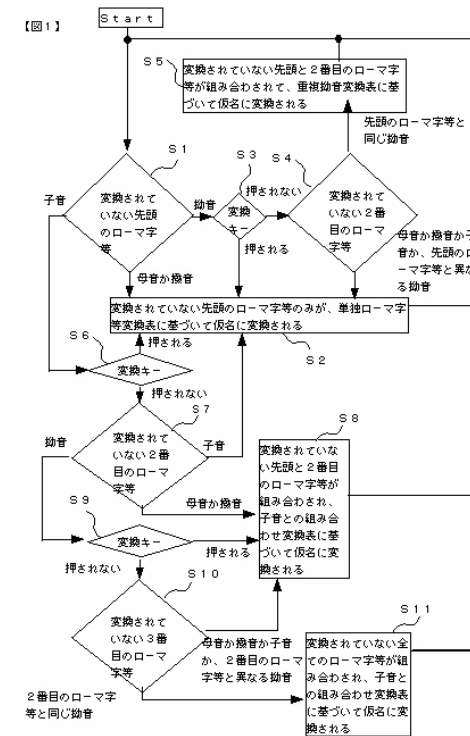
“The best thing that happened to industrial control”

IEC 61131-3 Algorithmique

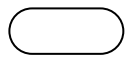
« Un algorithme est un ensemble de règles opératoires rigoureuses ordonnant, à un processeur particulier d'exécuter, dans un ordre déterminé, un nombre fini d'opérations élémentaires pour résoudre tous les problèmes d'un type donné. »

Notes de cours
Philippe RAYMOND
octobre 2005

【書類名】図面



NFZ 61-100



Début, fin, interruption

Début, fin ou interruption d'un programme, point de contrôle, ...

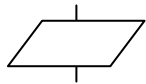
Symbole général "traitement"

Opération ou groupe d'opérations sur des données, instructions



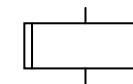
Entrée-sortie

Mise à disposition d'une information à traiter ou enregistrement d'une information traitée.



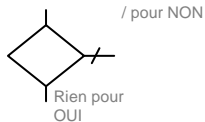
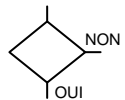
Sous-programme

Portion de programme considérée comme une simple opération



Embranchement ou test logique

Prise en compte d'une condition variable impliquant un choix parmi plusieurs voies



Algorithmique en langage ST

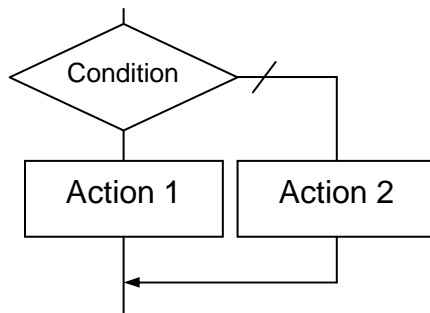
structure conditionnelle : elle n'offre que deux issues possibles à la poursuite de l'algorithme en s'excluant mutuellement.

Structure itérative : la structure itérative répète l'exécution d'une opération ou d'un traitement.

Structure de choix : elle permet d'effectuer des actions différentes suivant les valeurs que peut prendre une même variable

structure conditionnelle

[IF ... THEN... ELSE... END_IF]



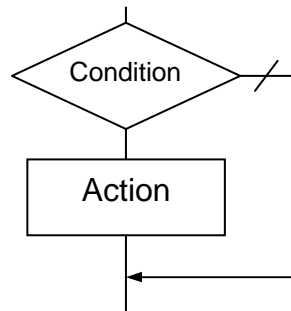
```
IF <condition>
THEN "action1"
ELSE "action 2"
END_IF
```

(*EXEMPLE*)

```
IF manual AND not (alarm) THEN
  level := manual_level;
  bx126 := bi12 OR bi45;
ELSIF over_mode THEN
  level := max_level;
ELSE
  level := (lv16 * 100) / scale;
END_IF;
```

structure conditionnelle

[IF ... THEN...END_IF]



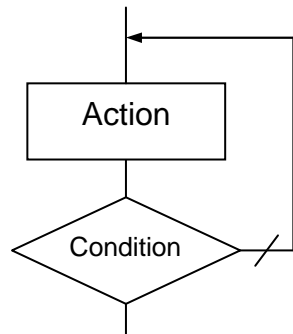
```
IF <condition>
THEN
"action"
END_IF
```

(*EXEMPLE*)

```
If overflow THEN
alm_level := true;
END_IF;
```

Structure itérative

[REPEAT ... UNTIL...END REPEAT]



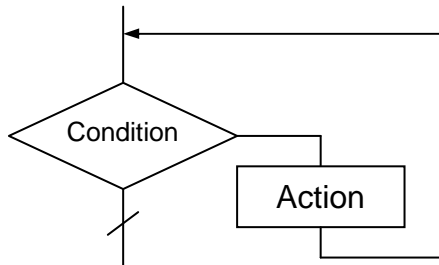
```
REPEAT "Action"  
UNTIL <condition>  
END_REPEAT
```

L'action est toujours exécutée
au moins une fois

```
(*EXEMPLE*)  
string := ""; (* chaîne vide *)  
nbchar := 0;  
IF ComIsReady ( ) THEN  
  REPEAT  
    string := string +ComGetChar ( );  
    nbchar := nbchar + 1;  
  UNTIL ( (nbchar >= 16) OR NOT  
    (ComIsReady ( ) ) )  
  END_REPEAT;  
END_IF;
```

Structure itérative

[WHILE ... DO ... END_WHILE]



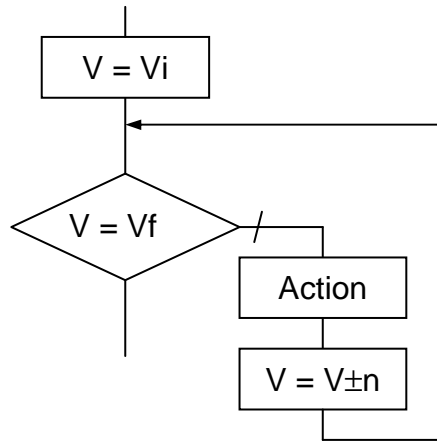
WHILE < condition >
DO "Action"
END_WHILE

L'action peut ne jamais être exécutée

(*EXEMPLE*)
string := ""; (* chaîne vide *)
nbchar := 0;
WHILE ((nbchar < 16) & ComIsReady ()) DO
string := string + ComGetChar ();
nbchar := nbchar + 1;
END_WHILE;

Structure itérative

[FOR ... TO ... BY ... DO ... END_FOR]



FOR $V = V_i$ **TO** V_f **BY** n **DO**
"Action"
END_FOR

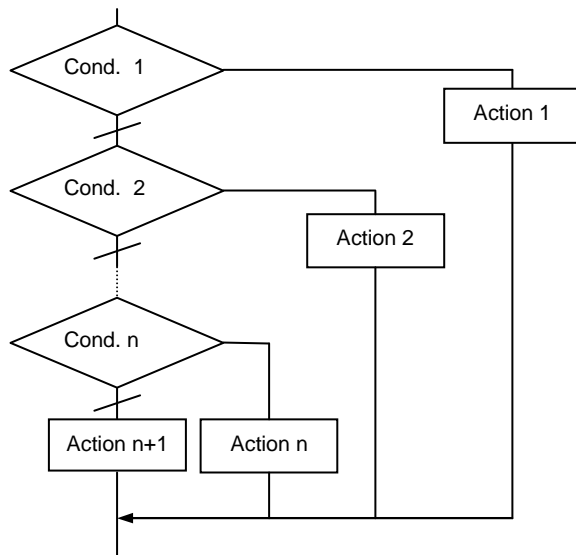
La mention [BY n] est optionnelle.
Lorsqu'elle n'est pas spécifiée,
le pas d'incrément par défaut est 1.

(*EXEMPLE*)

```
length := mlen (message);  
target := ""; (* chaîne vide *)  
FOR index := 1 TO length DO  
code := ascii (message, index);  
IF (code >= 48) & (code <= 57) THEN  
target := target + char (code);  
END_IF;  
END_FOR;
```

Structure de choix

[CASE ... OF... ELSE... END_CASE]

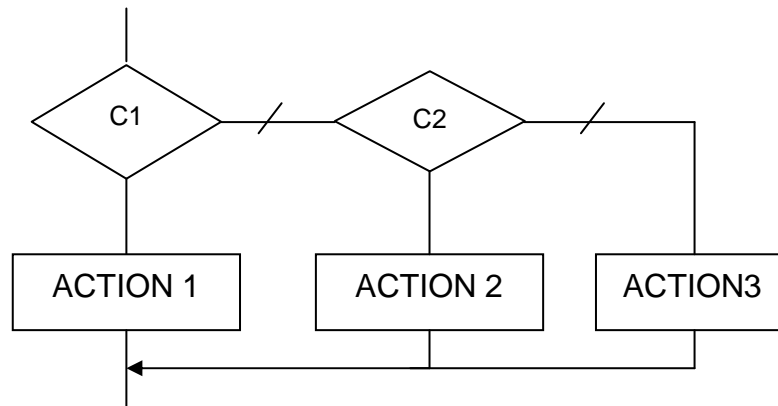


CASE var **OF**
Valeur 1 : "action 1"
Valeur 2 : "action 2"
.....
Valeur n : "action n"
ELSE action n+1
END_CASE

(*EXEMPLE*)

```
CASE error_code OF
255:
err_msg := 'Division by zero';
fatal_error := TRUE;
1:
err_msg := 'Overflow';
2, 3:
err_msg := 'Bad sign';
ELSE
err_msg := 'Unknown error';
END_CASE;
```

Autre réalisation (choix multiple)

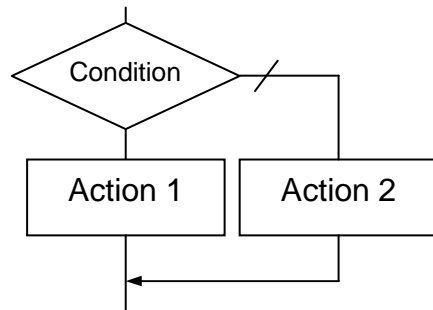


(*Ci réalise le test
(var = valeur i) *)

```
IF C1 THEN  
ACTION1  
ELSIF C2 THEN  
ACTION2  
ELSE  
ACTION3  
END_IF
```

Programmation en IL

[IF ... THEN... ELSE... END IF]



LD condition

JMPNC *action2* (*saut a l'étiquette *action2* si condition est faux*)

(* ici la description de l'action 1*)

JMP suite (*traitement de l'action 1 termine. poursuite du programme*)

action2:

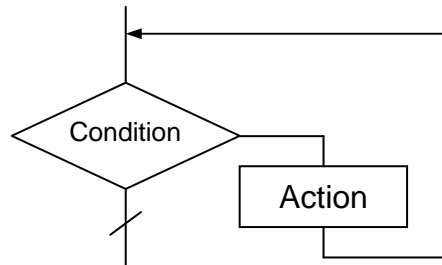
(* ici la description de l'action 2*)

suite:

(* ici la suite du traitement *)

Programmation en IL

[WHILE ... DO ... END WHILE]



JMP Jump to label
N (not) et **C** (condition) pour modifier
l'instruction **JUMP**

debut :

LD condition

JMPNC *suite* (*la condition est fausse, donc on saute à la suite *)

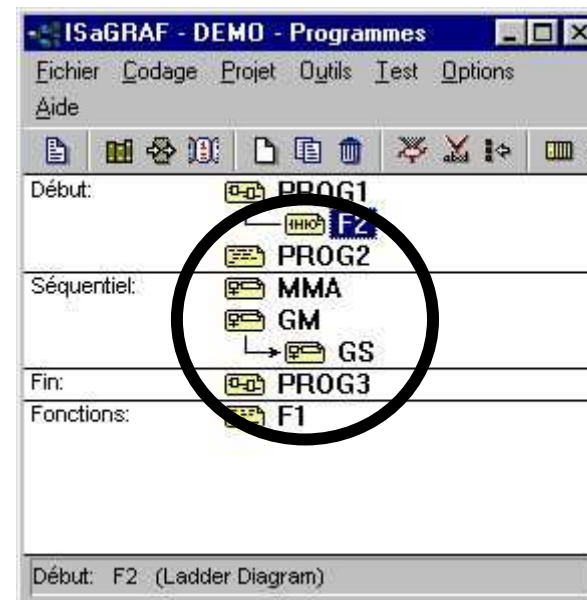
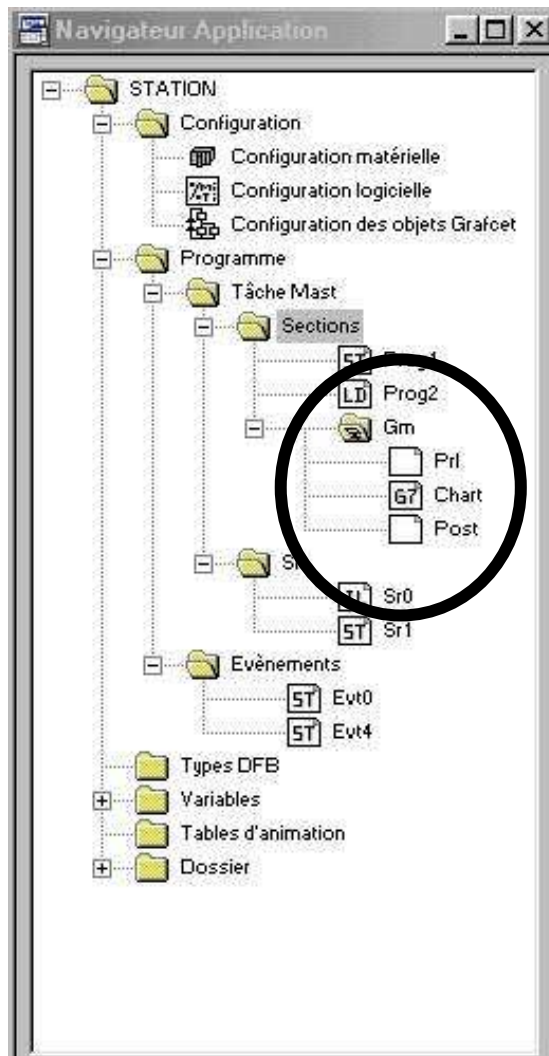
(* ici la description de l'action *)

JMP *debut* (*on retourne tester la condition*)

suite:

(* ici la suite du traitement *)

IEC 61131-3 Sequential Function Chart

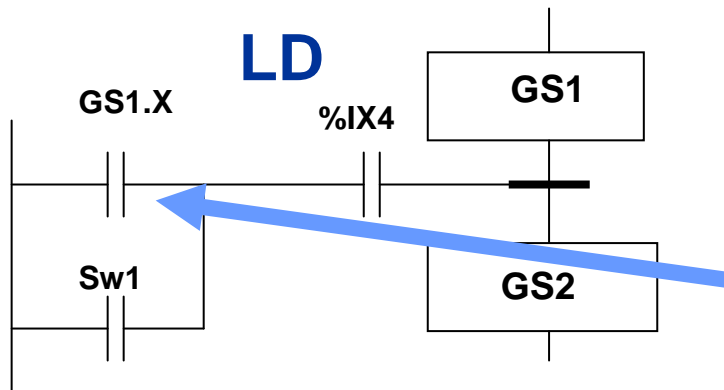
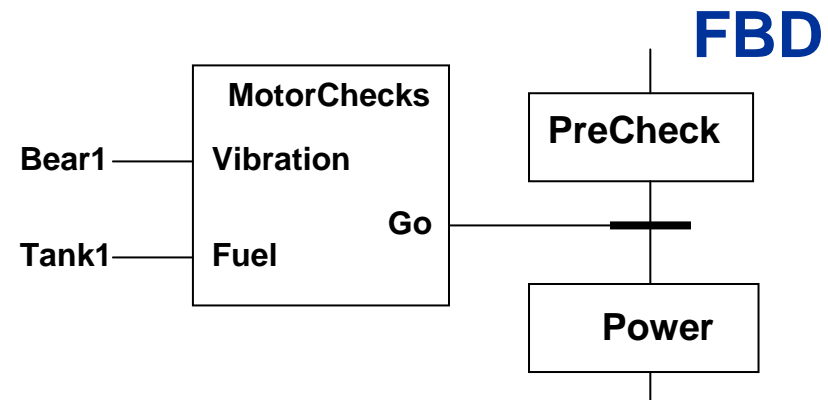
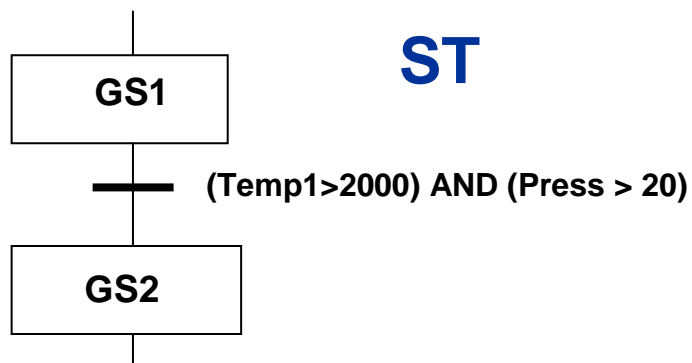


Notes de cours

Philippe RAYMOND

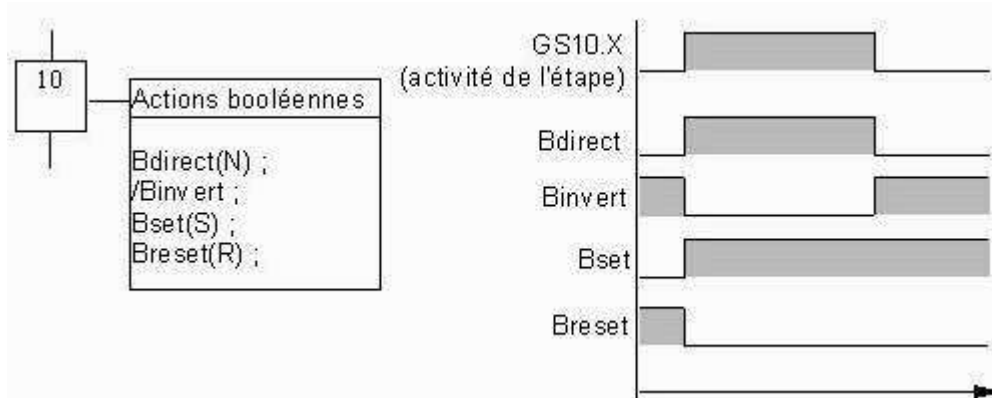
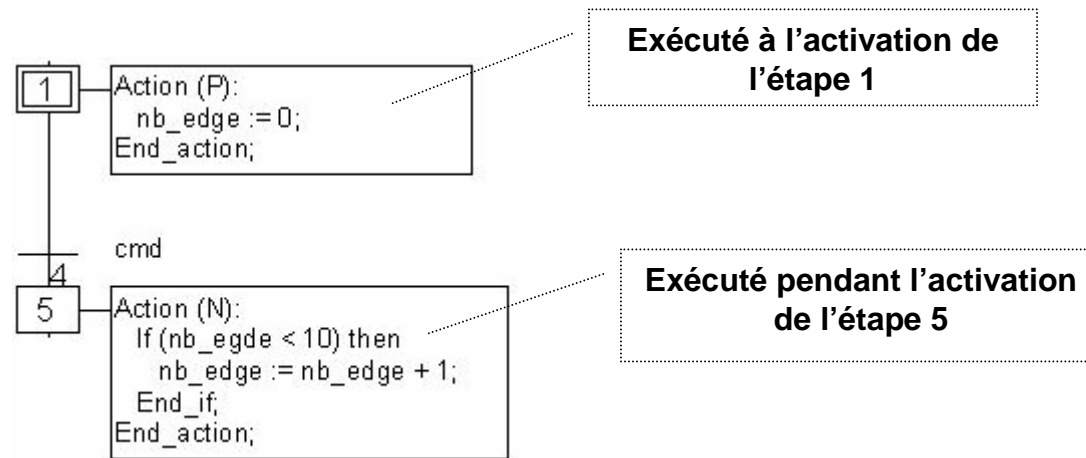
mars 2006

Programmation Des Réceptivités



***.X drapeau d'étape
(type booléen, domaine local)

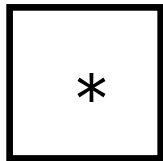
Bloc Actions exemples



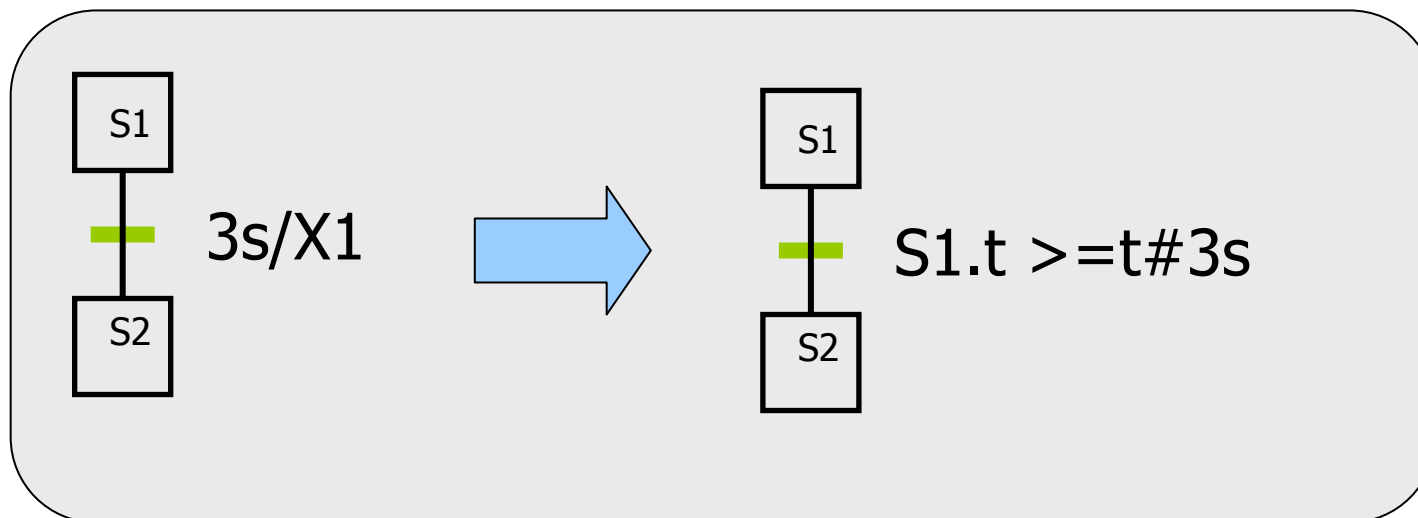
Bloc Actions possible

Qualifier	Explanation
None	Non-stored (null qualifier)
N	Non-stored
R	overriding Reset
S	Set (Stored)
L	time Limited
D	time Delayed
P	Pulse
SD	Stored and time Delayed
DS	Delayed and Stored
SL	Stored and time Limited

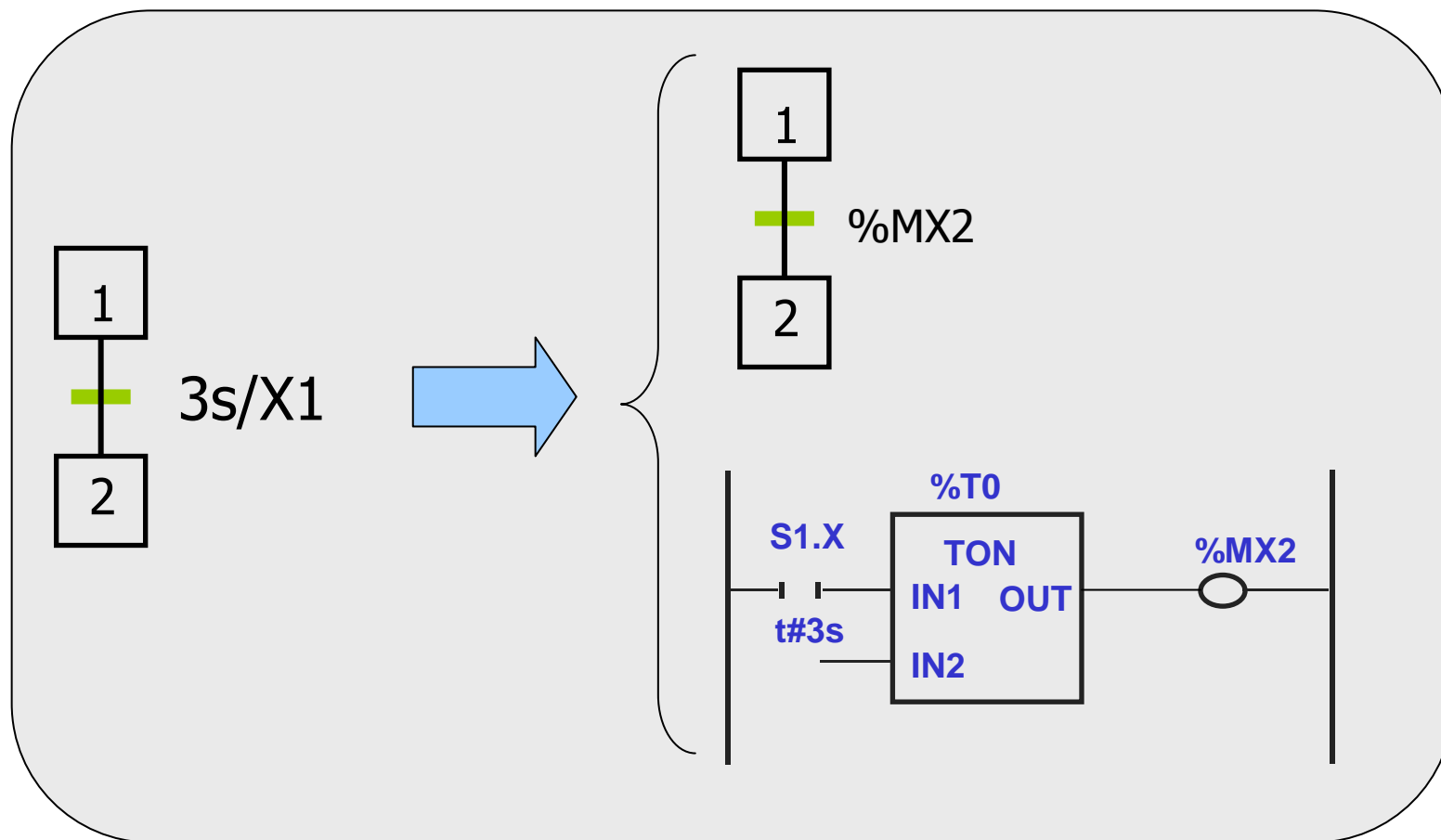
Réalisation De to/Xi



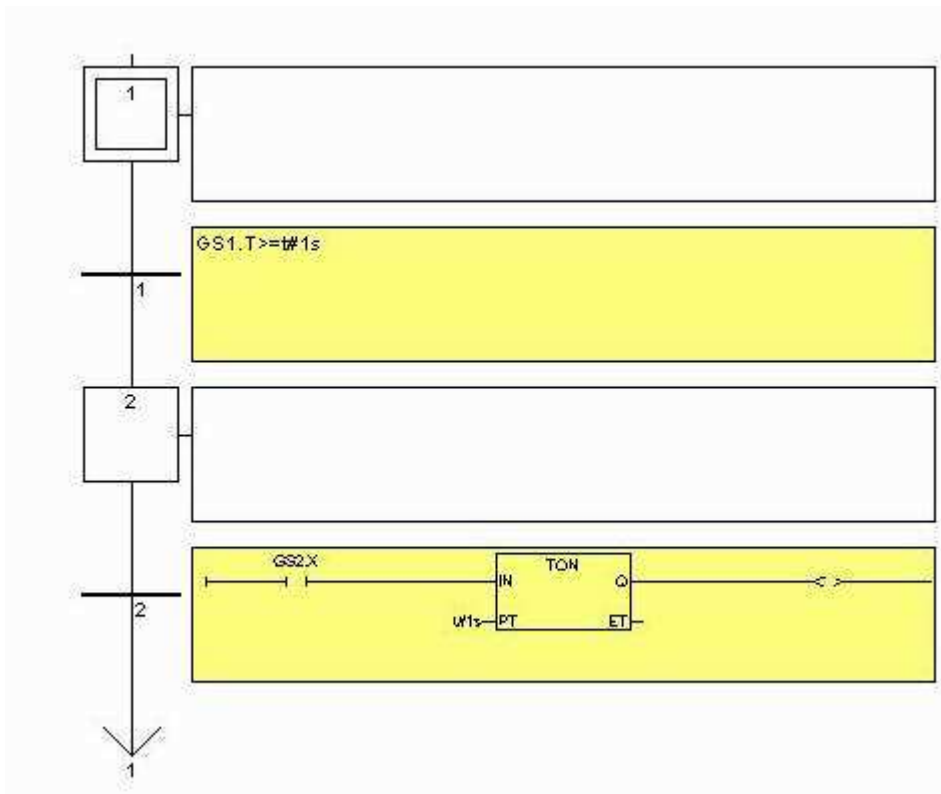
*.t temps écoulé depuis l'activation (type time)



Réalisation De to/Xi (autre méthode)



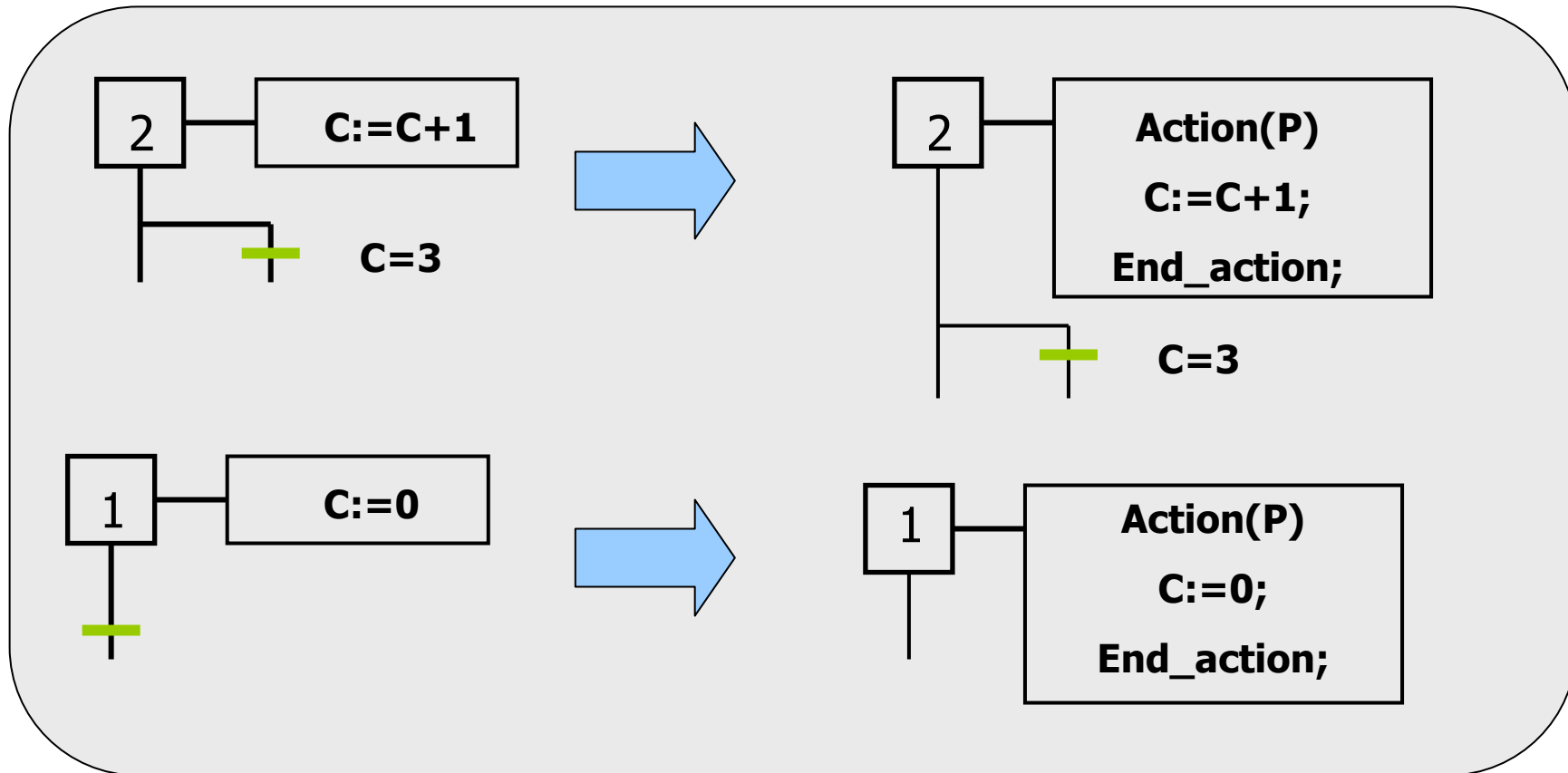
Exemple isagraf



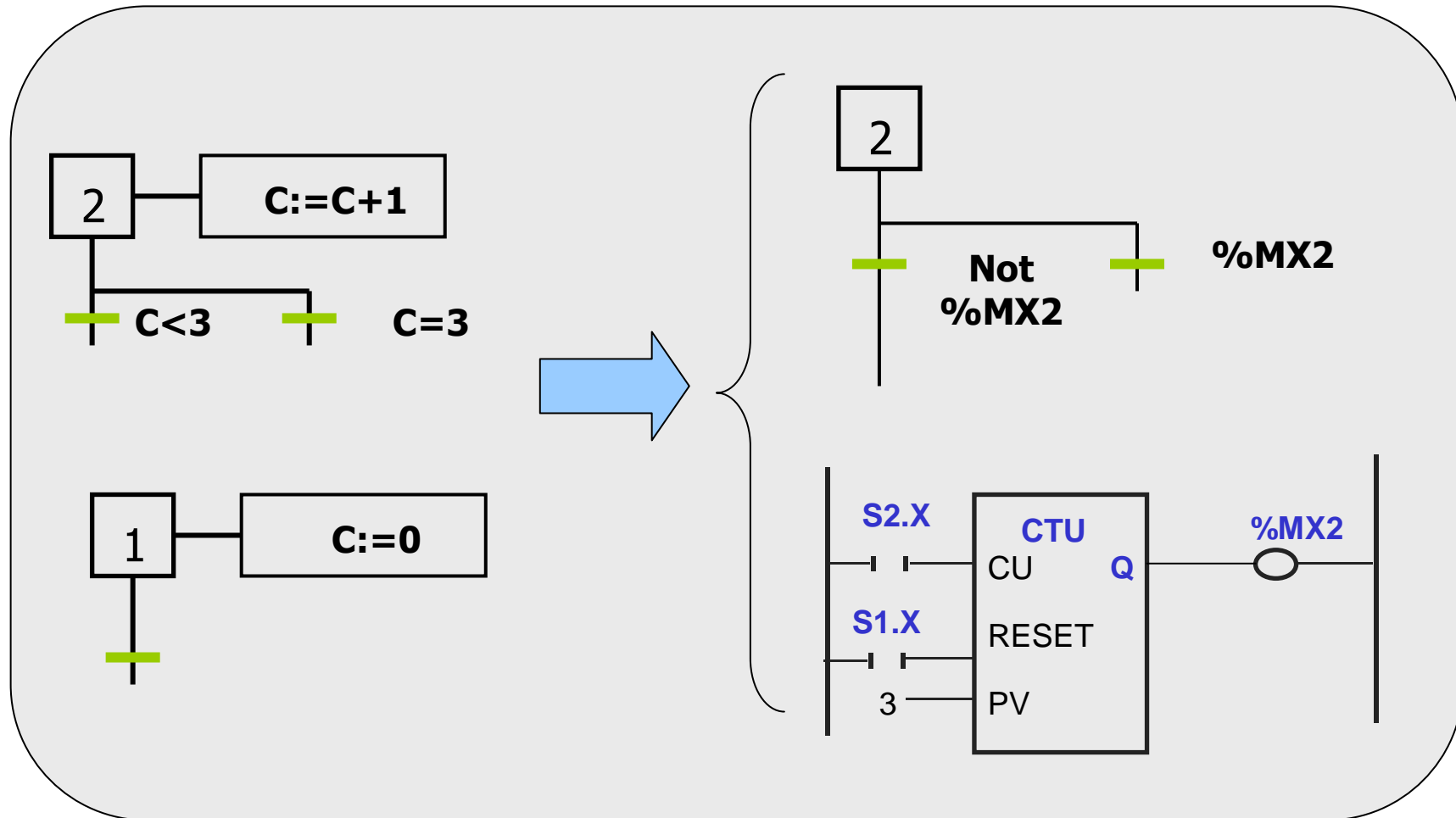
drapeau d'étape (ou variable d'étape) `GS*.x` pour isagraf ou `X*` pour pl7

Durée d'activation `GS*.t` pour isagraf et `X*.v` pour pl7

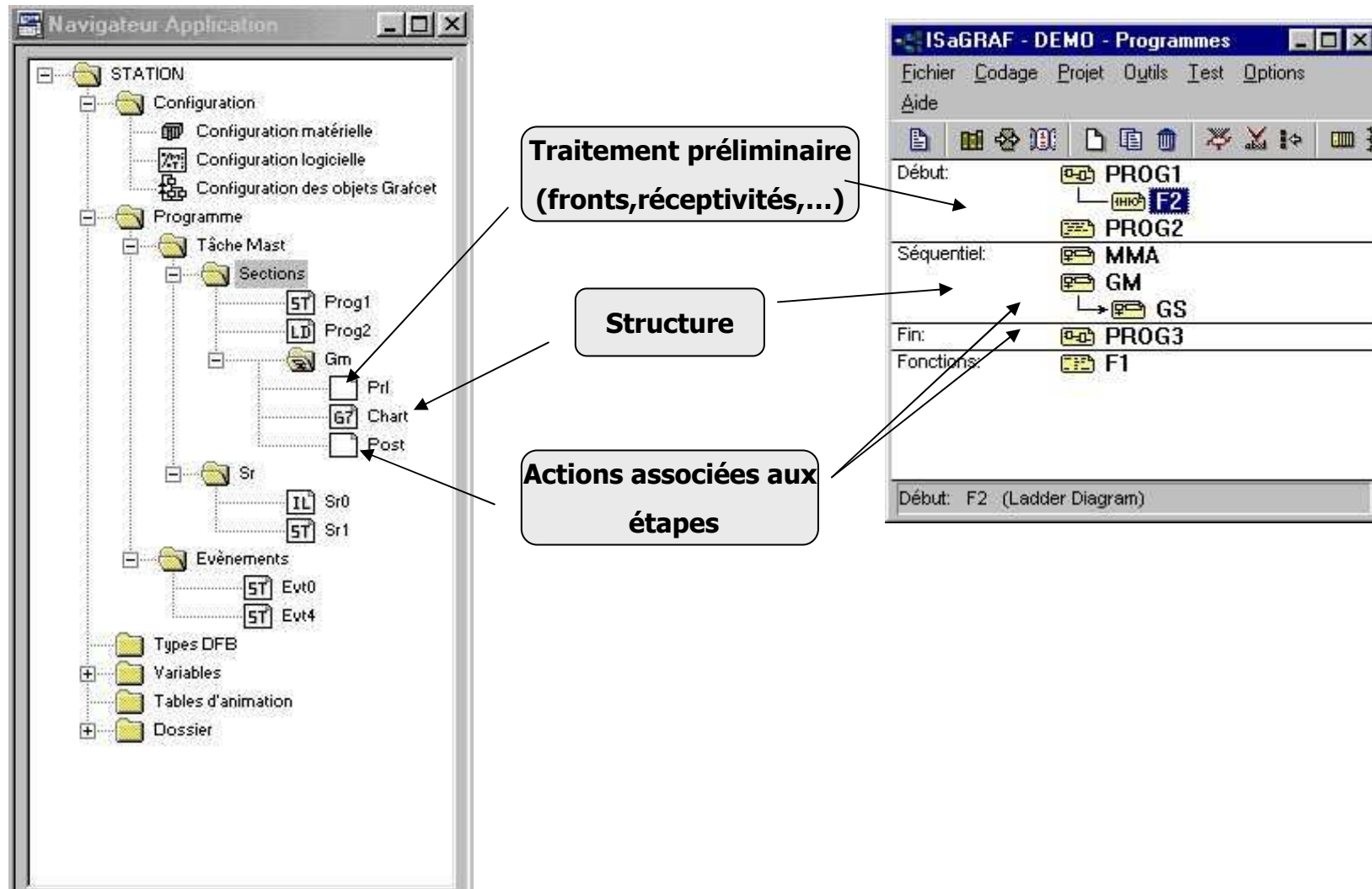
Réalisation des compteurs...



Réalisation des compteurs...



Structuration



Structuration Des Graphes

Si les Instructions de forçages ne sont pas définis dans la norme IEC 61131, les ateliers constructeurs offrent parfois des possibilités de hiérarchisation

Isagraf : Notion de père et de fils

Le père donne naissance au fils (GSTART)

Le père tue le fils (GKILL)

Le père gèle le fils (GFREEZE)

PL7 junior : Notion de bit système

Agit sur la totalité du chart !

%S21 pour G7 { INIT}

%S23 pour G7 {*}