# An algebraic approach for PLC programs verification

Jean-Marc Roussel, Jean-Marc Faure

# An algebraic approach for PLC programs verification

Jean-Marc Roussel, Jean-Marc Faure
LURPA-ENS de Cachan
61, Avenue du Pt Wilson
F-94235 Cachan Cedex FRANCE
Jean-Marc.Roussel@lurpa.ens-cachan.fr, Jean-Marc.Faure@lurpa.ens-cachan.fr

## Abstract

*This article presents a verification based on a specific Boolean algebra, called $\mathbb{I}$, and symbolic reasoning on equations defined in this algebra. The formal definition of this algebra enables to model binary signals that include variables states, events, as well as physical delays between events. The behavior of the generic function blocks of the IEC 61131 standard as well as of PLC programs using these function blocks can be described in this algebra. Properties proof on PLC programs is performed by demonstrating, from the program, the formulas that express in the $\mathbb{I}$ algebra the properties to be proved.*

## 1. Introduction

Programmable Logic Controllers (PLC) perform numerous control tasks in manufacturing systems, transport systems, power systems. In order to ensure safety of these systems, PLC programs formal verification is therefore a major industrial concern. Program verification does not aim simply at checking the intrinsic properties of the program, e.g. no infinite loop, no locking point, ... , regardless of the application, but also at checking that the program behaviour complies with the application requirements. In this article we will mainly focus on this last kind of properties: the compliance of a given program with the properties required for the application.

A lot of methods have been developed to formally verify PLC programs written in IEC 61131-3 languages [5]. They have often used or adapted to Control Engineering methods issued from Computer Science such as model-checking [1], translation into synchronous languages [6]. A good survey as well as a relevant classification can be found in [3].

Our laboratory has contributed to this issue by achieving several works in model-checking since ten years. The first works used a specific model-checking tool [10], developed for properties checking on Sequential Function Charts (SFC); the last ones [9], [7] take benefit of the SMV symbolic model-checker [8].

The results of these works are of interest because they have enabled to formally verify properties of industrial PLC programs written in several languages of the IEC 61131-3 standard. Nevertheless these researches have pointed out clearly that in some cases the model-checker is unable to provide a solution because of combinatory explosion. This drawback of model-checking has led us to undertake works aiming at providing an other complementary verification method.

To tackle the combinatory explosion problem implies to consider the underlying theory of model-checking tools. All these software are developed from DES theory and therefore consider a program as a state automaton. Even if symbolic model-checking is employed, the size of this automaton can be so huge that combinatory explosion occurs when dealing with some industrial programs. We have consequently searched for a verification method based on a more compact representation and have chosen an algebraic representation. With this approach the program shall be represented by a set of equations and verification shall be performed by symbolic reasoning on this set. The properties to be proved (the application requirements) shall therefore be also represented in the same algebraic form. Once the algebraic approach chosen, a problem arises: which algebra is to be used ? As the purpose is to represent the variables and the instructions of PLC standard languages, such as edge detectors, timer function blocks, Boolean memories, an algebra only dealing with states of Boolean variables is not suitable. We need to represent states, events and physical delays with the same formalism. It is the reason why we have decided to develop a new algebra providing this possibility. This algebra has been called $\mathbb{I}$ because its aim is to represent at one and the same time, states, events and delays; it is therefore an Integrating framework.

This article is structured in the following way. The first section gives an overview of the verification method. Then we present the elements of the $\mathbb{I}$ algebra as well as the way

in which we express the behavior of basic function blocks of PLC standard languages into this algebra. This enables to establish generic properties of these function blocks useful when demonstrating the required properties. An example of formal verification of a safety-related program is given in the last part.

## 2 Verification method overview

PLC programs are developed by control engineers which use their skills and their experience to elaborate these programs from the requirements, with or without a development method specific to the application field considered or imposed by the customer or by the system supplier. The verification method shall be independent of the chosen development method. On the other hand the languages of the IEC 61131-3 standard are widely used for PLC programming and we will only consider programs written in these languages.

The first step of the verification method (Figure 1) provides a formal representation in $\mathbb{I}$ of the program behaviour. In the same way, properties required for the application have to be formalised with algebraic formulas. The last step is merely symbolic reasoning on the first set of formulas (those obtained from the program) in order to obtain the formulas expressing the required properties.

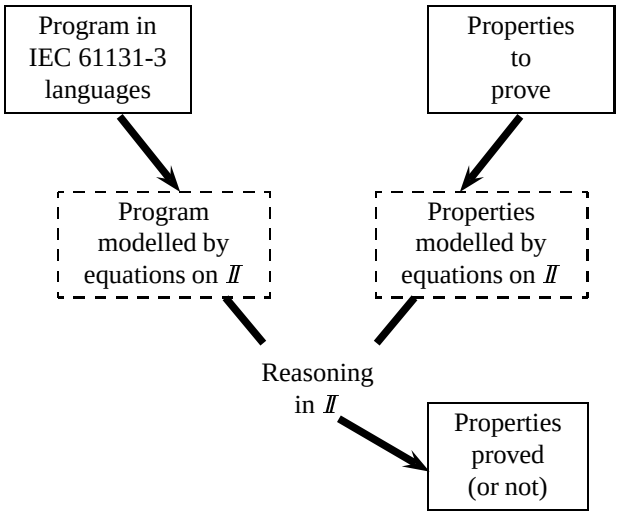



**Figure 1. The differents steps of the verification method**

## 3 A Boolean algebra for binary signals

### 3.1 Binary signals modelling

As mentioned in the introduction, the $\mathbb{I}$ algebra shall provide a formal framework to represent and manipulate Boolean variables states, Boolean events and physical delays between events. The main idea for the definition of this algebra has been to consider **binary signals**, i.e. variables describing the evolution during time of Boolean values.

These evolutions are usually represented by timing diagrams. This representation is quite useful for control engineers but is not at all based on a sound formalism. In order to provide a formal framework for binary signals, we propose to represent them as piecewise-continuous functions from $\mathbb{R}^{+*}$ to $\mathbb{B} = \{0, 1\}$. The elements of $\mathbb{I}$ are consequently formally defined in the following way :

$$\mathbb{I} = \left\{ \begin{array}{l} f : \mathbb{R}^{+*} \to \mathbb{B} \quad | \\ \forall t \in \mathbb{R}^{+*} : (\exists \epsilon_t > 0 : (\forall (\epsilon_1, \epsilon_2) \in (0, \epsilon_t)^2, \\ \qquad\qquad f(t - \epsilon_1) = f(t - \epsilon_2))) \end{array} \right\}$$

The figure 2 shows an example of a function element of $\mathbb{I}$. Attention shall be paid to the right-continuity used for the edges (at the dates $t_1$ and $t_3$) and to the double-discontinuity (for the dates $t_2$ and $t_4$), mandatory to model events. A more detailed presentation is given in [11].

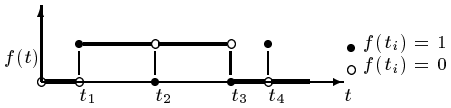



**Figure 2. Example of function element of $\mathbb{I}$**

To distinguish the operations on the elements of $\mathbb{I}$ from the operations on the booleans, different notations are used:

- $f$, $g$, $h$ refer to elements of $\mathbb{I}$,
- $f(t)$, $g(t)$, $h(t)$ refer to booleans, values of $f$, $g$, $h$ at a given instant $t$,
- "$\wedge$", "$\vee$", "$\neg$" mean respectively logical AND, OR, NOT,
- "$\cdot$", "$+$", "$^-$" are used for operations of $\mathbb{I}$.

$\mathbb{I}$ contains two special elements $1^*$ (the one element) and $0^*$ (the zero element) defined as follows:

$$\begin{array}{rcl} 1^* : \mathbb{R}^{+*} & \to & \mathbb{B} \\ 1^*(t) & \mapsto & 1 \end{array} \qquad \begin{array}{rcl} 0^* : \mathbb{R}^{+*} & \to & \mathbb{B} \\ 0^*(t) & \mapsto & 0 \end{array}$$

### 3.2 Structure of Boolean Algebra

To compose the elements of $\mathbb{I}$, three closed operations have been defined:

The AND operation

$$\begin{array}{ccc} \mathbb{I}^2 & \to & \mathbb{I} \\ (f,g) & \mapsto & (f.g) \end{array}$$

The OR operation

$$\begin{array}{ccc} \mathbb{I}^2 & \to & \mathbb{I} \\ (f,g) & \mapsto & (f+g) \end{array}$$

The NOT operation

$$\begin{array}{ccc} \mathbb{I} & \to & \mathbb{I} \\ (f,g) & \mapsto & \overline{f} \end{array}$$

with $\forall t \in \mathbb{R}^{+*}$,
$(f.g)(t) = f(t) \wedge g(t)$
$(f+g)(t) = f(t) \vee g(t)$
$\overline{f}(t) = \neg f(t)$

$(\mathbb{I}, .\,, +, \bar{}, 1^*, 0^*)$ is a Boolean algebra [4] because the following conditions are satisfied for all $f, g, h \in \mathbb{I}$:

| | |
|---|---|
| $f.g = g.f \quad f+g = g+f$ | Commutative Laws |
| $f.(g+h) = (f.g)+(f.h)$<br>$f+(g.h) = (f+g).(f+h)$ | Distributive Laws |
| $f.1^* = f \quad f+0^* = f$ | Identity Laws |
| $f.\overline{f} = 0^* \quad f+\overline{f} = 1^*$ | Inverse Laws |
| $0^* \neq 1^*$ | |

As $(\mathbb{I}, .\,, +, \bar{}, 1^*, 0^*)$ is a Boolean algebra, the properties hereafter are satisfied [4]:

| | |
|---|---|
| $f.f = f \quad f+f = f$ | Idempotent Laws |
| $f.0^* = 0^* \quad f+1^* = 1^*$ | Dominance Laws |
| $f.(f+g) = f$<br>$f+(f.g) = f$ | Absorption Laws |
| $\begin{cases} f.g = f.h \\ \overline{f}.g = \overline{f}.h \end{cases} \Rightarrow g = h$ | Cancellation Laws |
| $\begin{cases} f+g = f+h \\ \overline{f}+g = \overline{f}+h \end{cases} \Rightarrow g = h$ | |
| $f.(g.h) = (f.g).h$<br>$f+(g+h) = (f+g)+h$ | Associative Laws |
| $\overline{\overline{f}} = f$ | Law of the Double Complement |
| $\overline{f.g} = \overline{f}+\overline{g} \quad \overline{f+g} = \overline{f}.\overline{g}$ | DeMorgan's Laws |

A partial order between elements of $\mathbb{I}$ can be introduced by the subset relation "implication". This relation is defined as follows:

$$f \overset{\mathbb{I}}{\Rightarrow} g \quad \text{if and only if, } \forall t \in \mathbb{R}^{+*} : \quad f(t) \overset{\mathbb{B}}{\Rightarrow} g(t)$$

where $\overset{\mathbb{B}}{\Rightarrow}$ is the implication operation on $\mathbb{B}$.
It really matters to highlight the usefulness of this relation for properties checking. This will be illustrated in section 5. Furthermore, for all $f, g \in \mathbb{I}$, the six following relations are equivalent:

$$f \overset{\mathbb{I}}{\Rightarrow} g \quad \overline{f}+g = 1^* \quad f.\overline{g} = 0^*$$
$$\overline{g} \overset{\mathbb{I}}{\Rightarrow} \overline{f} \quad f.g = f \quad f+g = g$$

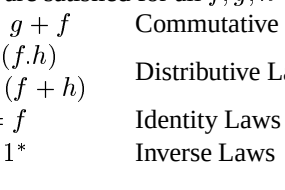
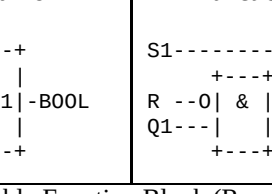
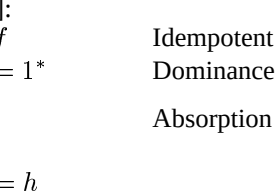
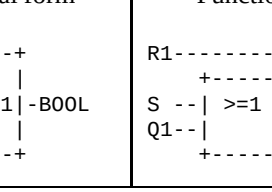
This algebra must be distinguished from process algebra that are aimed to formally represent state automata. In our case, the underlying model of the algebra is not a kind of state automaton, but the binary signal, piecewise-continuous function of time.

## 4 Function blocks behavior and properties

Once the algebra defined, it is possible to obtain a formal description of all the boolean function blocks of the IEC 61131-3 standard. This part focuses only on boolean memories, timers and edge detectors.

### 4.1 Memory operations

The bistable function blocks are defined in the standard as follows:

| Bistable Function Block (Set dominant) | |
|---|---|
| Graphical form | Function Block body |
|  |  |
| **Bistable Function Block (Reset dominant)** | |
| Graphical form | Function Block body |
|  |  |

Two operations on $\mathbb{I}$ have been defined for giving an algebraic semantic to bistable function blocks:

The $\mathsf{SR}$ operation

$$\begin{array}{ccc} \mathbb{I}^2 & \to & \mathbb{I} \\ (s,r) & \mapsto & \mathsf{SR}(s,r) \end{array}$$

The $\mathsf{RS}$ operation

$$\begin{array}{ccc} \mathbb{I}^2 & \to & \mathbb{I} \\ (s,r) & \mapsto & \mathsf{RS}(s,r) \end{array}$$

with $\forall t \in \mathbb{R}^{+*}$,

$$\mathsf{SR}(s,r)(t) = s(t) \vee (\exists t_1 < t \mid (s(t_1) = 1) \wedge (\forall t_2 \in (t_1, t],\ r(t_2) = 0))$$

$$\mathsf{RS}(s,r)(t) = (s(t) \wedge \overline{r}(t)) \vee (\exists t_1 < t \mid (s(t_1) = 1) \wedge (\forall t_2 \in [t_1, t],\ r(t_2) = 0))$$

Figure 3 depicts two binary signals $s$, $r$ inputs of two $\mathsf{SR}$ and $\mathsf{RS}$ function blocks and the corresponding outputs $\mathsf{SR}(s,r)$, $\mathsf{RS}(s,r)$.

With these definitions, the following theorems have been estabished:

$$s \overset{\mathbb{I}}{\Rightarrow} \mathsf{SR}(s,r) \quad \mathsf{SR}(s,\overline{s}) = s \quad \mathsf{SR}(s,r).r = s.r$$
$$r \overset{\mathbb{I}}{\Rightarrow} \overline{\mathsf{RS}(s,r)} \quad \mathsf{RS}(s,\overline{s}) = s \quad \mathsf{RS}(s,r).s = s.\overline{r}$$
$$\mathsf{SR}(s,1^*) = s \quad \mathsf{SR}(1^*,r) = 1^* \quad \mathsf{SR}(0^*,r) = 0^*$$
$$\mathsf{RS}(s,1^*) = 0^* \quad \mathsf{RS}(1^*,r) = \overline{r} \quad \mathsf{RS}(0^*,r) = 0^*$$
$$\mathsf{RS}(s,s) = 0^*$$
$$\mathsf{RS}(s,r) = \mathsf{RS}(s.\overline{r},r) \quad \mathsf{RS}(s+r.f,r) = \mathsf{RS}(s,r)$$
$$\mathsf{SR}(s,r_1+r_2) = \mathsf{SR}(s,r_1).\mathsf{SR}(s,r_2)$$
$$\mathsf{SR}(s_1+s_2,r) = \mathsf{SR}(s_1,r)+\mathsf{SR}(s_2,r)$$
$$\mathsf{RS}(s,r_1+r_2) = \mathsf{RS}(s,r_1).\mathsf{RS}(s,r_2)$$
$$\mathsf{RS}(s_1+s_2,r) = \mathsf{RS}(s_1,r)+\mathsf{RS}(s_2,r)$$

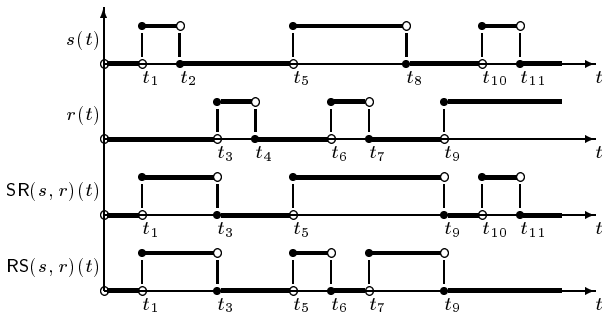



**Figure 3. Examples of functions $s$, $r$, $\mathsf{SR}(s,r)$ and $\mathsf{RS}(s,r)$**

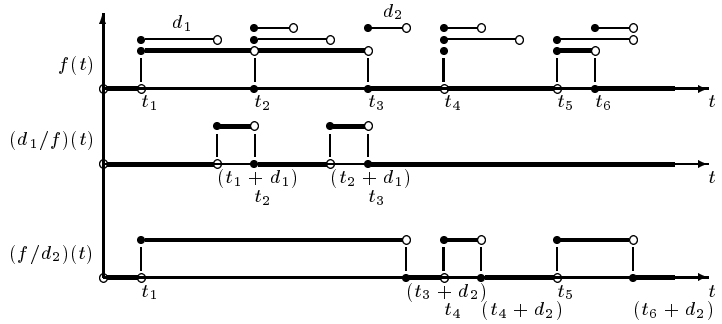
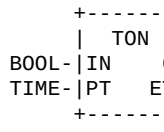
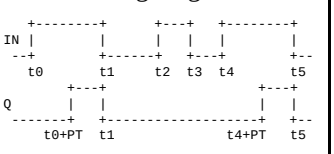
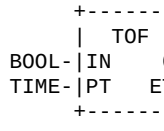
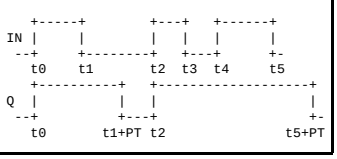



**Figure 4. TON and TOF operations for a function $f$ of $\mathbb{I}$**

### 4.2 Timing operations

The timer function blocks are defined in the standard as follows:

| ON-delay Timing (TON) | |
|---|---|
| Graphical form | Timing diagram |
|  |  |
| **OFf-delay Timing (TOF)** | |
| Graphical form | Timing diagram |
|  |  |



The algebraic semantics of these function blocks is the following:

| The TON operation | | The TOF operation | |
|---|---|---|---|
| $\mathbb{I}$ | $\rightarrow$ $\mathbb{I}$ | $\mathbb{I}$ | $\rightarrow$ $\mathbb{I}$ |
| $f$ | $\mapsto$ $d/f$ | $f$ | $\mapsto$ $f/d$ |

with $\forall t \in \mathbb{R}^{+*}$,

$$(d/f)(t) = \begin{cases} 0 & \forall t < d \\ (\forall t_1 \in (t-d, t], f(t_1) = 1) & \forall t \geq d \end{cases}$$

$$(f/d)(t) = \begin{cases} (\exists t_1 \in (0, t], f(t_1) = 1) & \forall t < d \\ (\exists t_1 \in (t-d_2, t], f(t_1) = 1) & \forall t \geq d \end{cases}$$

Figure 4 depicts a binary signal $f$, input of the TON and TOF function blocks and the corresponding outputs $d_1/f$, $f/d_2$.

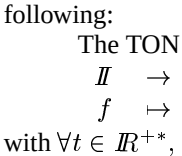
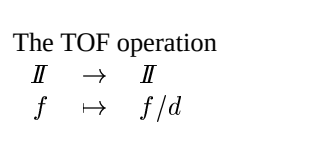
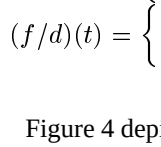
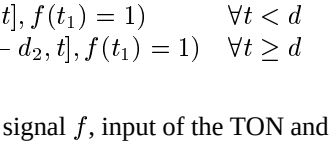
This definition enables to establish the following theorems:

$d/f \overset{\mathbb{I}}{\Rightarrow} f \quad f \overset{\mathbb{I}}{\Rightarrow} f/d$

$d/(f.g) = d/f \ . \ d/g$
$(f+g)/d = f/d \ + \ g/d$
$(d_1/f) \ . \ (d_2/f) = max(d_1, d_2)/f$
$(f/d_1) + (f/d_2) = f/max(d_1, d_2)$
$(d_1/f) + (d_2/f) = min(d_1, d_2)/f$
$(f/d_1) \ . \ (f/d_2) = f/min(d_1, d_2)$
$d_1/(d_2/f) = sum(d_1, d_2)/f$
$(f/d_1)/d_2 = f/sum(d_1, d_2)$
$\forall t \geq d, \quad \overline{d/f} = \overline{f}/d$
$\forall t \geq d, \quad \overline{f/d} = d/\overline{f}$

### 4.3 Edge operations

The edge detection function blocks are defined in the standard as follows:

| Rising edge detector | |
|---|---|
| Graphical form | Function Block body |
|  |  |
| **Falling edge detector** | |
| Graphical form | Function Block body |
|  |  |



As previously, two operations on $\mathbb{I}$ are defined:

| The Rising Edge operation | | The Falling Edge operation | |
|---|---|---|---|
| $\mathbb{I}$ | $\rightarrow$ $\mathbb{I}$ | $\mathbb{I}$ | $\rightarrow$ $\mathbb{I}$ |
| $f$ | $\mapsto$ $\uparrow f$ | $f$ | $\mapsto$ $\downarrow f$ |

with $\forall t \in \mathbb{R}^{+*}$,

$$\uparrow f(t) = \underline{f}(t) \wedge (\exists \epsilon_0 > 0 \mid \forall \epsilon \in \ (0, \epsilon_0)\ f(t-\epsilon) = 0)$$
$$\downarrow f(t) = \overline{f}(t) \wedge (\exists \epsilon_0 > 0 \mid \forall \epsilon \in \ (0, \epsilon_0)\ f(t-\epsilon) = 1)$$

Figure 5 depicts a binary signal $f$, input of the Rising Edge and Falling Edge function blocks and the corresponding outputs $\uparrow f, \downarrow f$.

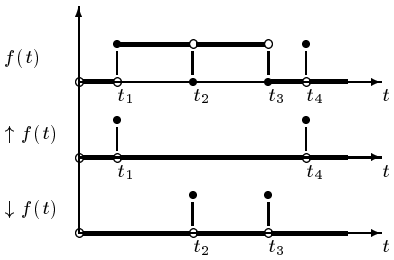



**Figure 5. Rising Edge and Falling Edge operations for a function $f$ of $I\!I$**

These formal definitions of edge detectors enable to write formulas including variables states and events as shown in the next section. Moreover, with this algebraic definition, the following theorems have been estabished:

$$\uparrow f \overset{I\!I}{\Rightarrow} f \quad \uparrow \overline{f} = \downarrow f \quad \downarrow f \overset{I\!I}{\Rightarrow} \overline{f} \quad \downarrow \overline{f} = \uparrow f$$
$$\uparrow (f.g) = \uparrow f.g + f. \uparrow g \quad \downarrow (f+g) = \downarrow f.\overline{g} + \overline{f}. \downarrow g$$
$$\uparrow (f+g) = \uparrow f. \uparrow g + \uparrow f.\overline{g}.\overline{\downarrow g} + \uparrow g.\overline{f}.\overline{\downarrow f}$$
$$\downarrow (f.g) = \downarrow f. \downarrow g + \downarrow f.g.\overline{\uparrow g} + \uparrow g.f.\overline{\uparrow f}$$

## 5 Example

The usefulness of the $I\!I$ algebra for properties checking will be demonstrated thanks to a simple safety-related program. The aim of this program is to monitor the safe operation of the two pushbuttons used to operate presses and similar dangerous machinery. It ensures that both hands of an operator are kept outside the danger zone during machine operation. Usually this safety-related function is realised by safety relays systems tested and approved by standards institutions. Nowadays this function is available in programmable safety systems. The behaviour of this function is standardised [2]. The main points are:

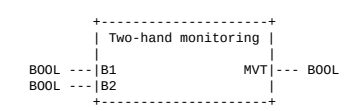
**P1** A cycle can only be initiated by pressing the two pushbuttons simultaneously (within 0.5 s).

**P2** A cycle is interrupted by releasing one or both buttons to stop the output.

**P3** The output signal can only be reinitiated after both inputs have been released and the pushbuttons are operated again.

To obtain this behaviour, the control program depicted in figure 6 has been designed. This program is written

Two-hand monitoring : external Interface




Two-hand monitoring : body

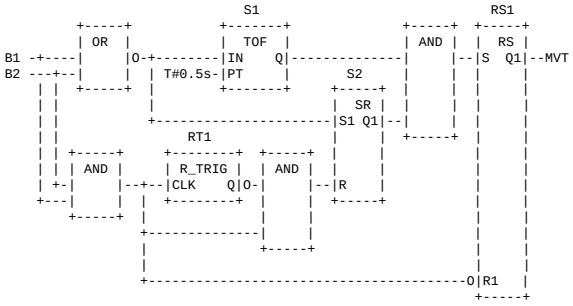



**Figure 6. Control program written in FBD language according to the IEC 61131-3 standard**

with the Function Block Diagram (FBD) PLC programming language, though it could be possible to give an equivalent program in Ladder Diagram (LD). Only standard functions: bitwise boolean functions (AND, OR), bistable function blocks (SR, RS) and edge detection function blocks (R_TRIG) have been used.

This program can be model in $I\!I$ as follows:

$$\begin{cases} MVT = \mathsf{RS}(S1.S2\ ,\ S3) & \text{with} \\ S_1 = \left(\overline{B_1 + B_2}\right) / 0,5s \\ S_2 = \mathsf{SR}\left(\overline{B_1 + B_2}\ ,\ B_1.B_2.\overline{\uparrow (B_1.B_2)}\right) \\ S_3 = \overline{B_1.B_2} \end{cases}$$

The properties P1 and P2 can be easily proved from this formal definition of the program. These properties shall be written on $I\!I$ as follows:

P1 To set "$MVT$", it is necessary to have the two pushbuttons pressed and not to have one or both buttons pressed from 0,5s.
$\uparrow MVT \overset{I\!I}{\Rightarrow} B_1\ .\ B_2\ .\ \overline{0,5s/(B_1 + B_2)}$

P2 If a pushbutton is released, the output "MVT" is reset.
$\overline{B_1} + \overline{B_2} \overset{I\!I}{\Rightarrow} \overline{MVT}$

The property P2 is merely proved as follows:

| | *Reasons* |
|---|---|
| $\overline{B_1} + \overline{B_2} = \overline{B_1.B_2} = S3$ | DeMorgan's Laws |
| $S3 \overset{I\!I}{\Rightarrow} \overline{MVT}$ | Using: $r \overset{I\!I}{\Rightarrow} \overline{\mathsf{RS}(s,r)}$ |
| $\overline{B_1} + \overline{B_2} \overset{I\!I}{\Rightarrow} \overline{MVT}$ | Consequently |

The property P1 is proved as follows:

$\uparrow MVT = \uparrow (\mathsf{RS}(S_1.S_2\ ,\ S_3))$

By definition of $MVT$

$\uparrow (\mathsf{RS}(S_1.S_2\ ,\ S_3)) \overset{\mathbb{I\!I}}{\Rightarrow} \uparrow ((S_1.S_2).\overline{S_3})$

Using: $\uparrow (\mathsf{RS}(s,r)) \overset{\mathbb{I\!I}}{\Rightarrow} \uparrow (s.\overline{r})$

(Property not yet presented)

$\uparrow (S_1.S_2.\overline{S3}) \overset{\mathbb{I\!I}}{\Rightarrow} S_1.S_2.\overline{S_3}$

Using: $\uparrow f \overset{\mathbb{I\!I}}{\Rightarrow} f$

$S_1.S_2.\overline{S_3} \overset{\mathbb{I\!I}}{\Rightarrow} S_1.\overline{S_3}$

By definition of $\overset{\mathbb{I\!I}}{\Rightarrow}$

$\overline{S_3} = \overline{\overline{B_1.B_2}} = B_1.B_2$

Law of the Double Complement

$S_1 = \overline{(B_1 + B_2)}\ /0,5s = \overline{0,5s/(B_1 + B_2)}$

Using: $\overline{t_1/f} = \overline{f}/t_1$

$S_1\ .\ \overline{S_3} = B_1\ .\ B_2\ .\ \overline{0,5s/(B_1 + B_2)}$

Using precedent results

$\uparrow MVT \overset{\mathbb{I\!I}}{\Rightarrow} B_1\ .\ B_2\ .\ \overline{0,5s/(B_1 + B_2)}$

Consequently

The P3 property involves states of the same variables at different dates (for instance both inputs shall be at the false level at a given date t and at the true level at another date t', greater than t) and therefore is not so easy to prove than the two first ones. This property can be written in CTL temporal logic as follows:

$$AG\,(mvt \Rightarrow \neg(E[\neg(b_1 \wedge b_2)]U[\neg mvt \wedge EX(mvt)]))$$

To verify that kind of property, we are currently developing new operations on $\mathbb{I\!I}$ that enable to analyse the past of binary signals.

## 6 Conclusion and perspectives

The $\mathbb{I\!I}$ algebra provides a formal framework to represent Boolean variables states, events and physical delays and has permitted to develop the verification method presented in this article. This method has been tested in several cases with success. It is particularly well-suited for structured programs as industrial ones. The example described in this article is written in FBD; the same equations and reasoning would be obtained with a program in Ladder Diagram. Moreover the function blocks presented are defined for all the IEC 61113-3 languages (e.g. SFC); the results obtained may be therefore applied to any program developed in these languages.

To help the designer when properties checking, we have developed during the last year a solver under Mathematica®. This software relies on the basic properties of this boolean algebra as well as on the theorems related to function blocks and is able to simplify expressions on $\mathbb{I\!I}$. The designer used this tool to realize symbolic calculus on $\mathbb{I\!I}$. For our example, the properties P1 and P2 have been demonstrated automatically thanks to this solver.

The perspectives of these works are both formal and methodological. As mentioned at the end of the previous section, new operations increasing the potentiality of checking in $\mathbb{I\!I}$ are under development. From a methodological point of view, we have to consider the cooperation between the two verification methods nowadays used in our laboratory: model-checking and symbolic reasoning in $\mathbb{I\!I}$. Rational and complementary use of these two approaches will be of benefit for large size industrial PLC programs verification.

## References


[1] B. Bérard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, and P. Schnoebelen. *Systems and Software Verification: Model-Checking Techniques and Tools*. Springer-Verlag, Heidelberg, 2001. ISBN 3-540-41523-8.

[2] E. C. for Standardization. EN 574: Safety of machinery - two-hand control devices - functional aspects - principles for design, 1996.

[3] G. Frey and L. Litz. Formal methods in PLC programming. In *Proceedings of IEEE International Conference on Systems, Man, and Cybernetics*, Nasville, Tennessee, USA, october 2000.

[4] R. P. Grimaldi. *Discrete and Combinatorial Mathematics: An Applied Introduction*. Addison-Wesley, 4th edition, 1999. ISBN 0-201-19912-2.

[5] International Electrotechnical Commission. IEC 61131-3: Programmable controllers - programming languages, 1993.

[6] F. Jimenez-Fraustro and E. Rutten. A synchronous model of IEC 61131 PLC languages in signal. In *Proceedings of the 13th Euromicro Conference on Real-Time Systems (ERTS'01)*, pages 229–237, Delft, The Netherlands, June 2001.

[7] S. Lampérière-couffin and J.-J. Lesage. Formal verification of the sequential part of PLC programs. In *Proceedings of 5th Workshop on Discrete Event Systems (WODES)*, Ghent, Belgium, august 2000.

[8] K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, Hardbound, 1993. ISBN 0-7923-9380-5.

[9] O. Rossi, O. De Smet, S. Couffin, J.-J. Lesage, H. Papini, and H. Guennec. Formal verification: a tool to improve the safety of control systems. In *Proceeding of SafeProcess2000 - 4th symposium on Fault Detection, Safety and Supervision of Technical Processes*, pages 885–890, Budapest, Hungary, june 2000.

[10] J.-M. Roussel and J.-J. Lesage. Validation and verification of GRAFCET using state machine. In *Proceedings of IMACS-IEEE Multiconference on Computational Engineering in Systems Applications (CESA'96)*, pages 758–764, Lille, France, July 1996.

[11] C. Thierry, J.-M. Roussel, and J.-J. Lesage. An extended boolean algebra for the control of logical systems. In *Proceedings of 16th IMACS World Congress 2000 on Scientific Computation, Applied Mathematics and Simulation*, Lausanne, Switzerland, august 2000.