Algebraic synthesis of logical controllers with optimization criteria

Hélène Leroux, Jean-Marc Roussel

Algebraic synthesis of logical controllers with optimization criteria

Hélène Leroux
LIRMM
161 rue Ada
F-34 095 Montpellier cedex 5, FRANCE
www.lirmm.fr
leroux@lirmm.fr

Jean-Marc Roussel
LURPA, ENS Cachan
61, Avenue du Pt Wilson
F-94235 Cachan Cedex, FRANCE
www.lurpa.ens-cachan.fr
jean-marc.roussel@lurpa.ens-cachan.fr

Formal methods can strongly contribute to improve dependability of logic controllers during design, by providing means to avoid flaws due to designers' omissions or specifications misinterpretations. This communication presents a formal synthesis method whose goal is to obtain the control laws of a logic system from specifications given in natural language.
The formal framework underlying the method is the Boolean algebra of n-variable switching functions. The operations and relations of this algebra enable to represent controller specifications formally, to detect inconsistencies within specifications and to generate control laws with optimization criteria.
The application of this formal design method to an example permits to illustrate its main advantages.

Dependable system, Controller synthesis, Algebraic approaches, Boolean algebra, Optimization

1. INTRODUCTION

Programmable Logic Controllers (PLCs) are frequently used to control safety-critical systems. Since failure of such systems may have disastrous effects, the use of formal methods to the software of such systems is highly desirable (12).

Numerous formal methods have been proposed during the last two decades. Their goal is either to detect flaws once the controller is designed or to avoid flaws during design. The first approach consists in letting the control system designer develop control laws based on the requirements contained in the set of specifications. Then an automatic analysis is done on the formal representation of these control laws. Such an analysis may be carried out by using model-checking techniques (2) provided that control laws can be formally represented in the form of state automata (5), (6), (1). These specifications can also be verified by theorem-proving (19), (23).

The second approach, qualified as synthesis, aims at deducing the control laws from the specifications, without any involvement of a designer (or at least in limiting it to a strict minimum). To design logic controllers, several research teams propose to exploit the Supervisory Control Theory (SCT) defined by Ramadge and Wonham (18). This language theory provides a formal framework for Discrete Event Systems analysis and synthesis. The starting point of this method is two distinct automata called Plant and Specification. Synthesis algorithms automatically generate an optimal Supervisor. It operates synchronously with the plant to restrict the language accepted by the plant to satisfy the specification. Results obtained on complex case studies (for example, (17)) point out the real potential of this theory for a high-level description plant model.

However, when the plant model must be very detailed to take into account all the reaction of an operative system, the efforts made to obtain a coherent plant model are often more important than the efforts made to find a solution by hand. An example of a plant model with this abstraction level is presented in (20). To obtain from the Supervisor a straightforwardly implementable Controller, a model of the interactions between the Controller and the Process has to be included in the Plant model. Indeed it makes the extraction of the Controller behavior from the Supervisor easier (4). As the quality of the controller obtained with the SCT depends on the precision of the plant model, this approach is not really well-adapted to obtain a software implementable into a PLC.

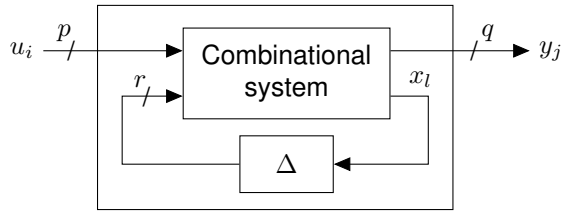
The proposed method was specially developed to get the control law to be implemented into these controllers. We have chosen to represent these control laws with recurrent Boolean equations in order to facilitate implementation aspects and the formalization of safety requirements. The formal framework underlying the method is the Boolean algebra of n-variable switching functions. With this choice, the formal description of the control laws obtained at the end of the design can be easily translated into a program developed in a standardized language for programmable logic controllers. The aim of this formal method is indeed to provide a seamless whole from specifications analysis to implementation.

This paper is organized as follows. Section 2 describes the objectives and the main principles of our method. Section 3 presents the formal framework used and the mathematical results on which this method is based. Our algebraic synthesis method with optimization criteria is finally developed in section 4 thanks to an illustrative example.

2. OBJECTIVE

The proposed method was specially developed to get the control law to be implemented into logical controllers from specifications given in natural language.

The generic model presented Fig. 1, has p Boolean inputs (u_i), q Boolean outputs (y_j) and r Boolean state variables (x_l). These inputs and outputs correspond to the inputs and outputs of the controller for which the control laws must be designed. The state variables, used to express the sequential behavior, will be represented with internal variables of the controller.



$$\begin{cases} y_j[k] = F_j(u_1[k], \cdots, u_p[k], x_1[k-1], \cdots, x_r[k-1]) \\ x_l[k] = F_{q+l}(u_1[k], \cdots, u_p[k], x_1[k-1], \cdots, x_r[k-1]) \end{cases}$$

Figure 1: *Generic model of sequential systems expressed with recurrent Boolean equations*

The behavior of this model can be fully defined according to the definition of $(q + r)$ switching functions of $(p + r)$ variables. This representation is very compact as the r Boolean state variables (x_l) of this model permit to represent as far as 2^r different states. The manual definition of these switching functions has always been a very tedious and error-prone task (11): the model, proposed Fig. 1, admits 2^p different inputs combinations, can send as far as 2^q different outputs combinations and can express $(2^{2^{(p+r)}})^{(q+r)}$ different sequential behaviors. That is why different methods based on state model representations have been rapidly defined (15), (16). Nevertheless, thanks to mathematical results obtained for Boolean algebras (22), (3), an automatic synthesis of these switching functions from their formal specifications is now possible.

To cope with combinatorial explosion, switching functions will be handled through a symbolic representation (and not their truth-tables which contains $2^{(p+r)}$ Boolean values). Each input u_i and output y_j of the controller will be represented by the a switching functions: U_i or Y_j. To take into account the recursive aspect of state variables, each state variable x_l will be represented by two switching functions: X_l (for time $[k]$) and ${}_pX_l$ (for time $[k-1]$). We propose to obtain switching functions Y_j and X_l by solving a Boolean equations system which represents the formal description of requirements given by the designer.

The input data of the proposed method (Fig 2) are informal functional and safety requirements given by the designer. In practice, these requirements are most often given in a textual form and/or by using technical Taylor-made languages (Gantt Diagrams, Function Blocks Diagrams, Grafcet...).

All the steps of our synthesis method are implemented into a prototype software tool developed in Python[1]. The first step is the formalization of requirements within an algebraic description. Requirements expressed with a state model can directly be translated into recurrent Boolean equations, thanks to the algorithm proposed by (14). Generic formalization of some requirements are also proposed for helping the designer to formalize “standard” requirements given in textual form (examples are given in Section 4.2). In case where the know-how of the designer enables him to build a priori the global form of the solution (or of a part of the whole solution) it is also possible to give fragments of solution as requirements (10).

The second step consists in analyzing all the requirements in order to control if they are consistent together. In our case, this step is realized by symbolic calculation on an algebraic formulation of them. If inconsistencies exist, the conditions to have them are given to the designer. It will help

[1]Case studies are available online: http://www.lurpa.ens-cachan.fr/isa/asc/case_studies.html

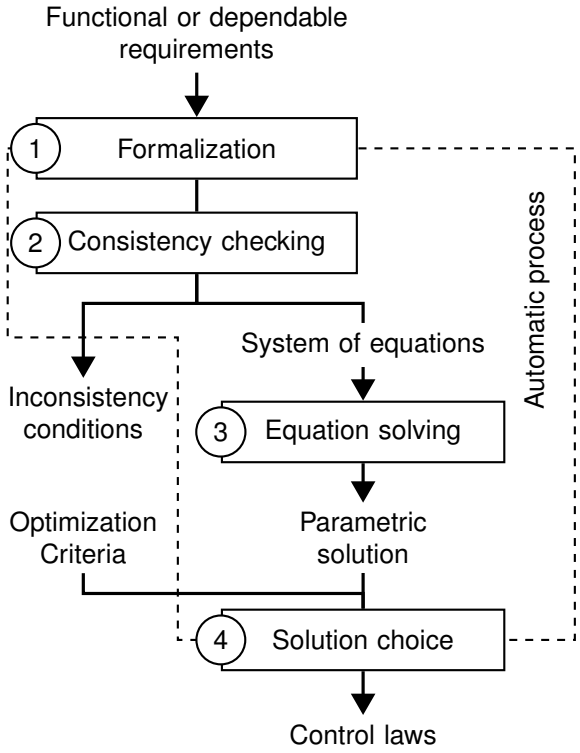


Figure 2: *Proposed method for designing logic controllers*

him modifying the given requirements. When the coherence is proved, the formal description of the requirements becomes a system of equations in which the searched control laws are the unknowns.

The third step is the synthesis of the control laws. This step consists in solving the system of equations representing the coherent requirements. The obtained results (Theorem 5) allows us to establish an exact symbolic form for solutions. The set of solutions is described thanks to a parametric expression of the solution.

The fourth step of the method is the choice of a solution by the designer. This step consists in fixing a specific value for each free parameter. The designer also has the possibility to complete the proposed requirements if the synthesized solutions are unsatisfactory. If our method can prove the coherence of the requirements, only the designer can judge whether the requirements are completed or not by analyzing the synthesized solutions.

To help the designer during this last step, we have obtained new mathematical results (Section 3.4) in order to be able to find automatically optimal solutions according to given criteria. The case study presented in Section 4 illustrates this fact.

In traditional design methods, the design procedure of a logic controller is an interactive process converging to an acceptable solution. At the beginning, requirements are not complete. New requirements can be introduced during the search of solutions. These complementary elements are given by the designer after an analysis of the partial solutions or the detection of inconsistencies. Designing a controller with an automatic synthesis technique will also be an interactive process on which the designer plays the leading part. Our method is based on this interactive process.

3. MATHEMATICAL FOUNDATIONS

This section is composed of four subsections. The first two regroup some classical results about Boolean algebras and the Boolean algebra of n-variable switching functions. Section 3.3 presents how to solve Boolean equations. Section 3.4 presents unpublished results.

3.1. Boolean Algebra: typical feature

Definition 1 (Boolean Algebra) [2] *Let $\mathcal{B}$ be a nonempty set that contains two special elements 0 (the zero element) and 1 (the unity, or one, element) and on which we define two closed binary operations $+$, $\cdot$, and an unary operation $\bar{}$. Then $(\mathcal{B}, +, \cdot, \bar{}, 0, 1)$ is called a Boolean algebra if the following conditions are satisfied for all $x, y, z \in \mathcal{B}$.*

$x + y = y + x$	$x \cdot y = y \cdot x$	*Commutative Laws*
$x \cdot (y + z) = (x \cdot y) + (x \cdot z)$		*Distributive Laws*
$x + (y \cdot z) = (x + y) \cdot (x + z)$		
$x + 0 = x$	$x \cdot 1 = x$	*Identity Laws*
$0 \neq 1$		

It exists many Boolean algebras. The most known is the two-element Boolean Algebra: $(\{0, 1\}, \vee, \wedge, \neg, 0, 1)$.

To avoid confusion between 0 as a Boolean value and 0 as the zero element of the Boolean algebra $\mathcal{B}$, Boolean values 0 and 1 will be noted respectively ${}_b0$ and ${}_b1$. The set of the two Boolean values ${}_b0$ and ${}_b1$ will be noted: $B = \{{}_b0, {}_b1\}$.

Definition 2 (Boolean Formula) [3] *A* Boolean formula *(or a Boolean expression) on $\mathcal{B}$ is any formula which represents a combination of members of $\mathcal{B}$ by the operations of $+$, $\cdot$, or $\bar{}$.*

By construction, any Boolean formula on $\mathcal{B}$ represents one and only one member of $\mathcal{B}$. Two Boolean formulae are equivalent if and only if they represent the same member of $\mathcal{B}$.
Later on, a Boolean formula $\mathcal{F}$ built with the members $(\alpha_1, \ldots, \alpha_n)$ of $\mathcal{B}$ is denoted $\mathcal{F}(\alpha_1, \ldots, \alpha_n)$.

[2] Definition 15.5 of (7)
[3] from Section 3.6 of (3)

Theorem 1 (Boole's expansion of a Boolean formula) *Let $(\alpha_1, \ldots, \alpha_n)$ be n members of $\mathcal{B} \setminus \{0,1\}$. Any Boolean Formula $\mathcal{F}(\alpha_1, \ldots, \alpha_n)$ can be expanded as:*

$$\mathcal{F}(\alpha_1, \ldots, \alpha_n) = \mathcal{F}_0(\alpha_2, \ldots, \alpha_n) \cdot \overline{\alpha_1} + \mathcal{F}_1(\alpha_2, \ldots, \alpha_n) \cdot \alpha_1$$

where $\mathcal{F}_0(\alpha_2, \ldots, \alpha_n)$ and $\mathcal{F}_1(\alpha_2, \ldots, \alpha_n)$ are Boolean formulae of only $\alpha_2, \ldots, \alpha_n$. These two formulae can be directly obtained from $\mathcal{F}(\alpha_1, \ldots, \alpha_n)$ as follows:

$$\begin{cases} \mathcal{F}_0(\alpha_2, \ldots, \alpha_n) = \mathcal{F}(\alpha_1, \ldots, \alpha_n)|_{\alpha_1 \leftarrow 0} \\ \qquad\qquad\qquad = \mathcal{F}(0, \alpha_2, \ldots, \alpha_n) \\ \mathcal{F}_1(\alpha_2, \ldots, \alpha_n) = \mathcal{F}(\alpha_1, \ldots, \alpha_n)|_{\alpha_1 \leftarrow 1} \\ \qquad\qquad\qquad = \mathcal{F}(1, \alpha_2, \ldots, \alpha_n) \end{cases}$$

This theorem is an adaptation for Boolean formulae of the Boole's expansion theorem initially developed for Boolean functions. In order to prove it[4], it is sufficient to verify its truthfulness for the elementary Boolean formulae (0, 1, α_1 and $\alpha_{j\neq 1}$) and to verify its conservation by composition with operations $+$, $\cdot$, and $\bar{}$.

The relation *Equality* is not the only defined relation on a Boolean Algebra. It is also possible to define a partial order relation between members of $\mathcal{B}$. This relation is called *Inclusion-Relation* in (3).

Definition 3 (Inclusion-Relation) [5]
If $x, y \in \mathcal{B}$, define $x \leq y$ if and only if $x \cdot y = x$.

Since in any Boolean algebra, $x \cdot y = x \Leftrightarrow x \cdot \overline{y} = 0$, we also have $x \leq y \Leftrightarrow x \cdot \overline{y} = 0$.

As Relation Inclusion is reflexive ($x \leq x$), antisymmetric (if $x \leq y$ and $y \leq x$, then $x = y$) and transitive (if $x \leq y$ and $y \leq z$, then $x \leq z$), this relation defines a partial order between members of $\mathcal{B}$ (Theorem 15.4 of (7)). It is only a partial order as some elements are not comparable.

3.2. The Boolean algebra of n-variable switching functions

To avoid confusion between Boolean variables and Boolean functions of Boolean variables, each Boolean variable b_i will be noted ${}_b b_i$.

Definition 4 (N-variable switching functions) [6]
An n-variable switching function is a mapping of the form

$$f : \quad B^n \to B \qquad \textit{where } B = \{{}_b 0, {}_b 1\}$$
$$({}_b b_1, \ldots, {}_b b_n) \mapsto f({}_b b_1, \ldots, {}_b b_n)$$

[4] A complete demonstration could be found in (9).
[5] Definition 15.6 of (7).
[6] From Section 3.11 of (3).

The domain of a n-variable switching function has 2^n elements and the co-domain has 2 elements; hence, there are 2^{2^n} n-variable switching functions. Let $F_n(B)$ be the set of the n-variable switching functions.

$F_n(B)$ contains $(n+2)$ specific n-variable switching functions: the 2 *constant functions* (0, 1), and the n *projection-functions* ($\mathsf{f}^i_{\mathsf{Proj}}$). These functions are defined as follows:

$$0 : \quad B^n \to B \qquad 1 : \quad B^n \to B$$
$$({}_b b_1, \ldots, {}_b b_n) \mapsto {}_b 0 \qquad ({}_b b_1, \ldots, {}_b b_n) \mapsto {}_b 1$$

$$\mathsf{f}^i_{\mathsf{Proj}} : \quad B^n \to B$$
$$({}_b b_1, \ldots, {}_b b_n) \mapsto {}_b b_i$$

3.2.1. Structure of Boolean Algebra

$F_n(B)$ can be equipped with three closed operations (two binary and one unary operations):

$$\text{Op. } + : \quad F_n(B)^2 \to F_n(B) \qquad \text{Op. } \cdot : \quad F_n(B)^2 \to F_n(B)$$
$$(f, g) \mapsto f + g \qquad (f, g) \mapsto f \cdot g$$

$$\text{Op. } \bar{} : \quad F_n(B) \to F_n(B)$$
$$f \mapsto \overline{f}$$

where $\forall ({}_b b_1, \ldots, {}_b b_n) \in B^n$,

$$(f+g)({}_b b_1, \ldots, {}_b b_n) = f({}_b b_1, \ldots, {}_b b_n) \vee g({}_b b_1, \ldots, {}_b b_n)$$
$$(f \cdot g)({}_b b_1, \ldots, {}_b b_n) = f({}_b b_1, \ldots, {}_b b_n) \wedge g({}_b b_1, \ldots, {}_b b_n)$$
$$\overline{f}({}_b b_1, \ldots, {}_b b_n) = \neg f({}_b b_1, \ldots, {}_b b_n)$$

$(F_n(B), +, \cdot, \bar{}, 0, 1)$ is a Boolean Algebra. Each of its member can be represented as a composition of $(\mathsf{f}^1_{\mathsf{Proj}}, \ldots, \mathsf{f}^n_{\mathsf{Proj}}, 0, 1)$ by operations $+$, $\cdot$ and $\bar{}$.

3.2.2. Boolean formula and relations on $F_n(B)$

As $(F_n(B), +, \cdot, \bar{}, 0, 1)$ is a Boolean Algebra, it is possible to write a Boolean formula of n-variable switching functions and relations between Boolean formula of n-variable switching functions.
As a n-variable switching function can always be defined by its truth-table, relations Equality and Inclusion can also be presented as follows:

- f and g are *equal* ($f = g$) iff the columns for f, g [in their respective truth-table] are exactly the same, i.e., $\forall ({}_b b_1, \ldots, {}_b b_n) \in B^n, f({}_b b_1, \ldots, {}_b b_n) = g({}_b b_1, \ldots, {}_b b_n)$.
- f is included into g ($f \leq g$) iff the value of g is always ${}_b 1$ when the value of f is ${}_b 1$, i.e., $\forall ({}_b b_1, \ldots, {}_b b_n) \in B^n, [f({}_b b_1, \ldots, {}_b b_n) = {}_b 0]$ or $[g({}_b b_1, \ldots, {}_b b_n) = {}_b 1]$.

3.3. Solutions of Boolean equations over Boolean algebra $F_n(B)$

In (3), F. M. Brown explains that many problems in the application of Boolean algebra may be reduced to solving an equation of the form

$$f(X) = 0$$

over a Boolean algebra $\mathcal{B}$. Formal procedures for producing solution of this equation were developed by Boole himself as a way to treat problems of logical inference. Boolean equations have been studied extensively since Boole's initial work (a bibliography of nearly 400 sources is presented in (22)) Those works concern essentially the two-element Boolean Algebra $(\{0,1\}, \vee, \wedge, \neg, 0, 1)$.

In this section, we focus on the Boolean algebra of n-variable switching functions $F_n(B)$. We consider a Boolean system composed of m relations among members of $F_n(B)$ for which k of them are considered as *unknowns*.

3.3.1. Canonic form of a Boolean system of k unknowns over Boolean Algebra $F_n(B)$

Consider the Boolean algebra of n-variable switching functions $(F_n(B), +, \cdot, \bar{}, 0, 1)$.

- Let $(\mathsf{f}^1_{\mathsf{Proj}}, \cdots, \mathsf{f}^n_{\mathsf{Proj}})$ be the n *projection-functions* of $F_n(B)$.
- Let $(x_1, \cdots, x_k)$ be k elements of $F_n(B)$ considered as *unknowns*.

For notational convenience, we note 'X_k' the vector $(x_1, \cdots, x_k)$ of the k unknowns and 'Proj' the vector $(\mathsf{f}^1_{\mathsf{Proj}}, \cdots, \mathsf{f}^n_{\mathsf{Proj}})$ of the n projection-functions of $F_n(B)$.

Theorem 2 (Reduction of a set of relations)
Any set of simultaneously-asserted relations of n-variable switching functions can be reduced to a single equivalent relation such as:

$$\mathcal{F}(\mathsf{X}_k, \mathsf{Proj}) = 0$$

To obtain this equivalent relation, it is necessary:

- to rewrite each equality according to:

$$\begin{aligned}\mathcal{F}_1(\mathsf{X}_k, \mathsf{Proj}) = &\mathcal{F}_2(\mathsf{X}_k, \mathsf{Proj}) \quad \Leftrightarrow \\ &\mathcal{F}_1(\mathsf{X}_k, \mathsf{Proj}) \cdot \overline{\mathcal{F}_2(\mathsf{X}_k, \mathsf{Proj})} + \\ &\overline{\mathcal{F}_1(\mathsf{X}_k, \mathsf{Proj})} \cdot \mathcal{F}_2(\mathsf{X}_k, \mathsf{Proj}) = 0\end{aligned}$$

- to rewrite each inclusion according to:

$$\begin{aligned}\mathcal{F}_1(\mathsf{X}_k, \mathsf{Proj}) \leq &\mathcal{F}_2(\mathsf{X}_k, \mathsf{Proj}) \quad \Leftrightarrow \\ &\mathcal{F}_1(\mathsf{X}_k, \mathsf{Proj}) \cdot \overline{\mathcal{F}_2(\mathsf{X}_k, \mathsf{Proj})} = 0\end{aligned}$$

- to group together rewritten equalities as follows:

$$\begin{cases}\mathcal{F}_1(\mathsf{X}_k, \mathsf{Proj}) = 0 \\ \mathcal{F}_2(\mathsf{X}_k, \mathsf{Proj}) = 0\end{cases} \Leftrightarrow$$
$$\mathcal{F}_1(\mathsf{X}_k, \mathsf{Proj}) + \mathcal{F}_2(\mathsf{X}_k, \mathsf{Proj}) = 0$$

In order to able to write a canonic form for a Boolean system of k unknowns over Boolean algebra $F_n(B)$, we introduce the following notation: for $x \in F_n(B)$ and $a \in \{0,1\}$, we define x^a by

$$x^0 = \bar{x} \qquad x^1 = x$$

This notation is extended to vectors as follows: for $\mathsf{X}_k = (x_1, \cdots, x_k) \in F_n(B)^k$ and $\mathsf{A}_k = (a_1, \cdots, a_k) \in \{0,1\}^k$, we define $\mathsf{X}_k^{A_k}$ by

$$\mathsf{X}_k^{A_k} = \prod_{i=1}^{i=k} x_i^{a_i} = x_1^{a_1} \cdot \ldots \cdot x_k^{a_k}$$

Theorem 3 (Canonic form of a Boolean equation)
Any Boolean equation $Eq(X_k, Proj) = 0$ can be expressed with the canonic form

$$\sum_{A_k \in \{0,1\}^k} Eq(A_k, Proj) \cdot X_k^{A_k} = 0$$

where $Eq(A_k, Proj)$ (with $A_k \in \{0,1\}^k$) are the 2^k discriminants[7] of $Eq(X_k, Proj)$ according to X_k.

This canonic form is obtained by expanding $\mathsf{Eq}(\mathsf{X}_k, \mathsf{Proj})$ according to the k unknowns $(x_1, \cdots, x_k)$. For example, we have:

$$\begin{aligned}&\mathsf{Eq}(x, \mathsf{Proj}) = \mathsf{Eq}(0, \mathsf{Proj}) \cdot \bar{x} + \mathsf{Eq}(1, \mathsf{Proj}) \cdot x \\ &\mathsf{Eq}(x_1, x_2, \mathsf{Proj}) = \\ &\quad \mathsf{Eq}(0, 0, \mathsf{Proj}) \cdot \overline{x_1} \cdot \overline{x_2} + \mathsf{Eq}(0, 1, \mathsf{Proj}) \cdot \overline{x_1} \cdot x_2 \\ &\quad + \mathsf{Eq}(1, 0, \mathsf{Proj}) \cdot x_1 \cdot \overline{x_2} + \mathsf{Eq}(1, 1, \mathsf{Proj}) \cdot x_1 \cdot x_2\end{aligned}$$

3.3.2. Solution of a single-unknown equation over $F_n(B)$

The result presented on the following theorem was initially obtained for the two-element Boolean Algebra $(\{0,1\}, \vee, \wedge, \neg, 0, 1)$. A generalization for all Boolean Algebra can be found in (3).

Theorem 4 (Solution of a single-unknown equation)
The Boolean equation over $F_n(B)$

$$Eq(x, Proj) = 0 \tag{1}$$

for which the canonic form is

$$Eq(0, Proj) \cdot \bar{x} + Eq(1, Proj) \cdot x = 0$$

[7] The term of 'discriminant' comes from (3).

is consistent (i.e. has at least one solution) if and only if the following condition is satisfied:

$$Eq(0, Proj) \cdot Eq(1, Proj) = 0 \tag{2}$$

In this case, a general form of the solutions is:

$$x = Eq(0, Proj) + p \cdot \overline{Eq(1, Proj)} \tag{3}$$

where p is an arbitrary parameter, i.e, a freely-chosen member of $F_n(B)$.

This solution can also be expressed as follows:

$$\begin{aligned} x &= \overline{Eq(1, Proj)} \cdot (Eq(0, Proj) + p) \\ &= \overline{p} \cdot Eq(0, Proj) + p \cdot \overline{Eq(1, Proj)} \end{aligned}$$

The proof of this theorem includes four steps:

- (1) is consistent if and only if (2).
- (3) is a solution of (1) if (2).
- Each solution of (1) can be expressed as (3).
- If (2), the 3 parametric forms are equivalent.

As p is a freely-chosen member of $F_n(B)$, the proposed parametric forms allows to describe all the solutions of (1). It should be noted that the same solution can be described with two different parameters. To establish a bijection between the set of solutions and the set of value of the parameter, it is necessary to impose the following constraint to the parameter p: $p \leq \overline{\mathsf{Eq}(0, \mathsf{Proj})} \cdot \overline{\mathsf{Eq}(1, \mathsf{Proj})}$

3.3.3. Solution of k-unknown equations over $F_n(B)$
The global result presented in the following theorem can be found in (22) or (3). However, in those works, the solution is not expressed with a parametric form, but with intervals only. The formulation presented in this communication is more adapted to symbolic computation and is mandatory to practice optimization. The proof of this version of this theorem can be found in (9).

A k-unknown equation can be solved by solving successively k single-unknown equations. If we consider the k-unknown equation as a single-unknown equation of x_k, its consistence condition corresponds to a $(k-1)$-unknown equation. The process can be iterated until x_1. After substituting $S(x_1)$ for x_1 in the last equation, it is possible to find the solution for x_2. Then, it is sufficient to apply this procedure again $(k-2)$ times to obtain successively the solutions $S(x_3)$ to $S(x_k)$.

Theorem 5 (Solution of a k-unknown equation)
The Boolean equation over $F_n(B)$

$$Eq_0(X_k, Proj) = 0 \tag{4}$$

is consistent (i.e. has at least one solution) if and only if the following condition is satisfied:

$$\prod_{A_k \in \{0,1\}^k} Eq_0(A_k, Proj) = 0 \tag{5}$$

When (5) *is satisfied, Equation* (4) *admits one or more k-tuple solutions $(S(x_1), \cdots, S(x_k))$ where each component $S(x_i)$ is defined by*

$$\begin{aligned} S(x_i) = &\prod_{A_{k-i} \in \{0,1\}^{k-i}} Eq_{i-1}(0, A_{k-i}, Proj) \\ &+ p_i \cdot \overline{\prod_{A_{k-i} \in \{0,1\}^{k-i}} Eq_{i-1}(1, A_{k-i}, Proj)} \end{aligned} \tag{6}$$

where

- $Eq_i(x_{i+1}, \cdots, x_k, Proj) = Eq_{i-1}(x_i, x_{i+1}, \cdots, x_k, Proj)|_{x_i \leftarrow S(x_i)}$
- p_i *is a arbitrary parameter, i.e, a freely-chosen member of* $F_n(B)$.

It is important to note that the order in which unknowns are treated affects only the parametric form of the k-tuple solution. This is due to the fact that the same k-tuple solution can be represented with several distinct parametric form.

3.3.4. Partial conclusions
Thanks to theorems presented in this section, it is possible to obtain a parametric representation of all the solutions of any set of simultaneously-asserted relations with k unknowns if a solution exists.

As the boolean algebra $F_n(B)$ is equipped with a partial order, the comparison of solutions according to a given criterion can be envisaged.

3.4. Optimal solutions of Boolean equations over $F_n(B)$

The goal is to be able to obtain automatically the parametric form of the k-tuples solutions of $F_n(B)$ which satisfy not only a given equation ($\mathsf{Eq}(\mathsf{X}_k, \mathsf{Proj}) = 0$) of boolean functions but also maximize (or minimize) a boolean formula of these boolean functions ($\mathcal{F}_\mathsf{C}(\mathsf{X}_k, \mathsf{Proj})$) corresponding to the desired optimization criterion.

Generally speaking, the search of the best solution tuples according to a given criterion when the space of solutions is composed of discrete values is a complex mathematical issue. It is sometimes necessary to make a side-by-side comparison of each solution in order to identify the best one. In our case, this exhaustive method cannot be used as $F_n(B)$ is only provided by a partial order: two particular solutions can not always be ordered between themselves.

Yet it is possible to obtain the researched parametric form of the k-tuples thanks to the following results:

- When an equation between boolean functions has one or more solution tuples in $F_n(B)$, every Boolean formula established from these Boolean functions can be rewritten thanks to only projection-functions of $F_n(B)$ and free parameters of $F_n(B)$ which are describing these solution tuples.
- Every boolean formula established from projection-functions of $F_n(B)$ and free parameters of $F_n(B)$ has an unique maximum and an unique minimum. These extrema can be expressed thanks to only projection-functions of $F_n(B)$.

Hence it is then possible to rewrite the initial problem:

$$\begin{cases} \text{Problem to solve:} \\ \quad \mathsf{Eq}(\mathsf{X}_k, \mathsf{Proj}) = 0 \\ \text{Optimization Criterion:} \\ \quad \text{Maximization of } \mathcal{F}_{\mathsf{C}}(\mathsf{X}_k, \mathsf{Proj}) \end{cases}$$

into a 2 equations system to solve

$$\begin{cases} \mathsf{Eq}(\mathsf{X}_k, \mathsf{Proj}) = 0 \\ \mathcal{F}_{\mathsf{C}}(\mathsf{X}_k, \mathsf{Proj}) = \mathcal{F}_{\mathsf{MaxC}}(\mathsf{Proj}) \end{cases}$$

where

$$\mathcal{F}_{\mathsf{MaxC}}(\mathsf{Proj}) = \underset{\{\mathsf{X}_k \mid \mathsf{Eq}(\mathsf{X}_k, \mathsf{Proj})=0\}}{\mathsf{Max}} (\mathcal{F}_{\mathsf{C}}(\mathsf{X}_k, \mathsf{Proj}))$$

3.4.1. Extrema of a Boolean formula according to freely-chosen members of $F_n(B)$

Considering the Boolean algebra of n-variable switching functions $(F_n(B), +, \cdot, \bar{}, 0, 1)$:

- Let $(\mathsf{f}^1_{\mathsf{Proj}}, \cdots, \mathsf{f}^n_{\mathsf{Proj}})$ be the n *projection-functions* of $F_n(B)$.
- Let $(p_1, \cdots, p_k)$ be k elements of $F_n(B)$ considered as *freely-chosen members*. Let 'P_k' be the corresponding vector.

Any formula $\mathcal{F}(\mathsf{P}_k, \mathsf{Proj})$ for which P_k are freely-chosen members of $F_n(B)$ defines a subset of $F_n(B)$. According to the relation $\leq$, elements of this subset can be compared. In this specific case, the subset defined by $\mathcal{F}(\mathsf{P}_k, \mathsf{Proj})$ admits a minimal element and a maximal element.

Theorem 6 (Minimum of a Boolean formula) *Any formula $\mathcal{F}$(P$_k$,Proj) for which P$_k$ are freely-chosen members of $F_n(B)$ admits a minimum defined as follows:*

$$\underset{P_k \in F_n(B)^k}{\mathit{Min}} (\mathcal{F}(P_k, \mathit{Proj})) = \prod_{A_k \in \{0,1\}^k} \mathcal{F}(A_k, \mathit{Proj})$$

Proof: To prove this theorem, it is necessary to establish that:
a) $\prod_{\mathsf{A}_k \in \{0,1\}^k} \mathcal{F}(\mathsf{A}_k, \mathsf{Proj})$ is a lower bound of $\mathcal{F}(\mathsf{P}_k, \mathsf{Proj})$.
b) It exists at least one specific combination of P_k for which $\mathcal{F}(\mathsf{P}_k, \mathsf{Proj}) = \prod_{\mathsf{A}_k \in \{0,1\}^k} \mathcal{F}(\mathsf{A}_k, \mathsf{Proj})$.

To establish a), it is sufficient to prove:

$$\prod_{\mathsf{A}_k \in \{0,1\}^k} \mathcal{F}(\mathsf{A}_k, \mathsf{Proj}) \leq \mathcal{F}(\mathsf{P}_k, \mathsf{Proj})$$

To establish b), it is sufficient to prove that the following equation admits solution:

$$\mathcal{F}(\mathsf{X}_k, \mathsf{Proj}) = \prod_{\mathsf{A}_k \in \{0,1\}^k} \mathcal{F}(\mathsf{A}_k, \mathsf{Proj})$$

Details of this proof can be found in (13).

□

Theorem 7 (Maximum of a Boolean formula) *Any formula $\mathcal{F}$(P$_k$,Proj) for which P$_k$ are freely-chosen members of $F_n(B)$ admits a maximum defined as follows:*

$$\underset{P_k \in F_n(B)^k}{\mathit{Max}} (\mathcal{F}(P_k, \mathit{Proj})) = \sum_{A_k \in \{0,1\}^k} \mathcal{F}(A_k, \mathit{Proj})$$

Proof: To prove this theorem, it is necessary to establish that:
a) $\sum_{\mathsf{A}_k \in \{0,1\}^k} \mathcal{F}(\mathsf{A}_k, \mathsf{Proj})$ is an upper bound of $\mathcal{F}(\mathsf{P}_k, \mathsf{Proj})$.
b) It exists at least one specific combination of P_k for which $\mathcal{F}(\mathsf{P}_k, \mathsf{Proj}) = \sum_{\mathsf{A}_k \in \{0,1\}^k} \mathcal{F}(\mathsf{A}_k, \mathsf{Proj})$.

To establish a), it is sufficient to prove:

$$\mathcal{F}(\mathsf{P}_k, \mathsf{Proj}) \leq \sum_{\mathsf{A}_k \in \{0,1\}^k} \mathcal{F}(\mathsf{A}_k, \mathsf{Proj})$$

To establish b), it is sufficient to prove that the following equation admits solution:

$$\mathcal{F}(\mathsf{X}_k, \mathsf{Proj}) = \sum_{\mathsf{A}_k \in \{0,1\}^k} \mathcal{F}(\mathsf{A}_k, \mathsf{Proj})$$

Details of this proof can be found in (13).

□

3.4.2. Solving on optimization problem

Considering the Boolean algebra of n-variable switching functions $(F_n(B), +, \cdot, \bar{}, 0, 1)$:

- Let $(\mathsf{f}^1_{\mathsf{Proj}}, \cdots, \mathsf{f}^n_{\mathsf{Proj}})$ be the n *projection-functions* of $F_n(B)$. Let 'Proj' be the corresponding vector.

- Let $(x_1, \cdots, x_k)$ be k elements of $F_n(B)$ considered as *unknowns*. Let 'X_k' be the corresponding vector.
- Let $(p_1, \cdots, p_k)$ be k elements of $F_n(B)$ considered as *freely-chosen members*. Let 'P_k' be the corresponding vector.
- Let $\mathsf{Eq}(\mathsf{X}_k, \mathsf{Proj}) = 0$ be the Boolean equation to solve.
- Let $\mathcal{F}_{\mathsf{C}}(\mathsf{X}_k, \mathsf{Proj})$ be the Boolean formula of the given criterion to optimize (maximization or minimization).

The method we propose, to obtain the parametric form of the k-tuple of switching functions solution of $\mathsf{Eq}(\mathsf{X}_k, \mathsf{Proj}) = 0$ according to the given optimization criterion $\mathcal{F}_{\mathsf{C}}(\mathsf{X}_k, \mathsf{Proj})$ is composed of five steps:

- The first step consists to establish the parametric form of the k-tuple solution of $\mathsf{Eq}(\mathsf{X}_k, \mathsf{Proj}) = 0$ only, thanks to Theorem 5.
- The second step consists to establish the parametric form of the given optimization criterion $\mathcal{F}_{\mathsf{C}}(\mathsf{X}_k, \mathsf{Proj})$ by substituting $S(x_i)$ for x_i. Let $\mathcal{F}_{\mathsf{SC}}(\mathsf{P}_k, \mathsf{Proj})$ be the result of this substitution.
- The third step consists to calculate the extremum of $\mathcal{F}_{\mathsf{SC}}(\mathsf{P}_k, \mathsf{Proj})$ according to Theorem 6 or Theorem 7. Let $\mathcal{F}_{\mathsf{EC}}(\mathsf{Proj})$ be the Boolean formula of this extremum.
- The fourth step consists to replace the given criterion by the equivalent relation:

$$\mathcal{F}_{\mathsf{C}}(\mathsf{X}_k, \mathsf{Proj}) = \mathcal{F}_{\mathsf{EC}}(\mathsf{Proj})$$

- The fifth step consists to establish the parametric form of the k-tuple solution of the equivalent problem:

$$\begin{cases} \mathsf{Eq}(\mathsf{X}_k, \mathsf{Proj}) = 0 \\ \mathcal{F}_{\mathsf{Crit}}(\mathsf{X}_k, \mathsf{Proj}) = \mathcal{F}_{\mathsf{ExtCrit}}(\mathsf{Proj}) \end{cases}$$

3.4.3. Partial conclusions

Thanks to the theorems presented in this section, it is now possible to obtain a parametric representation of the optimal solutions according to a given criterion, of any set of simultaneously-asserted relations with k unknowns if a solution exists.

The proposed method also permits to associate simultaneously or sequentially several criteria.

- When several criteria are treated simultaneously, the equivalent problem obtained can be incoherent and admits no solution. That is the case when given criteria are antagonist.
- When several criteria are treated sequentially, the obtained solutions satisfy given criteria with a priority order. The first criterion is satisfied without restriction. The second criterion is used only to order solutions which satisfy the first criterion. Therefore, it is possible to obtain solutions.

An example of optimization with several criteria treated sequentially is presented in the next section.

4. ALGEBRAIC SYNTHESIS OF LOGICAL CONTROLLERS WITH OPTIMIZATION CRITERIA

The proposed method is illustrated with the complete treatment of the synthesis of the control law of a water distribution system. The system and its specifications are given in Section 4.1. The formalization of this problem with the Boolean algebra of n-variable switching functions $F_n(B)$ is presented in Section 4.2. The resolution of this problem with the results presented in Section 3.4 is presented in Section 4.3. The state model of the obtained control law is given in Section 4.4.

4.1. Control system specifications

The studied system is the controller of a water distribution system composed of two pumps which are working in redundancy. The water distribution is made when it is necessary according to the possible failures of elements (the pumps and the distributing system).

4.1.1. Inputs and outputs of the controller

The Boolean inputs and outputs of this controller are given in Fig. 3. Each pump is controlled thanks to a Boolean output ('p1' and 'p2'). The controller is informed of water distribution requests thanks to the input 'req'. Inputs 'f1' and 'f2' inform the controller of a failure of the pumps and 'gf' of the global failure of the installation. The input 'Pr' precises which pump has priority.

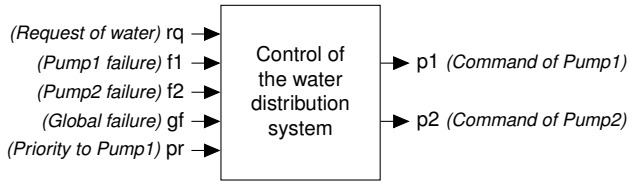


Figure 3: *Inputs and outputs of the controller to design*

4.1.2. Expected behavior

The expected behavior of the control system regarding the application requirements can be expressed by the set of assertions given hereafter:

- The two pumps never operate simultaneously.

- A pump cannot operate if it is out of order.
- When a global failure is detected, no pump can operate.
- Pumps can operate only if a water distribution request is present.
- Pumps operate when it is possible. Priority is given according to 'pr' (Pump1 has priority when 'pr' is true).
- In order to reduce the wear of the pump, it is necessary to restrict the number of starting of the pumps.

4.1.3. Control laws to design

Our approach does not permit to identify automatically which state variables must be used. They are given by the designer according to its interpretation of the specification. For the water distribution system, we propose to use two state variables: one for each output. According to this choice, we have two 7-variable switching functions to synthesize. The generic form of the control law we want to design is:

$$\begin{cases} \mathsf{p1}[k] = \mathsf{P1}(\mathsf{rq}[k], \mathsf{f1}[k], \mathsf{f2}[k]\, \mathsf{gf}[k], \mathsf{pr}[k], \\ \qquad\qquad\qquad\qquad \mathsf{p1}[k-1], \mathsf{p2}[k-1]) \\ \mathsf{p2}[k] = \mathsf{P2}(\mathsf{rq}[k], \mathsf{f1}[k], \mathsf{f2}[k]\, \mathsf{gf}[k], \mathsf{pr}[k], \\ \qquad\qquad\qquad\qquad \mathsf{p1}[k-1], \mathsf{p2}[k-1]) \\ \mathsf{p1}[0] = {}_b0 \quad \mathsf{p2}[0] = {}_b0 \end{cases}$$

As P1 and P2 are two 7-variable switching functions, this simple model permits to express 2^{256} $((2^{2^7})^2)$ different control laws. This explains why the manual synthesis of control laws expressed with recurrent Boolean equations is so difficult.

4.2. Formalization with the Boolean algebra of n-variable switching functions $F_n(B)$

The first step of the proposed method consists to formalize the expected behavior with relations between formula of n-variable switching functions. in order to do this, it is necessary to chose the size of the Boolean algebra to use and to precise the n projection-functions.

4.2.1. Choice of the Boolean Algebra to use

The Boolean algebra to use is fixed by the inputs of the controller to synthesize and the state variables chosen for the control laws.
In this case study, we have 5 inputs and 2 state variables. Then, the 7 projection-functions are:

- The 5 switching functions (Rq, F1, F2, GF, and Pr) which characterize the behavior of the inputs of the controller and are defined as follows:

$$\begin{aligned} \mathsf{Rq}: \quad & B^7 \to B \\ & (\mathsf{rq}[k], \ldots, \mathsf{p2}[k-1]) \mapsto \mathsf{rq}[k] \end{aligned}$$

- The 2 switching functions (${}_p\mathsf{P1}$ and ${}_p\mathsf{P2}$) which characterize the previous behavior of the state variables of the controller and are defined as follows:

$$\begin{aligned} {}_p\mathsf{P1}: \quad & B^7 \to B \\ & (\mathsf{rq}[k], \ldots, \mathsf{p2}[k-1]) \mapsto \mathsf{p1}[k-1] \end{aligned}$$

In our case, only 2 switching functions must be designed (P1 and P2). They represent the unknowns or our problem.

4.2.2. Formalization of requirements

Assertions describing the expected behavior of control systems in natural language can be translated into formal statements thanks to the relations Equality and Inclusion.

To illustrate the possibility of expression given by these two relations, several examples (generic assertions and equivalent formal relations illustrated on the case study) are given hereafter:

- Pump1 and Pump2 never operate simultaneously: $\mathsf{P1} \cdot \mathsf{P2} = 0$
- If Pump1 operates, Pump2 can not operate: $\mathsf{P1} \leq \overline{\mathsf{P2}}$
- If Pump1 is out of order (F1), Pump1 can not operate: $\mathsf{F1} \leq \overline{\mathsf{P1}}$
- Pump 1 can operate only if it is not out of order: $\mathsf{P1} \leq \overline{\mathsf{F1}}$
- When a global failure is detected, no pump can operate: $\mathsf{GF} \leq (\overline{\mathsf{P1}} \cdot \overline{\mathsf{P2}})$
- It is necessary to have a request for pumps operate: $(\mathsf{P1} + \mathsf{P2}) \leq \mathsf{Rq}$
- It is sufficient to have a request for pumps operate: $\mathsf{Rq} \leq (\mathsf{P1} + \mathsf{P2})$
- When Pump1 is failed, it is sufficient to have a request for Pump2 operate: $\mathsf{F1} \cdot \mathsf{Rq} \leq \mathsf{P2}$
- When Pump1 is failed, it is necessary to have a request for Pump2 operate: $\mathsf{F1} \cdot \mathsf{P2} \leq \mathsf{Rq}$

As P1 and ${}_p\mathsf{P1}$ represent the behavior of Pump1 at respectively times $[k]$ and $[k-1]$, it is also possible to express relations about starts and stops of this pump:

- It is necessary to have a request to start Pump1: $(\mathsf{P1} \cdot \overline{{}_p\mathsf{P1}}) \leq \mathsf{Rq}$
- When Pump1 operates, it is sufficient to have a global failure to stop Pump1: $({}_p\mathsf{P1} \cdot \mathsf{GF}) \leq (\overline{\mathsf{P1}} \cdot {}_p\mathsf{P1})$

Thanks to formal representations, it is possible to prove that some of them are equivalent (for example, the first two, the third and the fourth). When a designer hesitates between two forms, he has the possibility, by using symbolic calculation, to check if the proposed relations are equivalent or not.

It is important to note that the relation Inclusion permits to express distinctly necessary conditions and sufficient conditions. This relation is the cornerstone of our approach.

4.3. Resolution with optimal criteria

The result of the formalization of all the requirements is composed of a set of relations for which the solutions must be selected according to three criteria treated sequentially (Fig. 4).

$$\begin{cases} \textit{Problem to solve:} \\ \quad \begin{cases} \mathsf{P1} \cdot \mathsf{P2} = 0 \\ \mathsf{F1} \leq \overline{\mathsf{P1}} \\ \mathsf{F2} \leq \overline{\mathsf{P2}} \\ \mathsf{GF} \leq \overline{\mathsf{P1}} \cdot \overline{\mathsf{P2}} \\ (\mathsf{P1} + \mathsf{P2}) \leq \mathsf{Rq} \end{cases} \\ \textit{Optimization Criteria:} \\ \quad \textbf{1.} \text{ Maximization of: } (\mathsf{P1} + \mathsf{P2}) \\ \quad \textbf{2.} \text{ Minimization of: } ((\mathsf{P1} \cdot \overline{{}_p\mathsf{P1}}) + (\mathsf{P2} \cdot \overline{{}_p\mathsf{P2}})) \\ \quad \textbf{3.} \text{ Maximization of: } ((\mathsf{Pr} \cdot \mathsf{P1}) + (\overline{\mathsf{Pr}} \cdot \mathsf{P2})) \end{cases}$$

Figure 4: *Complete specification of the controller to design*

The first criterion corresponds to the maximization of the distribution of water. The second is the minimization of the possibility to start a pump. The last criterion consists to maximize the priority order between the two pumps (Pump1 operates when 'Pr' is true and Pump2 operates when 'Pr' is false).

For this problem, the method we propose permits to prove that proposed criteria cannot be treated simultaneously as some of them are antagonist (it is not possible to maximize the water distribution if the pumps never start). A resolution with the two first criteria conducts to an incoherent equivalent system. It is necessary to order given criteria according to their priorities (from maximal priority to minimal).

The details of the first optimization is given hereafter:
a) First resolution without optimization criterion:

$$\begin{cases} \mathsf{P1} \cdot \mathsf{P2} = 0 \\ \mathsf{F1} \leq \overline{\mathsf{P1}} \\ \mathsf{F2} \leq \overline{\mathsf{P2}} \\ \mathsf{GF} \leq \overline{\mathsf{P1}} \cdot \overline{\mathsf{P2}} \\ (\mathsf{P1} + \mathsf{P2}) \leq \mathsf{Rq} \end{cases} \Leftrightarrow \begin{cases} \mathsf{P1} = p_1 \cdot \mathsf{Rq} \cdot \overline{\mathsf{GF}} \cdot \overline{\mathsf{F1}} \\ \mathsf{P2} = p_2 \cdot \mathsf{Rq} \cdot \overline{\mathsf{GF}} \cdot \overline{\mathsf{F2}} \\ \qquad\qquad \cdot (\mathsf{F1} + \overline{p_1}) \\ (p_1, p_2) \in F_7(B)^2 \end{cases}$$

b) Parametric form of the given criterion:

$$(\mathsf{P1} + \mathsf{P2}) = \mathsf{Rq} \cdot \overline{\mathsf{GF}} \cdot (p_1 \cdot \overline{\mathsf{F1}} + p_2 \cdot \overline{\mathsf{F2}})$$

c) Calcul of the maximum of the given criterion:

$$\underset{(p_1,p_2)\in F_7(B)^2}{\mathsf{Max}} (\mathsf{P1} + \mathsf{P2}) = \mathsf{Rq} \cdot \overline{\mathsf{GF}} \cdot (\overline{\mathsf{F1}} + \overline{\mathsf{F2}})$$

d) Second resolution with the optimization criterion:

$$\begin{cases} \mathsf{P1} \cdot \mathsf{P2} = 0 \\ \mathsf{F1} \leq \overline{\mathsf{P1}} \\ \mathsf{F2} \leq \overline{\mathsf{P2}} \\ \mathsf{GF} \leq \overline{\mathsf{P1}} \cdot \overline{\mathsf{P2}} \\ (\mathsf{P1} + \mathsf{P2}) \leq \mathsf{Rq} \\ (\mathsf{P1} + \mathsf{P2}) = \mathsf{Rq} \cdot \overline{\mathsf{GF}} \cdot (\overline{\mathsf{F1}} + \overline{\mathsf{F2}}) \end{cases} \Leftrightarrow \begin{cases} \mathsf{P1} = \mathsf{Rq} \cdot \overline{\mathsf{GF}} \cdot \overline{\mathsf{F1}} \cdot (p_1 + \mathsf{F2}) \\ \mathsf{P2} = \mathsf{Rq} \cdot \overline{\mathsf{GF}} \cdot \overline{\mathsf{F2}} \cdot (\mathsf{F1} + \overline{p_1}) \\ p_1 \in F_7(B) \end{cases}$$

The same operations are made for the second and the third criteria. The last equivalent Boolean equations system to solve and its solution are given in Fig 5.

$$\begin{cases} \mathsf{P1} \cdot \mathsf{P2} = 0 \\ \mathsf{F1} \leq \overline{\mathsf{P1}} \\ \mathsf{F2} \leq \overline{\mathsf{P2}} \\ \mathsf{GF} \leq \overline{\mathsf{P1}} \cdot \overline{\mathsf{P2}} \\ (\mathsf{P1} + \mathsf{P2}) \leq \mathsf{Rq} \\ (\mathsf{P1} + \mathsf{P2}) = \mathsf{Rq} \cdot \overline{\mathsf{GF}} \cdot (\mathsf{P1} + \mathsf{P2}) \\ ((\mathsf{P1} \cdot \overline{{}_p\mathsf{P1}}) + (\mathsf{P2} \cdot \overline{{}_p\mathsf{P2}})) \\ \quad = \mathsf{Rq} \cdot \overline{\mathsf{GF}} \cdot (\overline{\mathsf{F1}} \cdot \mathsf{F2} \cdot \overline{{}_p\mathsf{P1}} + \mathsf{F1} \cdot \overline{\mathsf{F2}} \cdot \overline{{}_p\mathsf{P2}} \\ \qquad\qquad + \overline{\mathsf{F1}} \cdot \overline{\mathsf{F2}} \cdot \overline{{}_p\mathsf{P1}} \cdot \overline{{}_p\mathsf{P2}}) \\ ((\mathsf{Pr} \cdot \mathsf{P1}) + (\overline{\mathsf{Pr}} \cdot \mathsf{P2})) \\ \quad = \mathsf{Rq} \cdot \overline{\mathsf{GF}} \cdot (\overline{\mathsf{F1}} \cdot \mathsf{P2} \cdot (\mathsf{F2} + {}_p\mathsf{P1} + \overline{{}_p\mathsf{P2}}) \\ \qquad\qquad + \overline{\mathsf{F2}} \cdot \overline{\mathsf{P2}} \cdot (\mathsf{F1} + {}_p\mathsf{P2} + \overline{{}_p\mathsf{P1}})) \end{cases}$$

$$\Leftrightarrow \begin{cases} \mathsf{P1} = \mathsf{Rq} \cdot \overline{\mathsf{GF}} \cdot \overline{\mathsf{F1}} \\ \qquad \cdot (\mathsf{F2} + \mathsf{Pr} \cdot ({}_p\mathsf{P1} + \overline{{}_p\mathsf{P2}}) + {}_p\mathsf{P1} \cdot \overline{{}_p\mathsf{P2}}) \\ \mathsf{P2} = \mathsf{Rq} \cdot \overline{\mathsf{GF}} \cdot \overline{\mathsf{F2}} \\ \qquad \cdot (\mathsf{F1} + \overline{\mathsf{Pr}} \cdot ({}_p\mathsf{P2} + \overline{{}_p\mathsf{P1}}) + {}_p\mathsf{P2} \cdot \overline{{}_p\mathsf{P1}}) \end{cases}$$

Figure 5: *Last equivalent Boolean equations to solve and its solution*

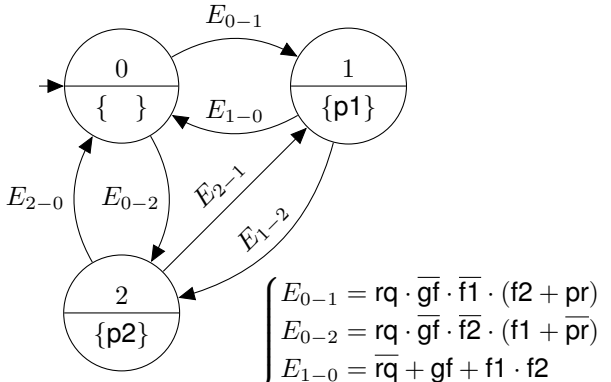
4.4. Obtained contol laws

The control laws presented hereafter was obtained by translating the expression of the unknowns according to the projection-functions into relations between recurrent Boolean equations. This control law was implemented into a PLC with the Ladder

Diagram language. The code is composed of only four rungs.

$$\begin{cases} \mathsf{p1}[k] = \mathsf{rq}[k] \cdot \overline{\mathsf{gf}[k]} \cdot \overline{\mathsf{f1}[k]} \\ \qquad \cdot (\mathsf{f2}[k] + \mathsf{pr}[k] \cdot (\mathsf{p1}[k-1] + \overline{\mathsf{p2}[k-1]}) \\ \qquad\quad + \mathsf{p1}[k-1] \cdot \overline{\mathsf{p2}[k-1]})) \\ \mathsf{p2}[k] = \mathsf{rq}[k] \cdot \overline{\mathsf{gf}[k]} \cdot \overline{\mathsf{f1}[k]} \\ \qquad \cdot (\mathsf{f1}[k] + \overline{\mathsf{pr}[k]} \cdot (\mathsf{p2}[k-1] + \overline{\mathsf{p1}[k-1]}) \\ \qquad\quad + \mathsf{p2}[k-1] \cdot \overline{\mathsf{p1}[k-1]}) \\ \mathsf{p1}[0] = {}_b0 \quad \mathsf{p2}[0] = {}_b0 \end{cases}$$

If this form is well-adapted for an implementation, its representation with a state model (Fig. 6) simplifies the work of the designer. For this control law, the equivalent state model (automatically built thanks to (8)) is composed of three states only. For each state, the set of emitted outputs is given. The six transition conditions are a Boolean expression of the inputs. By construction, this state model satisfies all the requirements given Section 4.1.2.



$$\begin{cases} E_{0-1} = \mathsf{rq} \cdot \overline{\mathsf{gf}} \cdot \overline{\mathsf{f1}} \cdot (\mathsf{f2} + \mathsf{pr}) \\ E_{0-2} = \mathsf{rq} \cdot \overline{\mathsf{gf}} \cdot \overline{\mathsf{f2}} \cdot (\mathsf{f1} + \overline{\mathsf{pr}}) \\ E_{1-0} = \overline{\mathsf{rq}} + \mathsf{gf} + \mathsf{f1} \cdot \mathsf{f2} \\ E_{1-2} = \mathsf{rq} \cdot \overline{\mathsf{gf}} \cdot \mathsf{f1} \cdot \overline{\mathsf{f2}} \\ E_{2-0} = \overline{\mathsf{rq}} + \mathsf{gf} + \mathsf{f1} \cdot \mathsf{f2} \\ E_{2-1} = \mathsf{rq} \cdot \overline{\mathsf{gf}} \cdot \mathsf{f2} \cdot \overline{\mathsf{f1}} \end{cases}$$

Figure 6: *State model of the obtained control law*

5. CONCLUSIONS

The aim of the proposed method is to obtain the control law of dependable logical systems from specifications given in natural language. We have chosen to represent these control laws with recurrent Boolean equations, in order to facilitate the formalization of safety requirements and the implementation into a controller as a PLC or an ECU.

These control laws are synthesized by symbolic calculation from relations between n-variable switching functions which represent the functional or dependable requirements. Thanks to mathematical results presented in this communication, we are able to find automatically the optimal solution according to given criteria. This new possibility simplifies greatly the designer's work as expected behaviors are simpler to express.

To facilitate the designer's work, we currently develop a complementary module for the formalization of requirements. Its aim is to detect incoherent requirements and to rewrite automatically the requirements, according to priority rules given by the designer among these requirements (21).

REFERENCES

[1] Houda Bel Mokadem, Béatrice Berard, Vincent Gourcuff, Olivier De Smet, and Jean-Marc Roussel. Verification of a timed multitask system with UPPAAL. *IEEE Transactions on Automation Science and Engineering*, 7(4):921 – 932, October 2010.

[2] B. Berard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, and P. Schnoebelen. *Systems and Software Verification: Model-Checking Techniques and Tools*. Springer Publishing Company, 1st edition, 1999.

[3] Frank Markham Brown. *Boolean Reasoning: The Logic of Boolean Equations*. Dover Publications, 2003.

[4] Matteo Cantarelli and Jean-Marc Roussel. Reactive control system design using the Supervisory Control Theory: evaluation of possibilities and limits. In *Proceedings of 9th International Workshop On Discrete Event Systems (WODES'08)*, pages 200–205, Göteborg, Sweden, May 2008.

[5] G. Frey and L. Litz. Formal methods in PLC programming. In *Systems, Man, and Cybernetics, 2000 IEEE International Conference on*, volume 4, pages 2431–2436, 2000.

[6] Vincent Gourcuff, Olivier De Smet, and Jean-Marc Faure. Efficient representation for formal verification of PLC programs. In *Proceedings of 8th International Workshop On Discrete Event Systems (WODES'06)*, pages 182–187, Ann Arbor, USA, July 2006.

[7] Ralph P. Grimaldi. *Discrete and Combinatorial Mathematics: An Applied Introduction*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, Fifth edition, 2004.

[8] Anaïs Guignard. Symbolic generation of the automaton representing an algebraic description of a logic system. Master's thesis, École Normale Supérieure de Cachan, July 2011.

[9] Yann Hietter. *Synthèse algébrique de lois de commande pour les systèmes à évènements discrets logiques*. PhD thesis, École Normale Supérieure de Cachan, May 2009.

[10] Yann Hietter, Jean-Marc Roussel, and Jean-Jacques Lesage. Algebraic synthesis of transition conditions of a state model. In *Proceedings of 9th International Workshop On Discrete Event Systems (WODES'08)*, pages 187–192, Göteborg, Sweden, May 2008.

[11] D. A. Huffman. The synthesis of sequential switching circuits. *J. of the Franklin Institute*, 257(3-4):161–190 and 275–303, 1954.

[12] IEC 61508. *IEC 61508 Standard: Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems*. International Electrotechnical Commission, Geneva, Switwerland, 1998.

[13] Hélène Leroux. Algebraic synthesis of logical controllers with optimization criteria. Master's thesis, École Normale Supérieure de Cachan, July 2011.

[14] José Machado, J.B., Bruno Denis, Jean-Jacques Lesage, Jean-Marc Faure, and Jaime Fereira. Logic controllers dependability verification using a plant model. In *Proceedings of the 3rd IFAC Workshop on Discrete-Event System Design, DESDes'06*, pages pp. 37– 42, Rydzyna (Poland), September 2006.

[15] George H. Mealy. A method for synthesizing sequential circuits. *Bell System Technical Journal*, 34(5):1045–1079, 1955.

[16] Edward F. Moore. Gedanken Experiments on Sequential Machines. In *Automata Studies*, pages 129–153. Princeton U., 1956.

[17] Jean-François Pétin, David Gouyon, and Gérard Morel. Supervisory synthesis for product-driven automation and its application to a flexible assembly cell. *Control Engineering Practice*, 15(5):595–614, 2007.

[18] P. J. G. Ramadge and W. M. Wonham. The control of discrete event systems. *Proceedings of the IEEE Transactions on Automatic Control*, 77(1):81–98, 1989.

[19] Jean-Marc Roussel and Bruno Denis. Safety properties verification of ladder diagram programs. *Journal Européen des Systèmes Automatisés*, 36(7):pp. 905–917, June 2002.

[20] Jean-Marc Roussel and Alessandro Giua. Designing dependable logic controllers using the supervisory control theory. In *Proceedings of the 16th IFAC World Congress, 2005*, Praha, Czech Republic, July 2005. CDROM paper 4427, 6 pages.

[21] Jean-Marc Roussel and Jean-Jacques Lesage. Algebraic synthesis of logical controllers despite inconsistencies in specifications. In *Proceedings of 11th International Workshop On Discrete Event Systems (WODES'2012)*, Guadalajara, Mexico, 2012. To appear, 6 pages.

[22] Sergiu Rudeanu. *Lattice Functions and Equations (Discrete Mathematics and Theoretical Computer Science)*. Springer, 2001.

[23] Litian Xiao, Ming Gu, and Jiaguang Sun. The Verification of PLC Program Based on Interactive Theorem Proving Tool COQ. In *4th IEEE International Conference on Computer Science and Information Technology (ICCSIT2011)*, Chengdu, China, June 2011.