Design of Logic Controllers Thanks to Symbolic Computation of Simultaneously Asserted Boolean Equations

Jean-Marc Roussel, Jean-Jacques Lesage

Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2014, Article ID 726246, 15 pages
http://dx.doi.org/10.1155/2014/726246

Research Article

Design of Logic Controllers Thanks to Symbolic Computation of Simultaneously Asserted Boolean Equations

Jean-Marc Roussel and Jean-Jacques Lesage

LURPA, ENS Cachan, 61 avenue du Président Wilson, 94230 Cachan, France

Correspondence should be addressed to Jean-Marc Roussel; jean-marc.roussel@lurpa.ens-cachan.fr

Received 11 December 2013; Revised 6 February 2014; Accepted 7 February 2014; Published 28 May 2014

Academic Editor: Hamid R. Karimi

Formal methods can strongly contribute to improve dependability of controllers during design, by providing means to avoid flaws due to designers' omissions or specifications misinterpretations. This paper presents a synthesis method dedicated to logic controllers. Its goal is to obtain the control laws from specifications given in natural language by symbolic computation. The formal framework that underlies this method is the Boolean algebra of n-variable switching functions. In this algebra, thanks to relations and theorems presented in this paper, it is possible to formally express logical controllers specifications, to automatically detect inconsistencies in specifications, and to obtain automatically the set of solutions or to choose an optimal solution according to given optimization criteria. The application of this synthesis method to an example allows illustrating its main advantages.

1. Introduction

Programmable logic controllers (PLCs) are industrial automation components that receive input signals coming from sensors and send output signals to actuators, in accordance with control laws implemented into a user program (Figure 1). The control algorithms that allow the real time calculation of new output values, according to the current state of the PLC and the observation of new values of inputs, are written in standardized languages, such as ladder diagram (LD), structured text (ST) or instruction list (IL) [1]. A PLC cyclically performs three tasks: inputs reading, program execution, and outputs updating. The period of this task may be constant (periodic scan) or may vary (cyclic scan).

Because of their reliability, even in very severe conditions in terms of temperature, vibrations, electromagnetic perturbations, and so forth, PLCs are frequently used for the control of safety-critical systems (energy production, transport, chemical industry, etc.). In this context, improving the reliability of the user program has been one of the main challenges of the past two decades in the field of automation. Among the different techniques that can be used in this aim [2], formal verification and validation and formal synthesis are the most efficient. Verification is the proof that the internal semantics of a model is correct, independently from the modeled system. The searched properties of the models are stability, deadlock existence, and so on. The validation determines if the model agrees with the designer's purpose [3]. Efficient validation/verification techniques of PLC programs [4], most often based on model-checking technique, have been proposed by researchers and are now widely used in industry [5], despite problems of state-space explosion that arise when treating large scale systems.

Contrary to verification techniques that aim at proving, after a PLC program has been more or less correctly designed by an expert, that control laws are safe, automatic synthesis methods aim at systematically generating control laws which guarantee by construction the respect of expected safety properties. The avoidance of human errors during the design of controllers is one of the main reasons for which synthesis is a very important subject of research in the field of discrete event systems (DES) since the end of 80's.

Most part of recent works in this area are still based onto the Supervisory Control Theory (SCT) [6] and are aiming for the synthesis of a *supervisor*, and not directly to the *controller* of an automated system. Furthermore, the use of state models (Finite Automata, Petri Nets, etc.) and their composition for the construction of the models of the plant and of

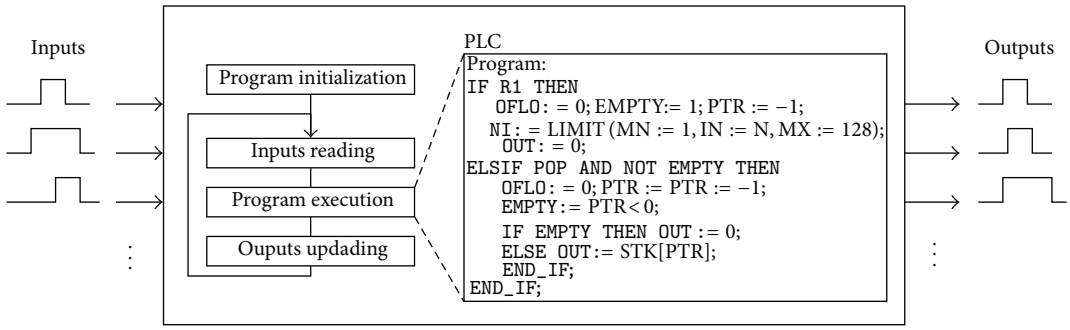


Figure 1: PLC basic principle.

the specifications generates a complexity which remains problematic for the synthesis of a supervisor for complex systems [7]. It is therefore interesting to explore other ways for performing synthesis, such as algebraic approaches. In previous works, we proposed a method specifically developed to get the control laws that can be directly implemented into the controller [8]. We have chosen to synthesize these control laws under the form of recurrent Boolean equations because of the wide possibilities they offer for the formalization of safety requirements and for implementation.

Nevertheless, whatever is the used synthesis method, one of the weak links of the automatic generation of the control laws is the step of formal transcription by the designer (within state models or algebraic expressions) of the informal requirements and safety properties the controller has to satisfy. In the case of SCT, some authors have proposed more or less generic approaches for the construction of the models of the plant [9] or of the specifications [10]. But in any case, the hypothesis that requirements can be inconsistent has never been taken into account. Unfortunately in the framework of industrial collaborations we have been able to verify that it is always the case. In this paper we show how, in consideration of specific hypotheses, it is possible to install a correction loop for helping the designer to formalize these requirements and so improving the synthesis method robustness to the lack of precision of the specifications.

This paper is organized as follows. Some basics of algebraic synthesis given in Sections 2 and 3 recall the main steps of our method. Section 4 presents the mathematical framework of our approach and new results that allow us to accept inconsistencies in specifications. The strategy we developed for making the synthesis more robust to the lack of consistency of the specifications is described in Section 5, thanks to a case study.

2. Problem Statement

Figure 2 proposes a generic representation of a DES whose controller has p Boolean inputs (u_i), q Boolean outputs (y_j), and r Boolean state variables (x_l). Plant and controller are connected through a closed loop exchanging inputs and outputs signals. The state variables, needed for expressing sequential behaviors of the controller, are represented by internal variables.

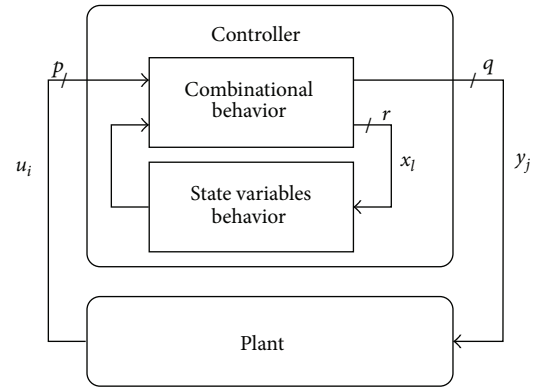


Figure 2: A sequential DES.

The algebraic modeling of the control laws of the controller necessitates the definition of $(q+r)$ switching functions of $(p+r)$ variables. Even if this representation is very compact (the r Boolean state variables allow the representation of 2^r different states), the construction by hands of these switching functions is a very tedious and error-prone task [11]; the controller of Figure 2 admits 2^p inputs combinations can send 2^q outputs combinations and can express $(2^{2^{(p+r)}})^{(q+r)}$ sequential behaviors. That is the reason why algebraic modeling approaches have been replaced by methods based on state models since the middle of 50's [12, 13]. Nevertheless, thanks to recent mathematical results obtained onto Boolean algebras [14, 15], the automatic algebraic synthesis of switching functions is now possible.

In [16] an interesting approach for the systematic construction of a reactive program from its formal specification is proposed. In this work, the program synthesis is considered as a theorem proving activity. A program with input x and output y, specified by the formula $\varphi(x, y)$, is constructed as a byproduct of proving the theorem $(\forall x)(\exists y)\varphi(x, y)$. The specification $\varphi(x, y)$ characterizes the expected relation between the input x and the output y computed by the program. This approach is based on the observation that

the formula $(\forall x)(\exists y)\varphi(x, y)$ is equivalent to the second-order formula $(\exists f)(\forall x)\varphi(x, f(x))$, stating the existence of a function f, such that $\varphi(x, f(x))$ holds for every x.

This approach provides a conceptual framework for the rigorous derivation of a program from its formal specification. It has also been used to synthesize specifications under the form of finite automata from their linear temporal logic (LTL) description [17].

The core of our approach is based on this strategy: we aim at deducing the $(q+r)$ switching functions of $(p+r)$ variables which define the behavior of the controller from a formula $\varphi(u_i[k], x_l[k-1], y_j[k], x_l[k])$ that holds for every k, every $u_i[k]$, and every $x_l[k-1]$.

To cope with combinatorial explosion, switching functions will be handled through a symbolic representation (and not their truth-tables which contain $2^{(p+r)}$ Boolean values). Each input u_i (resp., output y_j) of the controller will be represented by a switching function U_i (resp., 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 $[k-1]$).

According to this representation, the synthesis of control laws of a logical system from its specification can now be transformed into the search of the solutions to the mathematical problem as follows:

$$(\forall U_i)\left(\forall {}_pX_l\right)\left(\exists Y_j\right)\left(\exists X_l\right)\varphi\left(U_i, {}_pX_l, Y_j, X_l\right), \quad (1)$$

where $(U_i, {}_pX_l, Y_j, X_l)$ are $(p+q+2r)$ switching functions of $(p+r)$ variables.

3. Overview of Our Method

The input data of the proposed method (Figure 3) are unformal 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, etc.) or imposed standards.

All the steps of our synthesis method are implemented into a prototype software tool developed in Python (Case studies are available online: http://www.lurpa.ens-cachan.fr/-226050.kjsp). The first step is the formalization of requirements within an algebraic description; examples are given in Section 5.2. Requirements expressed with a state model can directly be translated into recurrent Boolean equations, thanks to the algorithm proposed by Machado et al. [18]. In case where the knowhow 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 [19].

The second step consists in checking the consistency of the set of requirements by symbolic calculation. The sufficient condition for checking this consistency has been given in [20] but no strategy has been proposed for coping with potential inconsistencies. In this paper we show that thanks to new theorems the causes of these inconsistencies can be pointed out. It is then possible for the designer to fix priority rules between the concerned requirements that will allow finding, if exist, solutions despite inconsistencies.

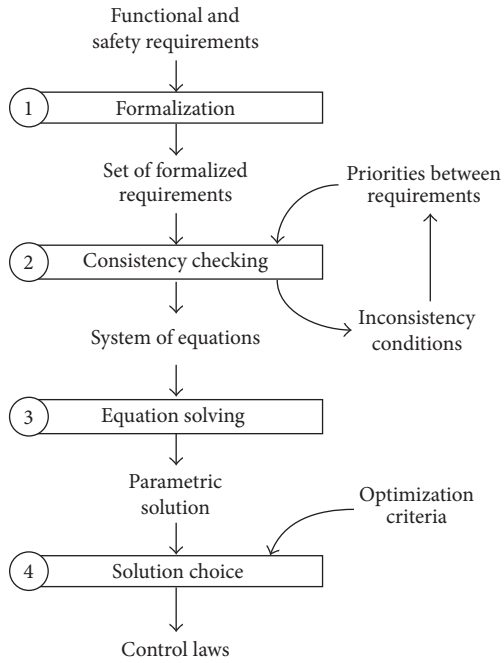


Figure 3: The algebraic synthesis method step by step.

The core of the method is the third step, which consists in the synthesis of the control laws. This step is performed by solving the system of equations which represents the set of consistent requirements. The mathematical results we have obtained (Theorem 12 given in Section 4.3), allow finding a parametric expression of the set of solutions.

In the fourth step of the method, a particular solution has to be chosen among the set of solutions. For that, a specific value of each parameter of the general solution has to be fixed. In a previous work [19], we showed how well chosen heuristics can be used for fixing these parameters. In this paper, we show that the choice of a particular solution among the set of solutions can be expressed as an optimization problem. We propose new theorems that allow calculating the maximum and the minimum of a Boolean formula, and we show how optimal solutions can be automatically found. For ergonomic reasons, the synthesized control laws can finally be displayed under the form of a finite automaton [21].

After the mathematical background of the method has been recalled, we are going to show how, in consideration of specific hypotheses, the second step of the method can be improved by a correction loop helping the designer to formalize the requirements and so improving the robustness of our synthesis method to the lack of precision of the specifications. The strategy to find an optimal solution according to given criteria will be also presented.

4. Mathematical Foundations

This section is composed of five subsections. Sections 4.1 and 4.2 recall some classical results about Boolean algebras

and the Boolean algebra of n-variable switching functions. Section 4.3 presents how to solve Boolean equations. Sections 4.4 and 4.5 present specific results obtained for the algebraic synthesis of control laws.

4.1. Boolean Algebra: Typical Feature

Definition 1 (Boolean algebra). (Definition 15.5 of [22]) Let $\mathscr{B}$ 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 $\overline{\ }$. Then $(\mathscr{B}, +, \cdot, \overline{\ }, 0, 1)$ is called a Boolean algebra if the following conditions are satisfied for all $x, y, z \in \mathscr{B}$:

$$\begin{aligned}
&\text{Commutative Laws:}\\
&\quad x + y = y + x\\
&\quad x \cdot y = y \cdot x\\
&\text{Distributive Laws:}\\
&\quad x \cdot (y + z) = (x \cdot y) + (x \cdot z)\\
&\quad x + (y \cdot z) = (x + y) \cdot (x + z)\\
&\text{Identity Laws:}\\
&\quad x + 0 = x\\
&\quad x \cdot 1 = x\\
&\text{Inverse Laws:}\\
&\quad x + \overline{x} = 1\\
&\quad x \cdot \overline{x} = 0\\
&\quad 0 \neq 1.
\end{aligned} \tag{2}$$

Many Boolean algebras could be defined. The most known are the two-element Boolean algebra: $(\{0, 1\}, \vee, \wedge, \neg, 0, 1)$ and the algebra of classes (set of subsets of a set S): $(2^S, \cup, \cap, \overline{\ }, \emptyset, S)$.

Definition 2 (Boolean formula). (From Section 3.6 of [15]) A *Boolean formula* (or a Boolean expression) on $\mathscr{B}$ is any formula which represents a combination of members of $\mathscr{B}$ by the operations $+$, $\cdot$, or $\overline{\ }$.

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

Theorem 3 (Boole's expansion of a Boolean formula). *Let $(\alpha_1, \ldots, \alpha_n)$ be n members of $\mathscr{B} \setminus \{0, 1\}$. Any Boolean Formula $\mathscr{F}(\alpha_1, \ldots, \alpha_n)$ can be expanded as*

$$\mathscr{F}(\alpha_1, \ldots, \alpha_n) = \mathscr{F}_0(\alpha_2, \ldots, \alpha_n) \cdot \overline{\alpha_1} + \mathscr{F}_1(\alpha_2, \ldots, \alpha_n) \cdot \alpha_1, \tag{3}$$

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

$$\begin{aligned}
\mathscr{F}_0(\alpha_2, \ldots, \alpha_n) &= \mathscr{F}(\alpha_1, \ldots, \alpha_n)\big|_{\alpha_1 \leftarrow 0} = \mathscr{F}(0, \alpha_2, \ldots, \alpha_n)\\
\mathscr{F}_1(\alpha_2, \ldots, \alpha_n) &= \mathscr{F}(\alpha_1, \ldots, \alpha_n)\big|_{\alpha_1 \leftarrow 1} = \mathscr{F}(1, \alpha_2, \ldots, \alpha_n).
\end{aligned} \tag{4}$$

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 $\mathscr{B}$. This relation is called *Inclusion-Relation* in [15].

Definition 4 (Inclusion-Relation). (Definition 15.6 of [22].) If $x, y \in \mathscr{B}$, define $x \leq y$ if and only if $x \cdot y = x$.

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 $\mathscr{B}$ (Theorem 15.4 of [22]).

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

Remark 5. For the algebra of classes $(2^S, \cup, \cap, \overline{\ }, \emptyset, S)$, the Inclusion-Relation is the well-known relation $\subseteq$ and we have: $x \subseteq y \Leftrightarrow x \cap y = x$.

Theorem 6 (reduction of a set of relations). *(Theorem 5.3.1 of [15].) Any set of simultaneously asserted relations built with the members $(\alpha_1, \ldots, \alpha_n)$ of $\mathscr{B}$ can be reduced to a single equivalent relation such as: $\mathscr{F}(\alpha_1, \ldots, \alpha_n) = 0$.*

To obtain this equivalent relation, it is necessary

(i) to rewrite each equality according to

$$\begin{aligned}
&\mathscr{F}_1(\alpha_1, \ldots, \alpha_n) = \mathscr{F}_2(\alpha_1, \ldots, \alpha_n)\\
&\Longleftrightarrow \mathscr{F}_1(\alpha_1, \ldots, \alpha_n) \cdot \overline{\mathscr{F}_2(\alpha_1, \ldots, \alpha_n)}\\
&\qquad + \overline{\mathscr{F}_1(\alpha_1, \ldots, \alpha_n)} \cdot \mathscr{F}_2(\alpha_1, \ldots, \alpha_n) = 0,
\end{aligned} \tag{5}$$

(ii) to rewrite each inclusion according to

$$\begin{aligned}
&\mathscr{F}_1(\alpha_1, \ldots, \alpha_n) \leq \mathscr{F}_2(\alpha_1, \ldots, \alpha_n)\\
&\Longleftrightarrow \mathscr{F}_1(\alpha_1, \ldots, \alpha_n) \cdot \overline{\mathscr{F}_2(\alpha_1, \ldots, \alpha_n)} = 0,
\end{aligned} \tag{6}$$

(iii) to group together rewritten equalities as follows:

$$\begin{aligned}
&\begin{cases} \mathscr{F}_1(\alpha_1, \ldots, \alpha_n) = 0\\ \mathscr{F}_2(\alpha_1, \ldots, \alpha_n) = 0 \end{cases}\\
&\quad \Longleftrightarrow \mathscr{F}_1(\alpha_1, \ldots, \alpha_n) + \mathscr{F}_2(\alpha_1, \ldots, \alpha_n) = 0.
\end{aligned} \tag{7}$$

4.2. 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 is denoted by ${}_b b_i$. The set of the two Boolean values ${}_b 0$ and ${}_b 1$ is denoted by $B = \{{}_b 0, {}_b 1\}$.

Definition 7 (N-variable switching functions). (From Section 3.11 of [15].) An n-variable *switching function* is a mapping of the form

$$\begin{aligned}
&f: B^n \longrightarrow B\\
&({}_b b_1, \ldots, {}_b b_n) \longmapsto f({}_b b_1, \ldots, {}_b b_n)\\
&\qquad \text{where } B = \{{}_b 0, {}_b 1\}.
\end{aligned} \tag{8}$$

The domain of a n-variable switching function has 2^n elements and the codomain has 2 elements; hence, there are 2^{2^n} n-variable switching functions. Let $F_n(B)$ be the set of the 2^{2^n} 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* (f^i_{Proj}). These functions are defined as follows:

$$\begin{aligned} 0\colon B^n &\longrightarrow B \\ ({}_b b_1, \ldots, {}_b b_n) &\longmapsto {}_b 0 \\ 1\colon B^n &\longrightarrow B \\ ({}_b b_1, \ldots, {}_b b_n) &\longmapsto {}_b 1 \\ f^i_{\text{Proj}}\colon B^n &\longrightarrow B \\ ({}_b b_1, \ldots, {}_b b_n) &\longmapsto {}_b b_i, \end{aligned} \tag{9}$$

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

$$\begin{aligned} \text{Op. } + : F_n(B)^2 &\longrightarrow F_n(B) \\ (f, g) &\longmapsto f + g \\ \text{Op. } \cdot : F_n(B)^2 &\longrightarrow F_n(B) \\ (f, g) &\longmapsto f \cdot g \\ \text{Op. } \bar{} : F_n(B) &\longrightarrow F_n(B) \\ f &\longmapsto \overline{f}, \end{aligned} \tag{10}$$

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

$$\begin{aligned} &(f+g)({}_b b_1, \ldots, {}_b b_n) \\ &\quad = f({}_b b_1, \ldots, {}_b b_n) \vee g({}_b b_1, \ldots, {}_b b_n), \\ &(f \cdot g)({}_b b_1, \ldots, {}_b b_n) \\ &\quad = f({}_b b_1, \ldots, {}_b b_n) \wedge g({}_b b_1, \ldots, {}_b b_n), \\ &\overline{f}({}_b b_1, \ldots, {}_b b_n) = \neg f({}_b b_1, \ldots, {}_b b_n). \end{aligned} \tag{11}$$

$(F_n(B), +, \cdot, \bar{}, 0, 1)$ is a Boolean algebra [22]. Then, it is possible to write a Boolean formula of n-variable switching functions and relations between Boolean formula of n-variable switching functions. In the case of n-variable switching functions, relations Equality and Inclusion can also be presented as follows:

(i) f and g are equal $(f = g)$ if and only if the columns of the truth-tables of f, g are exactly the same, that is, $\forall ({}_b b_1, \ldots, {}_b b_n) \in B^n$, $f({}_b b_1, \ldots, {}_b b_n) = g({}_b b_1, \ldots, {}_b b_n)$.

(ii) f is included into g $(f \leq g)$ if and only if the value of g is always ${}_b 1$ when the value of f is ${}_b 1$, that is, $\forall ({}_b b_1, \ldots, {}_b b_n) \in B^n$, $[f({}_b b_1, \ldots, {}_b b_n) = {}_b 0]$, or $[g({}_b b_1, \ldots, {}_b b_n) = {}_b 1]$.

Remark 8. Each n-variable switching function can be expressed as a composition of $(f^1_{\text{Proj}}, \ldots, f^n_{\text{Proj}}, 0, 1)$ by operations $+, \cdot$ and $\bar{}$.

Therefore, the Boolean algebra $(F_n(B), +, \cdot, \bar{}, 0, 1)$ is a mathematical framework which allows composing and to comparing switching functions. Thanks to the results presented in the next subsection, this framework allows also solving Boolean equations systems of switching functions.

4.3. Solutions of Boolean Equations over Boolean Algebra $F_n(B)$. In [15], Brown explains that many problems in the application of Boolean algebra may be reduced to solving an equation of the form

$$f(X) = 0, \tag{12}$$

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 [14]). These works concern essentially the two-element Boolean algebra $(\{{}_b 0, {}_b 1\}, \vee, \wedge, \neg, {}_b 0, {}_b 1)$.

In our case, 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. Theorems presented in this section permit to solve any system of Boolean equations as it exists in a canonic form of a Boolean system of k unknowns and we are able to calculate solutions for this form.

4.3.1. Canonic Form of a Boolean System of k Unknowns over Boolean Algebra $F_n(B)$. Consider the Boolean algebra of n-variable switching functions $(F_n(B), +, \cdot, \bar{}, 0, 1)$.

(i) Let $(f^1_{\text{Proj}}, \ldots, f^n_{\text{Proj}})$ be the n projection-functions of $F_n(B)$.

(ii) Let $(x_1, \ldots, x_k)$ be k elements of $F_n(B)$ considered as *unknowns*.

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

Theorem 9 (reduction of a set of relations between n-variable switching functions). *Any set of simultaneously asserted relations of switching functions can be reduced to a single equivalent relation such as*

$$\mathcal{F}(X_k, Proj) = 0. \tag{13}$$

This theorem comes from Theorem 6.

In order to be 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 $a \in \{0,1\}$, x^a is defined by

$$x^0 = \overline{x}, \qquad x^1 = x. \tag{14}$$

This notation is extended to vectors as follows: for $X_k = (x_1, \ldots, x_k) \in F_n(B)^k$ and $A_k = (a_1, \ldots, a_k) \in \{0,1\}^k$, $X_k^{A_k}$ is defined by

$$X_k^{A_k} = \prod_{i=1}^{i=k} x_i^{a_i} = x_i^{a_1} \cdot \cdots \cdot x_k^{a_k}. \tag{15}$$

Theorem 10 (canonic form of a Boolean equation). *Any Boolean equation $Eq(X_k, Proj) = 0$ can be expressed within the canonic form*

$$\sum_{A_k \in \{0,1\}^k} Eq\left(A_k, Proj\right) \cdot X_k^{A_k} = 0, \tag{16}$$

where $Eq(A_k, Proj)$ (with $A_k \in \{0,1\}^k$) are the 2^k discriminants of $Eq(X_k, Proj) = 0$ according to X_k (the term of "discriminant" comes from [15]).

This canonic form is obtained by expanding $\mathrm{Eq}(X_k, \mathrm{Proj})$ according to the k unknowns $(x_1, \ldots, x_k)$. For example, we have

$$\begin{aligned}
\mathrm{Eq}\left(x, \mathrm{Proj}\right) &= \mathrm{Eq}\left(0, \mathrm{Proj}\right) \cdot \overline{x} + \mathrm{Eq}\left(1, \mathrm{Proj}\right) \cdot x, \\
\mathrm{Eq}\left(x_1, x_2, \mathrm{Proj}\right) &= \mathrm{Eq}\left(0, 0, \mathrm{Proj}\right) \cdot \overline{x_1} \cdot \overline{x_2} \\
&\quad + \mathrm{Eq}\left(0, 1, \mathrm{Proj}\right) \cdot \overline{x_1} \cdot x_2 \\
&\quad + \mathrm{Eq}\left(1, 0, \mathrm{Proj}\right) \cdot x_1 \cdot \overline{x_2} \\
&\quad + \mathrm{Eq}\left(1, 1, \mathrm{Proj}\right) \cdot x_1 \cdot x_2.
\end{aligned} \tag{17}$$

4.3.2. Solution of a Single-Unknown Equation over $F_n(B)$. The following theorem has initially been demonstrated for the two-element Boolean algebra [14]. A generalization for all Boolean algebras can be found in [15], but no detailed demonstration is given. A new formalization of this theorem and its full demonstration are given below.

Theorem 11 (solution of a single-unknown equation). *The Boolean equation over $F_n(B)$*

$$Eq\left(x, Proj\right) = 0, \tag{18}$$

for which the canonic form is

$$Eq\left(0, Proj\right) \cdot \overline{x} + Eq\left(1, Proj\right) \cdot x = 0, \tag{19}$$

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

$$Eq\left(0, Proj\right) \cdot Eq\left(1, Proj\right) = 0. \tag{20}$$

In this case, a general form of the solutions is

$$x = Eq\left(0, Proj\right) + p \cdot \overline{Eq\left(1, Proj\right)}, \tag{21}$$

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

This solution can also be expressed as

$$\begin{aligned}
x &= \overline{Eq\left(1, Proj\right)} \cdot \left(Eq\left(0, Proj\right) + p\right) \\
&= \overline{p} \cdot Eq\left(0, Proj\right) + p \cdot \overline{Eq\left(1, Proj\right)}.
\end{aligned} \tag{22}$$

Proof. This theorem can be proved in four steps as follows:

(a) Equation (18) is consistent if and only if (20) is satisfied;

(b) Equation (21) is a solution of (18) if (20) is satisfied;

(c) each solution of (18) can be expressed as (21);

(d) if (20) is satisfied, the three parametric forms proposed are equivalent.

Step (a) can be proved as follows: Equation (20) is a sufficient condition for (18) to admit solutions since $x = \mathrm{Eq}(0, \mathrm{Proj})$ is an obvious solution of (18). Equation (20) is also a necessary condition as if (18) admits a solution, then (18) can be also expressed thanks to the consensus theorem as $\mathrm{Eq}(0, \mathrm{Proj}) \cdot \overline{x} + \mathrm{Eq}(1, \mathrm{Proj}) \cdot x + \mathrm{Eq}(0, \mathrm{Proj}) \cdot \mathrm{Eq}(1, \mathrm{Proj}) = 0$ and we have necessarily $\mathrm{Eq}(0, \mathrm{Proj}) \cdot \mathrm{Eq}(1, \mathrm{Proj}) = 0$.

To prove Step (b), it is sufficient to substitute the expression for x from (21) into (18) and to use (20) as follows:

$$\begin{aligned}
&\mathrm{Eq}\left(0, \mathrm{Proj}\right) \cdot \overline{x} + \mathrm{Eq}\left(1, \mathrm{Proj}\right) \cdot x \\
&\quad = \mathrm{Eq}\left(0, \mathrm{Proj}\right) \cdot \overline{\left(\mathrm{Eq}\left(0, \mathrm{Proj}\right) + p \cdot \overline{\mathrm{Eq}(1, \mathrm{Proj})}\right)} \\
&\qquad + \mathrm{Eq}\left(1, \mathrm{Proj}\right) \cdot \left(\mathrm{Eq}\left(0, \mathrm{Proj}\right) + p \cdot \overline{\mathrm{Eq}(1, \mathrm{Proj})}\right) \\
&\quad = \mathrm{Eq}\left(0, \mathrm{Proj}\right) \cdot \overline{\mathrm{Eq}\left(0, \mathrm{Proj}\right)} \cdot \overline{\left(p \cdot \overline{\mathrm{Eq}\left(1, \mathrm{Proj}\right)}\right)} \\
&\qquad + \mathrm{Eq}\left(0, \mathrm{Proj}\right) \cdot \mathrm{Eq}\left(1, \mathrm{Proj}\right) \\
&\qquad + p \cdot \mathrm{Eq}\left(1, \mathrm{Proj}\right) \cdot \overline{\mathrm{Eq}\left(1, \mathrm{Proj}\right)} \\
&\quad = 0 + 0 + 0 = 0.
\end{aligned} \tag{23}$$

To prove Step (c), it is sufficient to find one element p of $F_n(B)$ for each solution for x of (18). Let us consider p defined by "$p = \overline{\mathrm{Eq}(0, \mathrm{Proj})} \cdot \overline{\mathrm{Eq}(1, \mathrm{Proj})} \cdot x$" where x is a solution to (18). Then we have

$$\begin{aligned}
&\left\{\begin{array}{c}
\mathrm{Eq}\left(0, \mathrm{Proj}\right) \cdot \mathrm{Eq}\left(1, \mathrm{Proj}\right) = 0 \\
\mathrm{Eq}\left(0, \mathrm{Proj}\right) \cdot \overline{x} + \mathrm{Eq}\left(1, \mathrm{Proj}\right) \cdot x = 0 \\
p = \overline{\mathrm{Eq}(0, \mathrm{Proj})} \cdot \overline{\mathrm{Eq}(1, \mathrm{Proj})} \cdot x
\end{array}\right. \\
&\qquad \Longrightarrow x = \mathrm{Eq}\left(0, \mathrm{Proj}\right) + p \cdot \overline{\mathrm{Eq}(1, \mathrm{Proj})}
\end{aligned} \tag{24}$$

as

$$\begin{aligned}
x = 1 \cdot x &= \Big(\mathrm{Eq}\left(0, \mathrm{Proj}\right) + \mathrm{Eq}\left(1, \mathrm{Proj}\right) \\
&\qquad + \overline{\mathrm{Eq}\left(0, \mathrm{Proj}\right)} \cdot \overline{\mathrm{Eq}\left(1, \mathrm{Proj}\right)}\Big) \cdot x \\
&= \mathrm{Eq}\left(0, \mathrm{Proj}\right) \cdot x + \mathrm{Eq}\left(1, \mathrm{Proj}\right) \cdot x \\
&\quad + \overline{\mathrm{Eq}\left(0, \mathrm{Proj}\right)} \cdot \overline{\mathrm{Eq}\left(1, \mathrm{Proj}\right)} \cdot x \\
&= \mathrm{Eq}\left(0, \mathrm{Proj}\right) \cdot x + 0 + \overline{\mathrm{Eq}\left(1, \mathrm{Proj}\right)} \\
&\quad \cdot \left(\overline{\mathrm{Eq}\left(0, \mathrm{Proj}\right)} \cdot \overline{\mathrm{Eq}\left(1, \mathrm{Proj}\right)} \cdot x\right) \\
&\qquad \text{as } \mathrm{Eq}\left(1, \mathrm{Proj}\right) \cdot x = 0
\end{aligned}$$

$$
\begin{aligned}
&= \mathrm{Eq}(0,\mathrm{Proj})\cdot x + \mathrm{Eq}(0,\mathrm{Proj})\cdot \overline{x} \\
&\quad + \overline{\mathrm{Eq}(1,\mathrm{Proj})}\cdot p \quad \text{as } \mathrm{Eq}(0,\mathrm{Proj})\cdot \overline{x} = 0 \\
&= \mathrm{Eq}(0,\mathrm{Proj})\cdot (x+\overline{x}) + p\cdot \overline{\mathrm{Eq}(1,\mathrm{Proj})} \\
&= \mathrm{Eq}(0,\mathrm{Proj}) + p\cdot \overline{\mathrm{Eq}(1,\mathrm{Proj})}.
\end{aligned} \tag{25}
$$

To prove Step (d), it is sufficient to rewrite (21) in the two other forms by using (20) as follows:

$$
\begin{aligned}
x &= 1\cdot \mathrm{Eq}(0,\mathrm{Proj}) + p\cdot \overline{\mathrm{Eq}(1,\mathrm{Proj})} \\
&= \left(\mathrm{Eq}(1,\mathrm{Proj}) + \overline{\mathrm{Eq}(1,\mathrm{Proj})}\right)\cdot \mathrm{Eq}(0,\mathrm{Proj}) \\
&\quad + p\cdot \overline{\mathrm{Eq}(1,\mathrm{Proj})} \\
&= \mathrm{Eq}(0,\mathrm{Proj})\cdot \mathrm{Eq}(1,\mathrm{Proj}) + (\mathrm{Eq}(0,\mathrm{Proj}) + p) \\
&\quad \cdot \overline{\mathrm{Eq}(1,\mathrm{Proj})} \\
&= 0 + \overline{\mathrm{Eq}(1,\mathrm{Proj})}\cdot (\mathrm{Eq}(0,\mathrm{Proj}) + p) \\
&= \overline{\mathrm{Eq}(1,\mathrm{Proj})}\cdot (\mathrm{Eq}(0,\mathrm{Proj}) + p), \\
x &= 1\cdot \mathrm{Eq}(0,\mathrm{Proj}) + p\cdot \overline{\mathrm{Eq}(1,\mathrm{Proj})} \\
&= (p+\overline{p})\cdot \mathrm{Eq}(0,\mathrm{Proj}) + p\cdot \overline{\mathrm{Eq}(1,\mathrm{Proj})} \\
&= \overline{p}\cdot \mathrm{Eq}(0,\mathrm{Proj}) + p\cdot \left(\mathrm{Eq}(0,\mathrm{Proj}) + \overline{\mathrm{Eq}(1,\mathrm{Proj})}\right) \\
&= \overline{p}\cdot \mathrm{Eq}(0,\mathrm{Proj}) \\
&\quad + p\cdot \left(\mathrm{Eq}(0,\mathrm{Proj})\cdot \mathrm{Eq}(1,\mathrm{Proj}) + \overline{\mathrm{Eq}(1,\mathrm{Proj})}\right) \\
&= \overline{p}\cdot \mathrm{Eq}(0,\mathrm{Proj}) + p\cdot \left(0 + \overline{\mathrm{Eq}(1,\mathrm{Proj})}\right) \\
&= \overline{p}\cdot \mathrm{Eq}(0,\mathrm{Proj}) + p\cdot \overline{\mathrm{Eq}(1,\mathrm{Proj})}.
\end{aligned} \tag{26}
$$

□

4.3.3. Solution of k-Unknown Equations over $F_n(B)$. The global result presented in the following theorem can be found in [14] or [15]. However, in these works, the solution is not expressed with a parametric form, but with intervals only. The formulation presented in this paper is more adapted to symbolic computation and is mandatory for practice optimization.

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 12 (solution of a k-unknown equation). *The Boolean equation over $F_n(B)$*

$$Eq_0(X_k, Proj) = 0 \tag{27}$$

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

$$\prod_{A_k\in\{0,1\}^k} Eq_0(A_k, Proj) = 0. \tag{28}$$

If (28) *is satisfied,* (27) *admits one or more k-tuple solutions $(S(x_1),\ldots,S(x_k))$ such each component $S(x_i)$ is defined by*

$$
\begin{aligned}
S(x_i) = &\prod_{A_{k-i}\in\{0,1\}^{k-i}} Eq_{i-1}(0, A_{k-i}, Proj) \\
&+ p_i\cdot \overline{\prod_{A_{k-i}\in\{0,1\}^{k-i}} Eq_{i-1}(1, A_{k-i}, Proj)},
\end{aligned} \tag{29}
$$

with

(i) $Eq_i(x_{i+1},\ldots,x_k,Proj) = Eq_{i-1}(x_i,x_{i+1},\ldots,x_k,Proj)|_{x_i\leftarrow S(x_i)}$

(ii) *p_i is an arbitrary parameter, that is, a freely-chosen member of $F_n(B)$.*

The full demonstration of this theorem cannot be given in this paper because of lack of space (a full demonstration by mathematical induction can be found in [8]). A description of the different steps of the proof and the detail of the principal steps are given below.

Proof (elements of Proof). Equation (27) can be solved by applying Theorems 3 and 11 k times according to the unknowns x_k to x_1 as follows.

According to Theorem 3, (27) is equivalent to

$$\mathrm{Eq}_0(X_{k-1},0,\mathrm{Proj})\cdot \overline{x_k} + \mathrm{Eq}_0(X_{k-1},1,\mathrm{Proj})\cdot x_k = 0. \tag{30}$$

According to Theorem 11, (30) admits solutions in x_k if and only if

$$\mathrm{Eq}_0(X_{k-1},0,\mathrm{Proj})\cdot \mathrm{Eq}_0(X_{k-1},1,\mathrm{Proj}) = 0. \tag{31}$$

Equation (31) is an equation with $(k-1)$ unknowns. Each term of (31) can be expanded according to x_{k-1} and (31) can be written in the form

$$
\begin{aligned}
&\left(\mathrm{Eq}_0(X_{k-2},0,0,\mathrm{Proj})\cdot \mathrm{Eq}_0(X_{k-2},0,1,\mathrm{Proj})\right)\cdot \overline{x_{k-1}} \\
&+ \left(\mathrm{Eq}_0(X_{k-2},1,0,\mathrm{Proj})\cdot \mathrm{Eq}_0(X_{k-2},1,1,\mathrm{Proj})\right)\cdot x_{k-1} = 0.
\end{aligned} \tag{32}
$$

According to Theorem 11, (32) admits solutions in x_{k-1} if and only if

$$\prod_{A_2\in\{0,1\}^2} \mathrm{Eq}_0(X_{k-2},A_2,\mathrm{Proj}) = 0. \tag{33}$$

Equation (33) is an equation with $(k-2)$ unknowns. Each term of (33) can be expanded according to x_{k-2} and (33) can be written in the form

$$\left(\prod_{A_2\in\{0,1\}^2} \mathrm{Eq}_0\left(X_{k-3},0,A_2,\mathrm{Proj}\right)\right)\cdot\overline{x_{k-2}} + \left(\prod_{A_2\in\{0,1\}^2} \mathrm{Eq}_0\left(X_{k-3},1,A_2,\mathrm{Proj}\right)\right)\cdot x_{k-2}=0. \tag{34}$$

In the end, we obtain an equation of only one unknown x_1 defined by

$$\left(\prod_{A_{k-1}\in\{0,1\}^{k-1}} \mathrm{Eq}_0\left(0,A_{k-1},\mathrm{Proj}\right)\right)\cdot\overline{x_1} + \left(\prod_{A_{k-1}\in\{0,1\}^{k-1}} \mathrm{Eq}_0\left(1,A_{k-1},\mathrm{Proj}\right)\right)\cdot x_1=0. \tag{35}$$

According to Theorem 11, (35) admits solutions if and only if

$$\prod_{A_k\in\{0,1\}^k} \mathrm{Eq}_0\left(A_k,\mathrm{Proj}\right)=0. \tag{36}$$

When (36) is satisfied, the k equations for x_1 to x_k admit solutions. Equation (27) is then coherent and admits solutions.

When (36) is satisfied, solutions of (35) for x_1 are

$$S\left(x_1\right)=\prod_{A_{k-1}\in\{0,1\}^{k-1}} \mathrm{Eq}_0\left(0,A_{k-1},\mathrm{Proj}\right) + p_1\cdot\overline{\prod_{A_{k-1}\in\{0,1\}^{k-1}} \mathrm{Eq}_0\left(1,A_{k-1},\mathrm{Proj}\right)}. \tag{37}$$

After substituting $S(x_1)$ for x_1 into (27), we obtain a new equation $\mathrm{Eq}_1(x_2,\ldots,x_k,\mathrm{Proj})=0$ involving the $(k-1)$ unknowns $(x_2,\ldots,x_k)$, where

$$\mathrm{Eq}_1\left(x_2,\ldots,x_k,\mathrm{Proj}\right)=\mathrm{Eq}_0\left(x_1,x_2,\ldots,x_k,\mathrm{Proj}\right)|_{x_1\leftarrow S(x_1)}. \tag{38}$$

By applying the previous procedure, we can obtain $S(x_2)$ and $\mathrm{Eq}_2(x_3,\ldots,x_k,\mathrm{Proj})$. Then, it suffices to apply this procedure again $(k-2)$ times to obtain successively solutions $S(x_3)$ to $S(x_k)$. □

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 forms.

4.3.4. Partial Conclusions. Thanks to theorems presented above, 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. In practice, due to the complexity of systems to be designed, proposed set of simultaneously asserted relations is generally inconsistent [23]. To simplify the work of the designer, we have proved complementary theorems to improve the robustness of our method to the lack of precision of the specifications (Section 4.4).

When several solutions exist, the comparison of solutions according to a given criterion can be envisaged since the Boolean algebra $F_n(B)$ is equipped with a partial order. To simplify the work of the designer too, we have developed a method to calculate the best solutions according to one or several criteria (Section 4.5).

4.4. Theorems to Cope with Inconsistencies of Specifications. In practice, it is very difficult for a designer to specify the whole requirements of a complex system without inconsistencies. It is the reason why requirements given by the designer are often declared as inconsistent according to Theorem 12. Since the inconsistency condition is a Boolean formula, it is possible to use it for the detection of the origin of inconsistencies. Two cases have to be considered as follows:

(i) Several requirements cannot be simultaneously respected. In this case, a hierarchy between requirements can be proposed in order to find a solution. The requirements which have the lower priority have to be corrected for becoming consistent with the requirements which have the higher priority. This strategy is based on Theorem 14.

(ii) The detected inconsistency refers to specific combinations of projection-functions for which the designer knows that they are impossible blocking the synthesis process, it is necessary to introduce new assumptions and to use Theorem 13.

Theorem 13 (solution of a Boolean equation according to an assumption among the projection-functions). *The following problem*

$$\begin{aligned} &\textit{Equationtosolve:}\\ &\quad Eq_0\left(X_k,Proj\right)=0\\ &\textit{Assumptions:}\\ &\quad \mathscr{A}\left(Proj\right)=0 \end{aligned} \tag{39}$$

admits the same solutions as the following equation:

$$Eq_0\left(X_k,Proj\right)\le\mathscr{A}\left(Proj\right). \tag{40}$$

Proof. According to $\mathscr{A}(\text{Proj}) = 0$, $\text{Eq}_0(X_k, \text{Proj}) = 0$ can be rewritten as

$$\begin{aligned}
&\begin{cases} \text{Eq}_0(X_k, \text{Proj}) = 0 \\ \mathscr{A}(\text{Proj}) = 0 \end{cases} \\
&\quad \Longleftrightarrow \mathscr{A}(\text{Proj}) + \text{Eq}_0(X_k, \text{Proj}) = 0 \\
&\quad \Longleftrightarrow \mathscr{A}(\text{Proj}) + \overline{\mathscr{A}(\text{Proj})} \cdot \text{Eq}_0(X_k, \text{Proj}) = 0 \\
&\quad \Longleftrightarrow \begin{cases} \overline{\mathscr{A}(\text{Proj})} \cdot \text{Eq}_0(X_k, \text{Proj}) = 0 \\ \mathscr{A}(\text{Proj}) = 0 \end{cases} \\
&\quad \Longleftrightarrow \begin{cases} \text{Eq}_0(X_k, \text{Proj}) \le \mathscr{A}(\text{Proj}) \\ \mathscr{A}(\text{Proj}) = 0. \end{cases}
\end{aligned} \tag{41}$$

Equation $\overline{\mathscr{A}(\text{Proj})} \cdot \text{Eq}_0(X_k, \text{Proj}) = 0$ is consistent if and only if the following condition is true (Theorem 12):

$$\overline{\mathscr{A}(\text{Proj})} \cdot \prod_{A_k \in \{0,1\}^k} \text{Eq}_0(A_k, \text{Proj}) = 0. \tag{42}$$

By construction, this new condition is the subset of the initial condition ($\prod_{A_k \in \{0,1\}^k} \text{Eq}_0(A_k, \text{Proj}) = 0$) for which the proposed assumption is satisfied. All the other terms have been removed.

If (42) is satisfied, (40) admits one or more k-tuple solutions where each component $S(x_i)$ is defined by

$$\begin{aligned}
S(x_i) = {} & \overline{\mathscr{A}(\text{Proj})} \\
& \cdot \left(\prod_{A_{k-i} \in \{0,1\}^{k-i}} \text{Eq}_{i-1}(0, A_{k-i}, \text{Proj}) \right. \\
& \left. + p_i \cdot \overline{\prod_{A_{k-i} \in \{0,1\}^{k-i}} \text{Eq}_{i-1}(1, A_{k-i}, \text{Proj})} \right) \\
& + \mathscr{A}(\text{Proj}) \cdot p_i.
\end{aligned} \tag{43}$$

As $\mathscr{A}(\text{Proj}) = 0$, $S(x_i)$ can also be expressed as

$$\begin{aligned}
S(x_i) = {} & \prod_{A_{k-i} \in \{0,1\}^{k-i}} \text{Eq}_{i-1}(0, A_{k-i}, \text{Proj}) + p_i \\
& \cdot \overline{\prod_{A_{k-i} \in \{0,1\}^{k-i}} \text{Eq}_{i-1}(1, A_{k-i}, \text{Proj})}.
\end{aligned} \tag{44}$$

When $\mathscr{A}(\text{Proj}) = 0$ is satisfied, the solutions of (40) are also solution to $\text{Eq}_0(X_k, \text{Proj}) = 0$. □

Theorem 14 (Solution of a Boolean equation system according to a priority rule between requirements). *The following problem*

$$\begin{aligned}
&\textit{Equations system to solve:} \\
&\begin{array}{ll} HR & \mathscr{F}_{\mathscr{H}}(X_k, Proj) = 0 \\ LR & \mathscr{F}_{\mathscr{L}}(X_k, Proj) = 0 \\ OR & \mathscr{F}_{\mathscr{O}}(X_k, Proj) = 0 \end{array} \\
&\textit{Priority rule between requirements:} \\
&\quad HR \gg LR,
\end{aligned} \tag{45}$$

where

(i) $\mathscr{F}_{\mathscr{H}}(X_k, Proj) = 0$ *is the formal expression of the requirements which have the higher priority (HR);*

(ii) $\mathscr{F}_{\mathscr{L}}(X_k, Proj) = 0$ *is the formal expression of the requirements which have the lower priority (LR);*

(iii) $\mathscr{F}_{\mathscr{O}}(X_k, Proj) = 0$ *is the formal expression of the others requirements (OR);*

(iv) $HR \gg LR$ *is the priority rule between inconsistent requirements,*

admits the same solutions as the system of equations as follows:

$$\begin{aligned}
&\mathscr{F}_{\mathscr{H}}(X_k, Proj) = 0 \\
&\mathscr{F}_{\mathscr{L}}(X_k, Proj) \le \mathscr{I}(Proj) \\
&\mathscr{F}_{\mathscr{O}}(X_k, Proj) = 0,
\end{aligned} \tag{46}$$

where $\mathscr{I}(Proj)$ is the inconsistency condition between requirements "HR" and "LR":

$$\mathscr{I}(Proj) = \prod_{A_k \in \{0,1\}^k} \left(\mathscr{F}_{\mathscr{H}}(A_k, Proj) + \mathscr{F}_{\mathscr{L}}(A_k, Proj) \right). \tag{47}$$

Proof. Thanks to Theorem 12, the inconsistency condition $\mathscr{I}(\text{Proj})$ between requirements "HR" and "LR" can be found by solving equation $\mathscr{F}_{\mathscr{H}}(X_k, \text{Proj}) + \mathscr{F}_{\mathscr{L}}(X_k, \text{Proj}) = 0$. We have

$$\mathscr{I}(\text{Proj}) = \prod_{A_k \in \{0,1\}^k} \left(\mathscr{F}_{\mathscr{H}}(A_k, \text{Proj}) + \mathscr{F}_{\mathscr{L}}(A_k, \text{Proj}) \right). \tag{48}$$

To remove the inconsistency between requirements "HR" and "LR" according to the priority rule "HR ≫ LR", it is necessary to restrict the range of requirement "LR" to the part for which there is no inconsistency, that is, $\mathscr{I}(\text{Proj}) = 0$. That is the case, when $\mathscr{F}_{\mathscr{L}}(X_k, \text{Proj}) = 0$ is replaced by $\mathscr{F}_{\mathscr{L}}(X_k, \text{Proj}) \le \mathscr{I}(\text{Proj})$.

Thanks to Theorem 12, (49) admits always one or more k-tuple solutions and it is impossible to find a less restrictive condition over requirement "LR".

$$\begin{aligned}
&\mathscr{F}_{\mathscr{H}}(X_k, \text{Proj}) = 0 \\
&\mathscr{F}_{\mathscr{L}}(X_k, \text{Proj}) \le \mathscr{I}(\text{Proj}).
\end{aligned} \tag{49}$$

□

4.5. Optimal Solutions of Boolean Equations over $F_n(B)$. The goal of this step 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 ($\mathrm{Eq}(X_k, \mathrm{Proj}) = 0$) of Boolean functions but also which maximize (or minimize) a Boolean formula of these Boolean functions ($\mathscr{F}_C(X_k, \mathrm{Proj})$) 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 which cannot be used as $F_n(B)$ is only provided by a partial order; two particular solutions cannot always be ordered between themselves.

Nevertheless, it is possible to obtain the researched parametric form of the k-tuples thanks to the following results.

(i) When an equation between Boolean functions has one or more solution tuples in $F_n(B)$, every Boolean formula onto 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.

(ii) Every Boolean formula expressed as a composition of projection-functions of $F_n(B)$ and free parameters of $F_n(B)$ has a unique maximum and a 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

$$\begin{aligned} &\text{Problem to solve:} \\ &\quad \mathrm{Eq}\left(X_k, \mathrm{Proj}\right) = 0 \\ &\text{Optimization Criterion:} \\ &\quad \text{Maximization of } \mathscr{F}_{\mathrm{C}}\left(X_k, \mathrm{Proj}\right), \end{aligned} \tag{50}$$

into a 2-equation system to solve

$$\begin{aligned} &\mathrm{Eq}\left(X_k, \mathrm{Proj}\right) = 0 \\ &\mathscr{F}_{\mathrm{C}}\left(X_k, \mathrm{Proj}\right) = \operatorname*{Max}_{\{X_k \mid \mathrm{Eq}(X_k, \mathrm{Proj})=0\}} \left(\mathscr{F}_{\mathrm{C}}\left(X_k, \mathrm{Proj}\right)\right). \end{aligned} \tag{51}$$

4.5.1. Extrema of a Boolean Formula according to Freely Chosen Members of $F_n(B)$. Considering the Boolean algebra of n-variable switching functions $(F_n(B), +, \cdot, \bar{}, 0, 1)$,

(i) let $(f^1_{\mathrm{Proj}}, \ldots, f^n_{\mathrm{Proj}})$ be the *n projection-functions* of $F_n(B)$;

(ii) let $(p_1, \ldots, p_k)$ be k elements of $F_n(B)$ considered as *freely chosen members*. Let "P_k" be the corresponding vector.

Any formula $\mathscr{F}(P_k, \mathrm{Proj})$ 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 $\mathscr{F}(P_k, \mathrm{Proj})$ admits a minimal element and a maximal element.

Theorem 15 (minimum and maximum of a Boolean formula). *Any formula $\mathscr{F}(P_k, Proj)$ for which P_k are freely chosen members of $F_n(B)$ admits a minimum and a maximum defined as follows:*

$$\begin{aligned} \operatorname*{Min}_{P_k \in F_n(B)^k} \left(\mathscr{F}\left(P_k, Proj\right)\right) &= \prod_{A_k \in \{0,1\}^k} \mathscr{F}\left(A_k, Proj\right) \\ \operatorname*{Max}_{P_k \in F_n(B)^k} \left(\mathscr{F}\left(P_k, Proj\right)\right) &= \sum_{A_k \in \{0,1\}^k} \mathscr{F}\left(A_k, Proj\right), \end{aligned} \tag{52}$$

Proof. To prove this theorem, it is necessary to establish that

(1) $\prod_{A_k \in \{0,1\}^k} \mathscr{F}(A_k, \mathrm{Proj})$ is a lower bound of $\mathscr{F}(P_k, \mathrm{Proj})$;

(2) It exists at least one specific combination of P_k for which $\mathscr{F}(P_k, \mathrm{Proj}) = \prod_{A_k \in \{0,1\}^k} \mathscr{F}(A_k, \mathrm{Proj})$;

(3) $\sum_{A_k \in \{0,1\}^k} \mathscr{F}(A_k, \mathrm{Proj})$ is an upper bound of $\mathscr{F}(P_k, \mathrm{Proj})$;

(4) It exists at least one specific combination of P_k for which $\mathscr{F}(P_k, \mathrm{Proj}) = \sum_{A_k \in \{0,1\}^k} \mathscr{F}(A_k, \mathrm{Proj})$.

Details of this proof can be found in [24]. □

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

(i) let $(f^1_{\mathrm{Proj}}, \ldots, f^n_{\mathrm{Proj}})$ be the *n projection-functions* of $F_n(B)$. Let "Proj" be the corresponding vector;

(ii) Let $(x_1, \ldots, x_k)$ be k elements of $F_n(B)$ considered as *unknowns*. Let "X_k" be the corresponding vector;

(iii) Let $(p_1, \ldots, p_k)$ be k elements of $F_n(B)$ considered as *freely chosen members*. Let "P_k" be the corresponding vector.;

(iv) Let $\mathrm{Eq}(X_k, \mathrm{Proj}) = 0$ be the Boolean equation to solve;

(v) Let $\mathscr{F}_{\mathrm{C}}(X_k, \mathrm{Proj})$ 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 $\mathrm{Eq}(X_k, \mathrm{Proj}) = 0$ according to a given optimization criterion $\mathscr{F}_{\mathrm{C}}(X_k, \mathrm{Proj})$ is composed of five steps as follows.

(i) The first step is to establish the parametric form of the k-tuple solution to $\mathrm{Eq}(X_k, \mathrm{Proj}) = 0$ only, thanks to Theorem 12.

(ii) The second step is to establish the parametric form of the given optimization criterion $\mathscr{F}_{\mathrm{C}}(X_k, \mathrm{Proj})$ by substituting $S(x_i)$ for x_i. Let $\mathscr{F}_{\mathrm{SC}}(P_k, \mathrm{Proj})$ be the result of this substitution.

(iii) The third step is to calculate the extremum of $\mathscr{F}_{\mathrm{SC}}(P_k, \mathrm{Proj})$ according to Theorem 15. Let $\mathscr{F}_{\mathrm{EC}}(\mathrm{Proj})$ be the Boolean formula of this extremum.

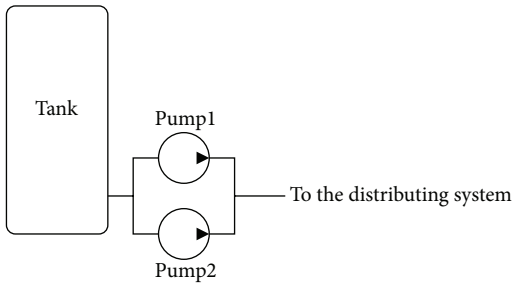


Figure 4: Structure of the water supply system.

(iv) The fourth step is to replace the given criterion by the equivalent relation

$$\mathscr{F}_C\left(X_k, \text{Proj}\right) = \mathscr{F}_{EC}\left(\text{Proj}\right). \tag{53}$$

(v) The fifth step is to establish the parametric form of the k-tuple solution of the equivalent problem

$$\begin{aligned} &\text{Eq}\left(X_k, \text{Proj}\right) = 0 \\ &\mathscr{F}_{\text{Crit}}\left(X_k, \text{Proj}\right) = \mathscr{F}_{\text{ExtCrit}}\left(\text{Proj}\right). \end{aligned} \tag{54}$$

4.5.3. Partial Conclusions. Thanks to 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.

(i) When several criteria are treated simultaneously, the optimization problem can admit no solution. That is the case when the given criteria are antagonist.

(ii) When several criteria are treated sequentially, the obtained solutions satisfy the criteria with a given priority order. An example of optimization with several criteria treated sequentially is presented in the next section.

5. Algebraic Synthesis of Logical Controllers with Optimization Criteria and Incoherent Requirements

5.1. Control System Specifications. The studied system is the controller of a water supply system composed of two pumps which are working in redundancy (Figure 4). The water distribution is made when it is necessary according to the possible failures of elements (the pumps and the distributing system).

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

(i) The two pumps never operate simultaneously.

(ii) A pump cannot operate if it is out of order.

(iii) When a global failure is detected, no pump can operate.

(iv) Pumps can operate if and only if a water distribution request is present.

(v) Priority is given according to "pr" (pump1 has priority when "pr" is true).

(vi) In order to reduce the wear of the pumps, it is necessary to restrict the number of starting of the pumps.

5.1.1. Inputs and Outputs of the Controller. The Boolean inputs and outputs of this controller are given in Figure 5(a). 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 corresponding pump and "gf" indicates a global failure of the installation. The values o or 1 of input "Pr" decide which pump has priority.

5.1.2. Control Laws to Synthetize. Our approach does not allow identifying 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 2 state variables, one for each output. According to this choice, 2 7-variable switching functions ($P1$ and $P2$) have to be synthesized (Figure 5(b)). They represent the unknowns of our problem. For this case study, the 7 projection-functions of $F_7(B)$ are therefore as follows.

(i) 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:

$$\begin{aligned} &\text{Rq}: B^7 \longrightarrow B \\ &\quad \left(\text{rq}\left[k\right], \ldots, \text{p2}\left[k-1\right]\right) \longmapsto \text{rq}\left[k\right]. \end{aligned} \tag{55}$$

(ii) The 2 switching functions ($_pP1$ and $_pP2$) which characterize the previous behavior of the state variables of the controller and are defined as follows:

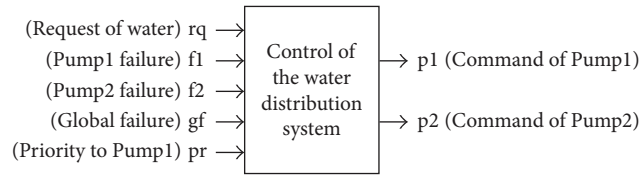
$$\begin{aligned} &{}_pP1: B^7 \longrightarrow B \\ &\quad \left(\text{rq}\left[k\right], \ldots, \text{p2}\left[k-1\right]\right) \longmapsto \text{p1}\left[k-1\right]. \end{aligned} \tag{56}$$

5.2. Algebraic Formalization of Requirements. The complete formalization of the behavior of the water distribution system is given in Figure 5(c). In order to illustrate the power of expression of relations Equality and Inclusion, several examples (generic assertions and equivalent formal relations illustrated in the case study) are given hereafter. 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.

(i) Pump1 and Pump2 never operate simultaneously: $P1 \cdot P2 = 0$;

(ii) If Pump1 operates, Pump2 cannot operate: $P1 \leq \overline{P2}$;

(a) *Inputs and Outputs of the Controller*



(b) *General form of the Expected Control Laws*

$$\mathrm{p1}\,[k] = \mathrm{P1}\left(\mathrm{rq}\,[k], \mathrm{f1}\,[k], \mathrm{f2}\,[k]\,\mathrm{gf}\,[k], \mathrm{pr}\,[k], \mathrm{p1}\,[k-1], \mathrm{p2}\,[k-1]\right)$$

$$\mathrm{p2}\,[k] = \mathrm{P2}\left(\mathrm{rq}\,[k], \mathrm{f1}\,[k], \mathrm{f2}\,[k]\,\mathrm{gf}\,[k], \mathrm{pr}\,[k], \mathrm{p1}\,[k-1], \mathrm{p2}\,[k-1]\right)$$

$$\mathrm{p1}\,[0] = {}_b0 \qquad \mathrm{p2}\,[0] = {}_b0$$

(c) *Formal Specification*

Requirements:

R1 $\mathrm{P1} \cdot \mathrm{P2} = 0$ (* *The two pumps never operate simultaneously.* *)
R2 $\mathrm{F1} \leq \overline{\mathrm{P1}}$ (* *Pump 1 cannot operate if it is out of order (F1).* *)
R3 $\mathrm{F2} \leq \overline{\mathrm{P2}}$ (* *Pump2 cannot operate if it is out of order (F2).* *)
R4 $\mathrm{GF} \leq (\overline{\mathrm{P1}} \cdot \overline{\mathrm{P2}})$ (* *When a global failure is detected (GF), no pump can operate.* *)
R5 $(\mathrm{P1} + \mathrm{P2}) \leq \mathrm{Rq}$ (* *It is necessary to have are quest for pumps operate.* *)
R6 $\mathrm{Rq} \leq (\mathrm{P1} + \mathrm{P2})$ (* *It is sufficient to have a request for pumps operate.* *)

Priority rules:

R4 $\gg$ R6 (* *Failure requirements has priority on a functional requirement.* *)
{R2, R3} $\gg$ R6 (* *Failure requirements has priority on a functional requirement.* *)

Optimization criteria:

(1) Minimization of: $((\mathrm{P1} \cdot \overline{{}_pP1}) + (\mathrm{P2} \cdot \overline{{}_pP2}))$ (* *Minimization of the possibility to start a pump.* *)
(2) Maximization of: $((\mathrm{Pr} \cdot \mathrm{P1}) + (\overline{\mathrm{Pr}} \cdot \mathrm{P2}))$ (* *Maximization of the priority order between the two pumps.* *)

(d) *Solution obtained by symbolic calculation*

$$\mathrm{P1} = \mathrm{Rq} \cdot \overline{\mathrm{GF}} \cdot \overline{\mathrm{F1}} \cdot \left(\mathrm{F2} + \mathrm{Pr} \cdot \left({}_pP1 + \overline{{}_pP2}\right) + {}_p\mathrm{P}\,1 \cdot \overline{{}_pP2}\right)$$

$$\mathrm{P2} = \mathrm{Rq} \cdot \overline{\mathrm{GF}} \cdot \overline{\mathrm{F2}} \cdot \left(\mathrm{F1} + \overline{\mathrm{Pr}} \cdot \left({}_pP2 + \overline{{}_pP1}\right) + {}_p\mathrm{P}\,2 \cdot \overline{{}_pP1}\right)$$

(e) *Control laws of the water distribution system*

$$\mathrm{p1}\,[k] = \mathrm{rq}\,[k] \wedge \neg\mathrm{gf}\,[k] \wedge \neg\mathrm{f1}\,[k] \wedge \left(\mathrm{f2}\,[k] \vee \mathrm{pr}\,[k] \wedge \left(\mathrm{p1}\,[k-1] \vee \neg\mathrm{p2}\,[k-1]\right) \vee \mathrm{p1}\,[k-1] \wedge \neg\mathrm{p2}\,[k-1]\right)$$

$$\mathrm{p2}\,[k] = \mathrm{rq}\,[k] \wedge \neg\mathrm{gf}\,[k] \wedge \neg\mathrm{f1}\,[k] \wedge \left(\mathrm{f1}\,[k] \vee \mathrm{pr}\,[k] \wedge \left(\mathrm{p2}\,[k-1] \vee \neg\mathrm{p1}\,[k-1]\right) \vee \mathrm{p2}\,[k-1] \wedge \neg\mathrm{p1}\,[k-1]\right)$$

$$\mathrm{p1}\,[0] = {}_b0 \qquad \mathrm{p2}\,[0] = {}_b0$$

Figure 5: Details of the case study.

(iii) It is necessary to have a request for pumps operate: $(P1 + P2) \leq \text{Rq}$;

(iv) It is sufficient to have a request for pumps operate: $\text{Rq} \leq (P1 + P2)$;

(v) When Pump1 is failed, it is sufficient to have a request for Pump2 operate: $\text{F1} \cdot \text{Rq} \leq P2$;

(vi) When Pump1 is failed, it is necessary to have a request for Pump2 operate: $\text{F1} \cdot P2 \leq \text{Rq}$.

It is possible to prove that some of these formal expressions are equivalent (e.g., the first two). 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.

As $P1$ and ${}_pP1$ 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 as follows.

(i) It is necessary to have a request to start pump1: $(P1 \cdot \overline{{}_pP1}) \leq \text{Rq}$.

(ii) When pump1 operates, it is sufficient to have a global failure to stop pump1: $({}_pP1 \cdot \text{GF}) \leq (\overline{P1} \cdot {}_pP1)$.

5.3. Synthesis Process. In traditional design methods, the design procedure of a logic controller is not a linear process, but an iterative one converging to an acceptable solution. At the beginning of the design, requirements are neither complete nor without errors. Most often, new requirements are added during the search of solutions, and others are corrected. This complementary information is given by the designer after analysis of the partial solutions he found or when inconsistencies have been detected. If we do not make the hypothesis that the specifications are complete and consistent, designing a controller with a synthesis technique is also an iterative process in which the designer plays an important role.

5.3.1. Analysis of Requirements. For this case study, we choose to start with requirements R1 to R6. For this subset of requirements, the result given by your software tool was the following inconsistency condition: $\mathcal{F} = \text{Rq} \cdot \text{GF} + \text{Rq} \cdot \text{F1} \cdot \text{F2}$.

Since requirements are declared inconsistent, we have to give complementary information to precise our specification. By analyzing each term of this formula, it is possible to detect the origin of the inconsistency:

(i) $\text{Rq} \cdot \text{GF}$: what happens if we have simultaneously a request and a global failure? We consider that requirement R4 is more important than requirement R6 (R4 $\gg$ R6) as no pump can operate for this configuration.

(ii) $\text{Rq} \cdot \text{F1} \cdot \text{F2}$: what happens if we have simultaneously a request and a failure of each pump? We consider that requirements R2 and R3 are more important than requirement R6 ({R2, R3} $\gg$ R6).

With these priority rules, all the requirements are now coherent and the set of all the solutions can be computed.

5.3.2. Optimal Solutions. For choosing a control law of the water supply system among this set of possible solutions, we will now take into account the given optimization criteria. The first criterion aims at minimizing the number of starting of each pump in order to reduce its wear. The second criterion aims at maximizing the use of the pump indicated by the value of parameter Pr. The method we propose allows proving that proposed criteria cannot be treated simultaneously since they are antagonist (to strictly the priority use of the pump fixed by parameter Pr, it is necessary to permute pumps when Pr changes of value, implying a supplementary start of a pump). Details can be found in [25].

All the priorities rules and optimization criteria used for this case study are given in Figure 5(c). The solution we obtain is proposed in Figure 5(d).

5.3.3. Implementing Control Laws. The synthesized control laws presented in Figure 5(e) have been obtained by translating the expression of the two unknowns according to the projection-functions into relations between recurrent Boolean equations. These control laws can be automatically translated in the syntax of the ladder diagram language [1] before being implemented into a PLC. The code is composed of only 4 rungs (Figure 6).

The synthesized control laws can be given under the form of an automatically built input/output automaton with guarded transitions [21] (Figure 7).

6. Discussion

In our approach, the synthesis of control laws is based on the symbolic calculation, a prototype software tool has been developed to avoid tedious calculus and to aid the designer during the different steps of the synthesis. This tool (that can be obtained on request by the authors) performs all the computations required for inconsistencies detection between requirements and for control laws generation. In this tool, all the Boolean formulas are stored in the form of reduced ordered binary decision diagrams, which allows efficient calculations. For example, the computations for synthesizing a controller for the water supply system that we developed above have been made in less than 10 ms onto a classical laptop.

Our approach has been tested on several studies cases (some of them are available online: http://www.lurpa.ens-cachan.fr/-226050.kjsp). The feedbacks of these experiences allowed us to identify some of its limits and its possibilities; the most important are given below.

We have first to recall that our method can only be used for binary systems (systems whose inputs and outputs of their controller are Boolean values). Nevertheless, in practice many systems, like manufacturing systems, transport systems, and so on, are fully or partially binary.

In our opinion the main advantage of our approach is that, contrary to traditional engineering approaches, the synthesized control laws are not depending on designer's skill or of his correct interpretation of the system requirements. On the other hand, the quality of the synthesis results highly

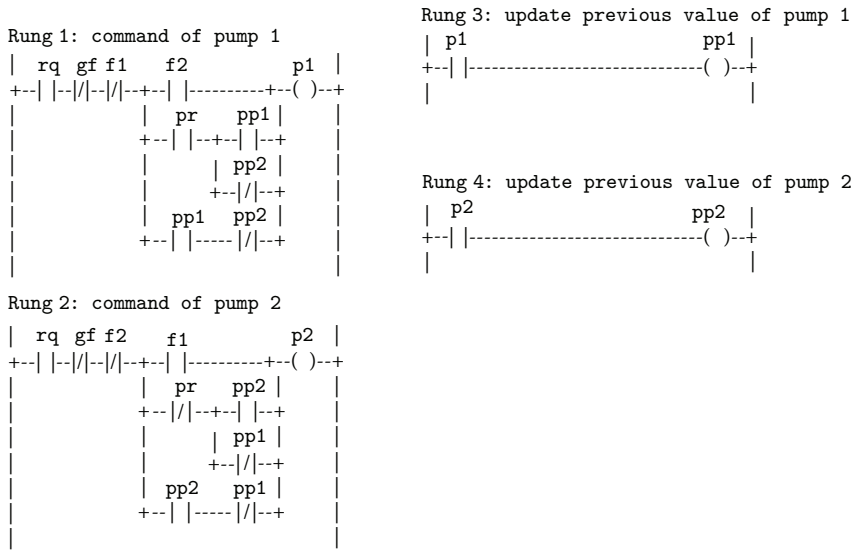


Figure 6: Ladder diagram of the code to implement into the PLC.

Table 1: Futures concerning a same case study.

	Formal requirements	Synthesized controller	PLC program (structured text)
Supervisory Control Theory	Plant behavior: 11 finite automata (481 states and 1330 transitions) Specifications: 11 finite automata	Finite automaton of 45 states and 70 transitions	130 lines
Algebraic synthesis	8 equations and 2 priority rules	2 6-variable switching functions	4 lines

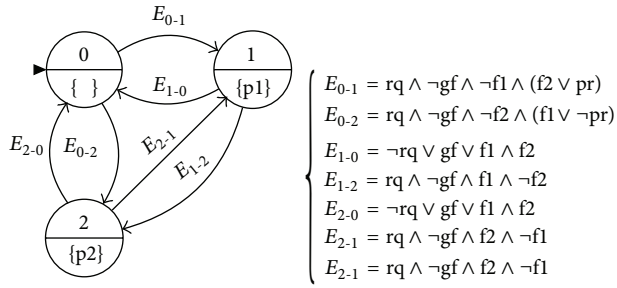


Figure 7: State model of the obtained control law.

depends on the relevance of the requirements proposed by the designer. This step of formalization, by the designer, of the informal requirements of the system to be controlled is the Achilles heel of all synthesis methods, including the Supervisory Control Theory (SCT), and cannot be automated.

The objective comparison of our approach with other synthesis methods, and more especially with SCT, is very difficult because the models used and the theoretical basics are very different. Nevertheless, we tested both approaches on same study cases. One of them, the control of an automatic parking gate, has been published in [26]. The results obtained in this case are summarized in Table 1.

Furthermore, one may note that the supervisor that is synthesized by SCT is optimal in the sense where it is the most permissive; that is, the one that reduces the less the plant behavior in order to force it to respect the specifications. As shown in this paper our method allows to cope with inconsistencies in specifications, what is not possible with SCT, and also allows to find optimal controllers by choosing different optimization criteria (most permissive, most restrictive, most safe controller, etc.).

7. Conclusion

Many research works in the field of DES aim at formalizing steps of the systems life cycle. Since 20 years, significant progresses have been obtained for the synthesis, verification, performance evaluation, and diagnosis of DESs. Nevertheless, one of the common difficulties of these works is the translation of informal expression of the knowledge of a system into formal requirements. Few works have paid attention to this important task which is very error prone. In this paper, we proposed an iterative process that allows coping with inconsistencies of the requirements during the synthesis of the controller. The framework in which we proposed this approach is an algebraic synthesis method. Since the problem is located in the frontier between formal and informal, intervention of the designer is necessary. Nevertheless, we have shown that this intervention can be guided by the results of the formal method provides.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

References

[1] International Electrotechnical Commission, *IEC 61131-3, IEC 61131-3 Standard: Programmable Controllers-Part 3: Programming Languages*, International Electrotechnical Commission, 2nd edition, 2003.

[2] G. Frey and L. Litz, "Formal methods in PLC programming," in *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, vol. 4, pp. 2431–2436, October 2000.

[3] B. Berard, M. Bidoit, A. Finkel et al., *Systems and Software Verification: Model-Checking Techniques and Tools*, Springer, New York, NY, USA, 1st edition, 1999.

[4] H. Bel Mokadem, B. Bérard, V. Gourcuff, O. De Smet, and J. Roussel, "Verification of a timed multitask system with UPPAAL," *IEEE Transactions on Automation Science and Engineering*, vol. 7, no. 4, pp. 921–932, 2010.

[5] J. L. Boulanger, Ed., *Industrial Use of Formal Methods: Formal Verification (ISTE)*, Wiley-ISTE, New York, NY, USA, 2012.

[6] P. J. G. Ramadge and W. M. Wonham, "Control of discrete event systems," *Proceedings of the IEEE*, vol. 77, no. 1, pp. 81–98, 1989.

[7] P. Gohari and W. M. Wonham, "On the complexity of supervisory control design in the RW framework," *IEEE Transactions on Systems, Man, and Cybernetics B*, vol. 30, no. 5, pp. 643–652, 2000.

[8] Y. Hietter, *Synthése algèbrique de lois de commande pour les systemes a évènements discrets logiques [Ph.D. thesis]*, ENS Cachan, Cachan, France, 2009.

[9] H.-M. Hanisch, A. Lueder, and J. Thieme, "Modular plant modeling technique and related controller synthesis problems," in *Proceedings of the 1998 IEEE International Conference on Systems, Man, and Cybernetics*, pp. 686–691, October 1998.

[10] J.-M. Roussel and A. Giua, "Designing dependable logic controllers using the supervisory control theory," in *Proceedings of the 16th IFACWorld Congress*, cDRom paper 4427, p. 6, Praha, Czech Republic, 2005.

[11] D. A. Huffman, "The synthesis of sequential switching circuits," *Journal of the Franklin Institute*, vol. 257, no. 3-4, pp. 161–303, 1954.

[12] G. H. Mealy, "A method for synthesizing sequential circuits," *Bell System Technical Journal*, vol. 34, no. 5, pp. 1045–1079, 1955.

[13] E. F. Moore, "Gedanken-experiments on sequential machines," in *Automata Studies*, pp. 129–153, Princeton University Press, Princeton, NJ, USA, 1956.

[14] S. Rudeanu, *Lattice Functions and Equations (Discrete Mathematics and Theoretical Computer Science)*, Springer, New York, NY, USA, 2001.

[15] F. M. Brown, *Boolean Reasoning: the Logic of Boolean Equations*, Dover, Mineola, NY, USA, 2003.

[16] A. Pnueli and R. Rosner, "On the synthesis of a reactive module," in *Proceedings of the 16th ACM symposium on Principles of programming languages (POPL '89)*, pp. 179–190, ACM, New York, NY, USA, 1989.

[17] E. Filiot, N. Jin, and J. Raskin, "Antichains and compositional algorithms for LTL synthesis," *Formal Methods in System Design*, vol. 39, no. 3, pp. 261–296, 2011.

[18] J. Machado, B. Denis, J. J. Lesage, J. M. Faure, and J. Fereira, "Logic controllers dependability verification using a plant model," in *Proceedings of the 3rd IFAC Workshop on Discrete-Event System Design (DESDes '06)*, pp. 37–42, Rydzyna, Poland, 2006.

[19] Y. Hietter, J.-M. Roussel, and J.-L. Lesage, "Algebraic synthesis of transition conditions of a state model," in *Proceedings of the 9th International Workshop on Discrete Event Systems (WODES '08)*, pp. 187–192, Göteborg, Sweden, May 2008.

[20] Y. Hietter, J.-M. Roussel, and J. J. Lesage, "Algebraic synthesis of dependable logic controllers," in *Proceedings of the 17th World Congress, International Federation of Automatic Control (IFAC '08)*, pp. 4132–4137, Seoul, South Korea, July 2008.

[21] A. Guignard, *Symbolic generation of the automaton representing an algebraic description of a logic system [M.S. thesis]*, ENS Cachan, Cachan, France, 2011.

[22] R. P. Grimaldi, *Discrete and Combinatorial Mathematics: An Applied Introduction*, Addison-Wesley Longman, Boston, Mass, USA, 5th edition, 2004.

[23] J.-M. Roussel and J.-J. Lesage, "Algebraic synthesis of logical controllers despite inconsistencies in specifications," in *Proceeding of the 11th International Workshop on Discrete Event Systems (WODES '12)*, pp. 307–314, Guadalajara, Mexico, 2012.

[24] H. Leroux, *Algebraic Synthesis of Logical Controllers with Optimization Criteria*, ENS Cachan, Cachan, France, 2011.

[25] H. Leroux and J. -M. Roussel, "Algebraic synthesis of logical controllers with optimization criteria," in *Proceedings of the 6th International Workshop on Verification and Evaluation of Computer and Communication Systems (VECOS '12)*, pp. 103–114, Paris, France, 2012.

[26] M. Cantarelli and J. 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)*, pp. 200–205, Göteborg, Sweden, May 2008.