IFAC

# Algebraic synthesis of dependable logic controllers

**Yann Hietter** * **Jean-Marc Roussel** * **Jean-Jacques Lesage** *

* *LURPA, ENS Cachan, UniverSud, 61 Av. du President Wilson*
*F-94235 Cachan Cedex (email: {yann.hietter, jean-marc.roussel,*
*jean-jacques.lesage}@lurpa.ens-cachan.fr)*

**Abstract:** This paper presents an algebraic method to synthesize control laws for logical system controllers. The starting point is a set of functional and dependable requirements expressed with algebraic relations or state models. We propose to synthesize control laws by solving a Boolean equation which represents all the requirements. The mathematical results that we have obtained allow to establish the exact form of the solutions if this equation has solutions.
The first step of this method is the formalization of each requirement with Boolean relations between Boolean functions. Under this formulation, the requirements can be assembled and their coherence can be analyzed. This step consists in verifying if the Boolean equation, which represents all the requirements, has solutions. The third step is the synthesis of the control laws by solving this equation. At the end of this step, a parametric formulation of all the possible solutions is given. The fourth step of the method is the choice of a particular solution. This choice is made by the designer from heuristics. This method is illustrated with an example.

Keywords: Dependable system, Controller synthesis, Algebraic approaches, Boolean algebra, logic controllers

## 1. INTRODUCTION

Logic controllers are used in a very large number of systems such as embedded systems, transport systems, power plants or production systems. Several components of our daily lives and of the economy at large thereby rely upon the successful operations of these controllers. This explains why the dependability of logic controllers is a major concern for control engineers.

To improve the dependability of logic controllers, it is important to ensure that no flaw due to a misinterpretation of the requirements or to any other reason has been introduced during design. A logic controller can in fact only be referred to as dependable if its behaviour fulfills the application requirements and must therefore not include design errors leading to non-functional or hazardous behaviours.

To reach this goal, many formal methods have been proposed during the last twenty years. They aim at detecting flaws once the controller has been designed or avoiding flaws during design (Faure and Lesage [2001]). The first class of approaches consists in letting the control system designer develops control laws based on the requirements contained in the set of specifications, and then in automatically analysing a formal representation of these control laws. Such an analysis may be carried out by using formal verification techniques such as model-checking (De Smet and Rossi [2002], Klein et al. [2003]) or theorem proving (Roussel and Faure [2002]). The second class of approaches aims at deducing the control laws from the requirements and the dependable properties expected, without any involvement of the designer (or at least by limiting his involvement to a strict minimum). The method presented herein belongs to this second class of approaches that come within the synthesis.

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 is finally developed in section 4 thanks to an illustrative example.

## 2. OBJECTIVES

To design controllers, several research teams propose to exploit the Supervisory Control Theory (SCT) defined by Ramadge and Wonham [1987]. 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 (Kumar et al. [1991]; Ramadge and Wonham [1987]) automatically generate an optimal supervisor which 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, Pétin et al. [2007]) 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 possibilities 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 manually. An example of plant model with this abstraction level is presented in Roussel and Guia [2005]. This abstraction level is often necessary to obtain control laws for dependable logic controllers.

The method that we propose (figure 1) is specially developed to synthesize dependable logic controllers. The starting point of this method is functional and dependable requirements which are given by the designer. The first step is their formalization with state models or algebraic models. Examples of this formalization are given in section 4.

The second step consists in analyzing all the requirements to control if the whole is coherent. In our case, this step is made by symbolic calculus on an algebraic formulation of them. If inconsistencies exist, the conditions to have these inconsistencies are given to the designer to help him modify given requirements. If coherence is proved, the formal description of the requirements becomes a system of equations in which the control laws searched are the unknowns.

The third step is the synthesis of the control laws. This step is made by solving the system of equations which represents coherent requirements. The mathematical results that we have obtained allow to establish an exact symbolic form for solutions. The set of solutions is described with a parametric expression of the generic solution.

The last 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 requirements proposed if the solutions synthesized do not satisfy him. If our method can prove that the requirements are coherent, only the designer could judge whether the requirements are completed by analyzing the solutions synthesized.

In traditional design methods, the design procedure of a logic controller is an interactive process which converges to an acceptable solution. At the beginning, requirements are not complete. New requirements are introduced during the search of solutions. These complementary elements are given by the designer after an analysis of 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. For that reason, to be used in an industrial context, an automatic synthesis method must take into account the three following aspects:

- A designer will agree to use a tool which automatically finds a solution only if the effort made to express the requirements is less important than the effort made to find a solution manually.
- The solution automatically proposed must be readable by the designer to allow its analysis.
- The designer can obtain the solution for a specific system by adapting the solution obtained for a previous system studied if both systems are similar.

To take into account these aspects, our method accepts to start from requirements completed with a partial solution. In this case, the automatic synthesis is used to complete this solution. This possibility simplifies the designer's work by reducing the number of requirements to give. As the solutions synthesized are built from the partial solution, their analysis is simplier. The case study presented in section 4 is made in this context.
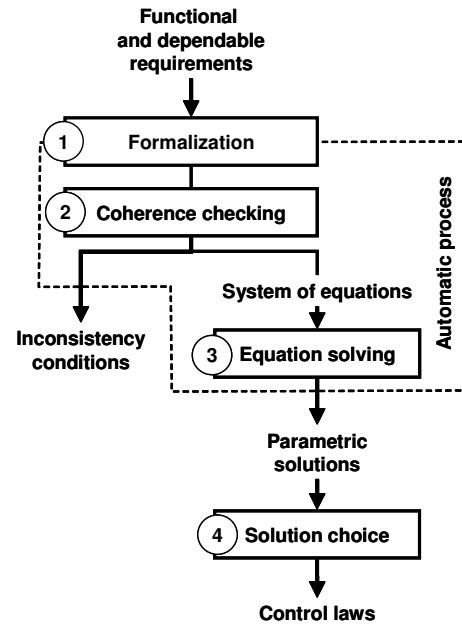


Fig. 1. Method proposed to design dependable logic controllers

## 3. FORMAL FRAMEWORK USED

This section is structured as follows. First, the Boolean algebra structure of Boolean functions is recalled. The second subsection presents the two main relations which are used to express the functional requirements and dependable properties. In the last part the technique developed to solve systems of Boolean equations with several unknowns is given.

### 3.1 The Boolean algebra of Boolean functions

For automation purposes, Boolean Functions (BFs) are often used to determine the value of Boolean outputs according to the logical values of Boolean inputs (figure 2).
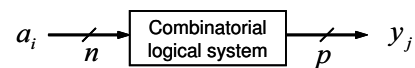


Fig. 2. A combinatorial logical system

From a mathematical point of view, a BF is a function from $\mathbb{B}^n \to \mathbb{B}$, where $\mathbb{B} = \{0, 1\}$ is a Boolean domain. For each $n$-tuple of Boolean variables $(a_1, \ldots, a_n)$, $2^{(2^n)}$ different BFs can be defined. Designing a combinatorial logical system consists in finding, for every output, one of these BFs which satisfies all the requirements. Generally, the solution is not unique.

Let $(a_1, \ldots, a_n)$ be a $n$-tuple of Boolean variables. Let $\Psi_n$ be the set of the $2^{(2^n)}$ possible BFs. $\Psi_n$ contains two BFs which are defined as follows:

$$\mathcal{F}: \begin{array}{c} \mathbb{B}^n \to \mathbb{B} \\ (a_1, \ldots, a_n) \mapsto 0 \end{array} \quad \bigg| \quad \mathcal{T}: \begin{array}{c} \mathbb{B}^n \to \mathbb{B} \\ (a_1, \ldots, a_n) \mapsto 1 \end{array}$$

$\Psi_n$ also contains $n$ specific BFs whose image is a Boolean variable of $(a_1, \ldots, a_n)$. These functions are defined as follows:

$$A_i: \begin{array}{c} \mathbb{B}^n \to \mathbb{B} \\ (a_1, \ldots, a_n) \mapsto a_i \end{array}$$

Nota: In this communication, to avoid confusion between Boolean variables and Boolean functions, Boolean variables are noted with lowercase letters ($a_i$) whereas Boolean functions are noted with an uppercase first letter ($A_i$).

As presented in Grimaldi [2004], $\Psi_n$ with the two binary operations OR (noted "+"), AND (noted "·"), the unary operation NOT (noted "‾") and the two elements $\mathcal{T}$ and $\mathcal{F}$, is a Boolean algebra. These three operations are defined as follows:

$(C + D)(a_1, \ldots, a_n) = (C(a_1, \ldots, a_n)) \vee (D(a_1, \ldots, a_n))$
$(C \cdot D)(a_1, \ldots, a_n) = (C(a_1, \ldots, a_n)) \wedge (D(a_1, \ldots, a_n))$
$\overline{C}(a_1, \ldots, a_n) = \neg(C(a_1, \ldots, a_n))$

where $\vee$, $\wedge$, $\neg$ are the basic operations on Boolean variables.

Nota: To avoid unreadable formulae, the reference to $n$-tuple $(a_1, \ldots, a_n)$ in the notation of BFs will now be removed.

### 3.2 Relations between BFs

As $(\Psi_n, +, \cdot, \overline{\phantom{x}}, \mathcal{F}, \mathcal{T})$ is a Boolean algebra, a partial order relation between elements of $\Psi_n$ can be defined as follows (Grimaldi [2004]):

*Definition 1.* (Inclusion Relation). Let $C, D \in \Psi_n$.

$C \leq D$ iff $C \cdot D = C$ (read "C is included into D")

In other terms: BFs $C$ and $D$ satisfy relation $C \leq D$ if and only if the set of values of $n$-tuple $(a_1, \ldots, a_n)$ whose image by $C$ is 1 is included into the set of values of $n$-tuple $(a_1, \ldots, a_n)$ whose image by $D$ is 1.

In our approach, this relation is often used to express functional requirements and dependable properties. For example, relation $C \leq D$ can be the formal translation of the three following requirements given in natural language:

- When $C$ is true, $D$ is also true.
- It is sufficient to have $C$ to have $D$.
- It is necessary to have $D$ to have $C$.

Moreover, relation "Equality" is used to translate requirements such as:

- $C$ and $D$ are never simultaneously true: $C \cdot D = \mathcal{F}$
- One of BFs $C$ and $D$ is always true: $C + D = \mathcal{T}$

More complex requirements could be expressed by composing BFs using operations OR, AND and NOT.

It is important to notice that requirements expressed with relations "Equality" and "Inclusion" could be associated and reduced to a simple relation. This characteristic is due to the following equivalences:

$$C \leq D \Leftrightarrow C \cdot D = C \qquad (1)$$

$$C = D \Leftrightarrow C \cdot \overline{D} + \overline{C} \cdot D = \mathcal{F} \qquad (2)$$

$$\begin{cases} C = \mathcal{F} \\ D = \mathcal{F} \end{cases} \Leftrightarrow C + D = \mathcal{F} \qquad (3)$$

### 3.3 Solving equations between Boolean functions with several unknowns

The results presented in this section have been demonstrated for any Boolean algebra, and in particular for the algebra of BFs. These demonstrations are the generalization of previous results obtained for Boolean equations between Boolean variables. In Bernstein [1932] the condition for an equation to have a unique solution is given. In Toms [1966], the case of systems of equations is studied. Levchenkov has expressed solutions of equations with several unknowns (Levchenkov [2000]).

Designing the system presented in figure 2 consists in finding, for each output $y_j$, a BF $Y_j$ which allows the calculation of the value of $y_j$ according to the values of each input $a_i$. Functions $Y_j$, which are the unknowns of the problem, must satisfy all the requirements expected for the system.

We consider that all the requirements and properties expected of the system to design can be expressed by using relations "equality" or "inclusion". Examining equation (1) above shows that every inclusion form can be put under an equality form. Equation (2) shows that all equality forms can be put under an "always false" form ($\ldots = \mathcal{F}$). As equation (3) shows that a set of "always false" relations can be reduced into a single "always false" relation, the set of all requirements can globally be expressed by a unique relation of the form:

$$R(Y_1, \ldots, Y_p) = \mathcal{F}$$

where $R(Y_1, \ldots, Y_p)$ is an expression written with only $A_i$ and $Y_j$ and operations OR, AND and NOT.

*Case of one unknown: $R(Y) = \mathcal{F}$*

By generalizing Shannon's decomposition to functions of BFs, $R(Y)$ can be expressed as follows:

$$R(Y) = E_1 \cdot \overline{Y} + E_2 \cdot Y$$

where $E_1$ and $E_2$ are two expressions containing $A_i$ and can be obtained directly by substitution:

$$R(\mathcal{F}) = E_1 \cdot \overline{\mathcal{F}} + E_2 \cdot \mathcal{F} = E_1 + \mathcal{F} = E_1$$
$$R(\mathcal{T}) = E_1 \cdot \overline{\mathcal{T}} + E_2 \cdot \mathcal{T} = \mathcal{F} + E_2 = E_2$$

In this case, the solutions of requirements equation $R(Y) = \mathcal{F}$ are given by the resolution of the following equation:

$$R(\mathcal{F}) \cdot \overline{Y} + R(\mathcal{T}) \cdot Y = \mathcal{F} \qquad (4)$$

*Theorem 1.* Let us consider equation $R(Y) = \mathcal{F}$ expressed on the Boolean algebra of BFs $(\Psi_n, +, \cdot, \overline{\phantom{x}}, \mathcal{F}, \mathcal{T})$, where $R(Y)$ is the generic expression of a single unknown $Y$.
$R(Y) = \mathcal{F}$ has solutions if and only if:

$$R(\mathcal{F}) \cdot R(\mathcal{T}) = \mathcal{F}$$

In this case, all the solutions of this equation can be expressed as follows:

$$Y = R(\mathcal{F}) + \overline{R(\mathcal{T})} \cdot G \qquad (5)$$

Indeed, if the condition $R(\mathcal{F}) \cdot R(\mathcal{T}) = \mathcal{F}$ is respected, then $Y = R(\mathcal{F})$ and $Y = \overline{R(\mathcal{T})}$ are two particular solutions. It can be demonstrated than form 5 represents the whole space of solutions. $G$ is a constant term of $\Psi_n$ whose set of values describes all the solutions. Furthermore, $R(Y) = \mathcal{F}$ has a single solution if and only if:

$$R(\mathcal{T}) = \overline{R(\mathcal{F})}$$

*General case of several unknowns: $R(Y_1, \ldots, Y_p) = \mathcal{F}$*

Starting from the simplified case of logical systems with only one output, we have generalized the previous result to logical systems with $p$ outputs. Before giving the generic form of the general solution, we must now introduce some notations.

By generalizing equation (4), the requirement relation with $p$ unknowns can be expressed as the sum of $2^p$ terms. Each term is the product of two expressions. The second expression is one of the $2^p$ combinations on $\{Y_1, \ldots, Y_p\}$. The first expression is the image of the corresponding combination of functions $\mathcal{F}$ and $\mathcal{T}$ by $R$.

For example, for $p = 2$, the generic notation of $R$ is:
$$R(Y_1, Y_2) = R(\mathcal{F}, \mathcal{F}) \cdot (\overline{Y_1} \cdot \overline{Y_2}) + R(\mathcal{F}, \mathcal{T}) \cdot (\overline{Y_1} \cdot Y_2)$$
$$+ R(\mathcal{T}, \mathcal{F}) \cdot (Y_1 \cdot \overline{Y_2}) + R(\mathcal{T}, \mathcal{T}) \cdot (Y_1 \cdot Y_2)$$

Let $\Omega_p$ be the set of the $2^p$ $p$-tuples whose components are $\mathcal{F}$ or $\mathcal{T}$. Let $\omega$ be an element of this set and $\omega_i$ the $i^{th}$ component of $p$-tuple $\omega$.

$$R(\overbrace{\underbrace{\mathcal{F}}_{\omega_1}, \mathcal{T}, \ldots, \underbrace{\mathcal{T}}_{\omega_i}, \ldots, \mathcal{F}, \underbrace{\mathcal{T}}_{\omega_p}}^{\omega})$$

The generic notation for $R(Y_1, \ldots, Y_p)$ becomes:

$$R(Y_1, ..., Y_p) = \sum_{\omega \in \Omega_p} \left( R(\omega) \cdot \prod_{i=1}^{p} (\omega_i \cdot Y_i + \overline{\omega_i} \cdot \overline{Y_i}) \right)$$

Taking into account these notations, the fundamental theorem (which can not be demonstrated herein) that we will use in the next part of this paper is as follows:

*Theorem 2.* $R(Y_1, Y_2, \ldots, Y_p) = \mathcal{F}$ has solutions iff:

$$\prod_{\omega \in \Omega_p} R(\omega) = \mathcal{F} \qquad (6)$$

In this case, the result of theorem 1 was generalized and a generic form of the solutions is as follows:

$$\begin{cases} Y_1 = \prod_{\omega \in \Omega_{p-1}} R(\mathcal{F}, \omega_1, \ldots \omega_{p-1}) \\ \quad + \overline{\prod_{\omega \in \Omega_{p-1}} R(\mathcal{T}, \omega_1, \ldots \omega_{p-1})} \cdot G_1 \\ \ldots \\ Y_i = \prod_{\omega \in \Omega_{p-i}} R(Y_1, \ldots, Y_{i-1}, \mathcal{F}, \omega_1, \ldots \omega_{p-i}) \\ \quad + \overline{\prod_{\omega \in \Omega_{p-i}} R(Y_1, \ldots, Y_{i-1}, \mathcal{T}, \omega_1, \ldots \omega_{p-i})} \cdot G_i \\ \ldots \\ Y_p = R(Y_1, \ldots, Y_{p-1}, \mathcal{F}) + \overline{R(Y_1, \ldots, Y_{p-1}, \mathcal{T})} \cdot G_p \end{cases}$$

where $G_1, \ldots, G_p$ are constant terms of $\Psi_n$ whose set of values describes the whole space of solutions.

The algorithm that implements the calculation of this parametric solution is the head of our synthesis method.

## 4. SYNTHESIS OF CONTROL LAWS

### 4.1 Model of sequential aspects

The mathematical results presented before allow to design a logical system, starting from requirements given by an algebraic generic relation. These results can directly be used to synthesize combinatorial logic controllers. However, additional information must be given to be able to treat the sequential part of logical systems.

During the design step, the designer often uses state machines such as Automaton, Statecharts or Sequential Function Chart to model sequential behaviours. When the design step is over, these state models are often translated into an algebraic program which is more adapted to its execution by programmable controllers. These translations are based on the well-known model presented in figure 3 based on the works of Shannon [1940]. Efficient techniques to translate a state model into a set of recurrent equations can be found in Machado et al. [2006].
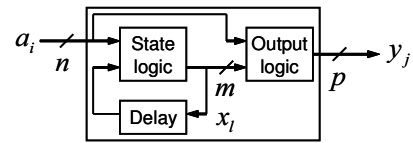


Fig. 3. Diagram of a Sequential Machine

The diagram in figure 3 is composed of two combinatorial blocks "State Logic" and "Output logic". The values of state variables $x_l$ are calculated by block "State Logic" and the values of the outputs are calculated by block "Output logic" according to the following equations:

$$\begin{cases} x_l(k) = S_l(a_1(k), \ldots, a_n(k), x_1(k-1), \ldots, x_m(k-1)) \\ y_j(k) = Y_j(a_1(k), \ldots, a_n(k), x_1(k), \ldots, x_m(k)) \end{cases}$$

In this system of equations, $x_l(k)$ is the value of $x_l$ at instant $k$ and $x_l(k-1)$ is the value of $x_l$ at previous instant $(k-1)$. It is possible to treat such recurrent equations by using the formal framework presented before by associating a different variable to the "current value" and to the "previous value" of the same variable. We are now going to detail our method by treating a simple example that has two characteristics: its specification is given by using both logical propositions and an automaton; its behaviour is partially combinatorial and partially sequential.

### 4.2 Application to an example

The example used to present our approach deals with the control of the electrical energy of a production machine. To protect operators, this machine is equipped with a safety system which allows access only by using a door. In normal mode (during operation), this door is closed and automatically locked. For maintenance operations, the door can be opened. In this case, the technician is protected by the obligation to use a two-hand control station (figure 4).

The driving of the machine by operators is made through a control console which allows:

- to select the operation mode (Production or Maintenance) with a two-position selector ($prod/maint$),
- to ask for the locking of the door with a switch (request of locking: $rl$),
- to start and stop the machine with two distinct push-buttons ($start$ and $stop$).
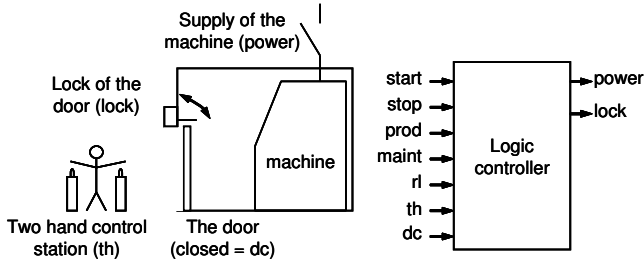
Fig. 4. Structure of the production machine

The controller is informed that the door is closed by sensor ($dc$) and that the two-hand control station is activated ($th$). The control laws to synthesize are the power supply of the machine ($power$) and the door locking ($lock$).

The six following requirements are given in natural language under the form of logical propositions. The two first ones are safety requirements whereas the other ones are functional requirements:

R1: In Production mode, the machine is supplied only if the door is locked.
R2: In Maintenance mode, the machine is supplied only if the two-hand control station is active.
R3: In Maintenance mode, the door is never locked.
R4: In Maintenance mode, when the two-hand control station is active, the machine is supplied.
R5: To lock the door, it is necessary that the door is closed.
R6: In Production mode, when the door is closed, the door is locked if and only if the locking request is active.

According to the method presented in figure 1, we suppose for this example that the designer has proposed a partial solution for output *power* (figure 5). Indeed, the control law of *power* concerns both the operation of the machine and the security of operators. In this Moore's machine, only a part of the transition conditions is fixed (the functional parts), the parts to be synthesized (the security parts) are expressed by $U_i$. In fact, function $U_i$ represented the parts which are unknown for the designer.
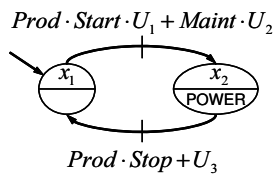


Fig. 5. A partial solution proposed by the designer

The two control laws (*power* and *lock*) are now going to be synthesized from previous requirements and the partial solution proposed by the designer.

*Formalization step*

The result of the formalization step is given below. The six requirements given in natural language are expressed by using relation "Inclusion". The behaviour of the Moore's machine is described by six other relations. The three first ones concern its dynamics ($X_i$ is the BF associated to variable $x_i(k)$ and $X_ip$ represents variable $x_i(k - 1)$). The two following ones are introduced to precise that this Moore's machine has one and only one active

state. The last relation specifies that the behaviour of the machine must be stable (no transition can be fired without inputs values change). This requirement was previously introduced by Huffman [1954].

$$\begin{cases} \text{R1: } Prod \cdot Power \leq Lock \\ \text{R2: } Maint \cdot Power \leq Th \\ \text{R3: } Maint \leq \overline{Lock} \\ \text{R4: } Maint \cdot Th \leq Power \\ \text{R5: } Lock \leq Dc \\ \text{R6: } Prod \cdot Dc \leq Rl \cdot Lock + \overline{Rl} \cdot \overline{Lock} \\ X_1 = X_2p \cdot (Prod \cdot Stop + U_3) \\ \quad + X_1p \cdot \overline{(Prod \cdot Start \cdot U_1 + Maint \cdot U_2)} \\ X_2 = X_1p \cdot (Prod \cdot Start \cdot U_1 + Maint \cdot U_2) \\ \quad + X_2p \cdot \overline{(Prod \cdot Stop + U_3)} \\ Power = X_2 \\ X_1 \cdot X_2 = \mathcal{F} \\ X_1 + X_2 = \mathcal{T} \\ (Prod \cdot Start \cdot U_1 + Maint \cdot U_2) \\ \quad \cdot (Prod \cdot Stop + U_3) = \mathcal{F} \end{cases}$$

Each one of these BFs depends on the 9 Boolean variables ($start$, $stop$, $prod$, $maint$, $rl$, $th$, $dc$, $x_1p$, $x_2p$). Four of these BFs are unknowns: $Lock$, $U_1$, $U_2$ and $U_3$.

Relations between the 9 elementary BFs also exist as they are not totally independent. These relations which are used as hypotheses during calculation are:

$$\begin{cases} X_1p + X_2p = \mathcal{T} \\ X_1p \cdot X_2p = \mathcal{F} \\ Prod + Maint = \mathcal{T} \\ Prod \cdot Maint = \mathcal{F} \end{cases}$$

The two first relations are introduced to precise that the Moore's machine has one and only one previous active state. The two last ones are introduced to precise that the operational mode selector has only two positions.

The set of requirements can be reduced into a unique equation of the following form:

$$R(Lock, U_1, U_2, U_3) = \mathcal{F} \tag{7}$$

The generic form of $R(Lock, U_1, U_2, U_3)$ is composed of 16 terms. The global expression of R is:

$$R(Lock, U_1, U_2, U_3) = R(\mathcal{F}, \mathcal{F}, \mathcal{F}, \mathcal{F}) \cdot \overline{Lock} \cdot \overline{U_1} \cdot \overline{U_2} \cdot \overline{U_3} \\ + \ldots + R(\mathcal{T}, \mathcal{T}, \mathcal{T}, \mathcal{T}) \cdot Lock \cdot U_1 \cdot U_2 \cdot U_3$$

With for example, term $R(\mathcal{F}, \mathcal{F}, \mathcal{F}, \mathcal{F})$ is:

$$R(\mathcal{F}, \mathcal{F}, \mathcal{F}, \mathcal{F}) = Prod \cdot (Dc \cdot Rl + \overline{Stop} \cdot X_2p) \\ + Maint \cdot (Th \cdot \overline{X_2p} + \overline{Th} \cdot X_2p)$$

*Coherence checking step*

The second step of our method is the coherence checking of the requirements. In our case, the condition expressed in equation (6) is respected:

$$\prod_{\omega \in \Omega_p} R(\omega) = R(\mathcal{F}, \mathcal{F}, \mathcal{F}, \mathcal{F}) \cdot \ldots \cdot R(\mathcal{T}, \mathcal{T}, \mathcal{T}, \mathcal{T}) = \mathcal{F}$$

The set of requirements is then coherent.

*Equation solving step*

The algorithm that implements the calculation of the parametric solution given in 3.3 produces:

$$
\begin{cases}
Lock = Dc \cdot Prod \cdot Rl \\
U_1 = (Maint + \overline{Start} + Dc \cdot Rl \cdot \overline{Stop}) \cdot G_1 \\
U_2 = (Prod + Th) \cdot G_2 + Maint \cdot Th \cdot X_1 p \\
U_3 = Maint \cdot \overline{Th} \cdot X_2 p + (\overline{Dc} + \overline{Rl}) \cdot Prod \cdot \overline{Stop} \cdot X_2 p \\
\quad + (Prod \cdot (\overline{Dc} + \overline{Start} + Stop + \overline{Rl} + \overline{G_1}) \\
\quad\quad + Maint \cdot \overline{Th}) \cdot G_3
\end{cases}
$$

Only one solution exists for *Lock*. $U_1$, $U_2$ and $U_3$ have a space of solutions expressed by parameters $G_1$, $G_2$ and $G_3$.

*Solution choice*

The last step of the method is the choice of a particular solution. This choice is made by the designer according to heuristics that define his strategy. For example, to obtain the most permissive solution (priority to the functioning) that satisfies the requirements, it is necessary to maximize the reachability of state 2 and to minimize the evolution possibilities from state 2. To obtain this result, the parameters are fixed as follows: $G_1 = \mathcal{T}$, $G_2 = \mathcal{T}$, $G_3 = \mathcal{F}$

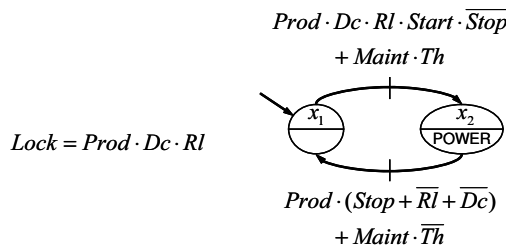Finally, figure 6 presents the control laws synthesized.



Fig. 6. Control laws algebraically synthesized

## 5. CONCLUSION

The method presented herein has been developed to design dependable logic controllers from requirements given using state machines or logical propositions. This method can be used to find the laws to implement into industrial controllers or to complete a partial solution given by the designer. These two characteristics are offered by the mathematical framework used (an algebra for Boolean functions) and more specially by the possibility to solve Boolean equations with several unknowns.

To be able to evaluate the potential of this method from case studies, we have developed (under tool Mathematica) an experimental module which solves sets of equations with several unknowns. It is based on a symbolic calculus module which develops and simplifies Boolean expressions. The results already obtained have pointed out the real interest to exploit the partial solution proposed by the designer. For the designer, it is simpler to propose a partial solution than to express the corresponding functional requirements. At the end of the design step, the solution obtained satisfies all the requirements given. As the state model structure is not changed, the designer can analyze it more easily.

The potential of this method can be increased by offering additional models to express requirements. The possibility to use temporal logic is currently studied. We are also searching if some heuristics could be defined to help the designer for the choice of a solution for specific classes of problems.

To be operational for the design of complex dependable control systems, it is also necessary to develop strategies to identify the independent parts of the problem, in order to solve each part independently. The size of the equations studied will be reduced.

## REFERENCES

B.A. Bernstein. Note on the Condition that a Boolean Equation Have a Unique Solution. *American Journal of Mathematics*, 54(2):417–418, 1932.

O. De Smet and O. Rossi. Verification of a controller for a flexible manufacturing line written in Ladder Diagram via model-checking. *American Control Conference, 2002. Proceedings of the 2002*, 5:4147–4152, 2002.

J.-M. Faure and J.-J. Lesage. Methods for safe control systems design and implementations. *Proceedings of the 10th IFAC Symposium on Information Control Problems in Manufacturing, Vienna, Austria*, 2001.

R.P. Grimaldi. *Discrete and Combinatorial Mathematics: An Applied Introduction.* Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 5 edition, 2004. ISBN : 0-321-21103-0, 980 pages.

D.A. Huffman. The synthesis of sequential switching circuits. *Journal of the franklin Institute*, 257(3):161–190, 1954.

S. Klein, G. Frey, and L. Litz. Designing fault-tolerant controllers using Model-Checking. *proc. of the 5th IFAC Symposium on Fault Detection, Supervision and Safety of Technical Processes (SAFEPROCESS 2003), Washington DC (USA), June*, pages 115–120, 2003.

R. Kumar, V.K. Garg, and S.I. Marcus. On controllability and normality of discrete event dynamical systems. *Systems and Control Letters*, 17(3):157–168, 1991.

V.S. Levchenkov. Boolean equations with many unknowns. *Computational Mathematics and Modeling*, 11(2):143–153, 2000.

J. Machado, B. Denis, J.-J. Lesage, J.-M. Faure, and J. Ferreira Da Silva. Logic controllers dependability verification using a plant model. *3 rd IFAC Workshop on Discrete-Event System Design: DESDes'06*, pages 37–42, 2006.

J.-F. Pétin, D. Gouyon, and G. Morel. Supervisory synthesis for product-driven automation and its application to a flexible assembly cell. *Control Engineering Practice*, 15(5):595–614, 2007.

P.J. Ramadge and W.M. Wonham. On the supremal controllable sublanguage of a given language. *SIAM Journal of Control and Optimization*, 25(3):637–659, 1987.

J.-M. Roussel and J.-M. Faure. An algebraic approach for PLC programs verification. *Proceedings. Sixth International Workshop on Discrete Event Systems*, pages 303–308, 2002.

J.-M. Roussel and A. Guia. Designing dependable logic controllers using the supervisory control theory. *16th IFAC World Congress, CDROM paper n04427*, 2005.

C.E. Shannon. A symbolic analysis of relay and switching circuits. Master's thesis, Massachusetts Institute of Technology, Dept. of Electrical Engineering, 1940.

R.M. Toms. Systems of Boolean Equations. *The American Mathematical Monthly*, 73(1):29–35, 1966.