Designing dependable logic controllers using algebraic specifications

Jean-Marc Roussel(1), Jean-Marc Faure(1)(2)

(1)LURPA, ENS de CACHAN, 61 avenue du président Wilson, F-94235 CACHAN Cedex, France.
(2)SUPMECA, 3 rue Fernand Hainaut, F-93407 Saint-Ouen Cedex, France.

Abstract

Formal methods can strongly contribute to improve dependability of logic controllers during design, by providing means to avoid flaws due to designers' omissions or specifications misinterpretations. This article presents a formal synthesis method that is aimed at obtaining the control laws of a logic system from specifications given in natural language. The formal framework that underlies the method is a Boolean algebra for logic Discrete Event Systems. The operations and relations of this algebra enable to represent controller specifications formally, to detect inconsistencies within specifications and to generate control laws from a consistent specifications set. The scalability of this method is clearly demonstrated with the help of the case study of an experimental manufacturing line.

Keywords: Discrete Event Systems; Formal methods; Dependability; Logic controllers; Algebraic approach; Boolean algebra; Synthesis

1. Introduction

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

To improve dependability of logic controllers, it is important to make sure that no flaw due to a misinterpretation of the specifications 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 fulfils the application requirements and therefore must not include design errors leading to non-functional or hazardous behaviours.

To solve this problem, numerous formal methods have been proposed during the last decade. They are aimed either at detecting flaws once the controller is designed or at avoiding flaws during design. The first approach consists in letting the control system designer develop control laws based on the requirements contained in the set of specifications and then in automatically analysing a formal representation of these control laws. Such an analysis may be carried out by using model-checking techniques (Bérard et al., 1999) provided that control laws can be formally represented in the form of state automata (De Smet & Rossi, 2002; Klein et al., 2003). The second approach (Ramadge & Wonham, 1989; Wang, 2000; Zaytoon & Carré-Ménétrier, 2001; Nourelfath & Niel, 2004; Gouyon et al., 2004) is intended to deduce directly the control laws from the specifications, without any involvement of a designer (or at least in limiting involvement to a strict minimum).

The work presented herein is aimed at avoiding flaws during the design of logic controllers, by proposing a formal method that enables the deduction of a complete and consistent formal description of control laws, from specifications expressed in natural language. A specific algebraic modelling framework underlies this formal method; this feature prevents state space explosion that is often encountered when applying the previously mentioned formal methods to industrial size problems. Moreover the formal description of control laws obtained at the end of design can be easily translated into a program developed in a standardised language for programmable logic controllers. The aim of this formal method is indeed to provide a seamless whole from specifications analysis to implementation.

This article is organised as follows. Section 2 is dedicated to the objective and the main principles of this method as well as to the introduction to the formal framework. The basic definitions, operations and relations of this framework are discussed in section 3, while section 4 deals with symbolic calculus possibilities, focusing mainly on equations system solving. Once the formal bases are defined, the design method is presented in section 5 thanks to a simple example. The result obtained is compared in section 6 with a SFC developed by a traditional engineering approach, so as to point out two significant benefits of the proposed formal method. In order to assess the scalability of this method, section 7 describes a case study of a controller including 72 control laws. The results obtained in this case study provide investigation prospects discussed in the last section.

2. Objective

The starting point of the proposed design method is the set of specifications inherent in the control system, as expressed in natural language. These specifications describe the expected behaviour of the control system, in the form of vivacity constraints (what the control system must accomplish) and safety constraints (what the system must not accomplish), and may include constraints coming from actuator and sensor technology choices. All of these constraints are to be expressed in the form of logic assertions, i.e., propositions that must be true for the desired control system (see section 5.1). At this point, however, it is important to highlight the following points:

- The control specifications of a logic system can make reference to logic variable *states*, to state changes of these logic variables (*events*), or to *physical time* values. The formalism that supports the design method is to be endowed with the capability of expressing these three types of variables.

- A set of specifications does not necessarily have to be consistent. The design method must therefore be capable of detecting possible inconsistencies in the specifications and then of proposing solutions to solve these problems.

Control laws are to be designed on the basis of this list of assertions. For a dynamic logic control system with n Boolean input variables $U_1(t)$ to $U_n(t)$ and m Boolean output variables $Y_1(t)$ to $Y_m(t)$, the target control laws must specify at each date t the output values as functions of input values, which leads to (Cassandras & Lafortune, 1999):

$$\begin{cases} Y_1(t) = f_1(U_1(t), \ldots, U_n(t)) \\ \qquad \ldots \\ Y_m(t) = f_m(U_1(t), \ldots, U_n(t)) \end{cases}$$

The search for solutions to this system of m equations generally requires a reformulation of the problem in the form of a state model, thereby the introduction of other variables, state variables X_j recording the system evolution. Despite its advantages, this approach displays the disadvantage of merely providing specific solutions, at a given point in time, as exemplified in the form of "Y_i becomes true when U_k becomes false and if variables X_2, X_5 and X_8 are false while variables X_3 and X_{10} true", but never a general solution that holds true regardless of the date considered. Moreover modelling a state automaton-based dynamic system corresponds to an imperative design approach, whereas control system specifications are in most cases given in declarative form. The transition from one of these representation modes to the other one always requires a major effort.

For both reasons, the proposed design method relies upon a special formalism, named algebra $I\!I$ (called I), that was developed by our research team. The basic elements of this formalism consist of time functions $U_i(t)$, $Y_i(t)$. The operations that enabled us to compose these functions lead to defining an algebraic structure in which the simultaneous manipulation of Boolean variable states, state changes of these variables (events) and physical time values is possible.

In sum, the proposed design method for dependable logic control systems calls for developing a set of control laws in the form of time functions, from specifications given in the form of assertions in natural language. This method (Fig. 1) requires formalizing specifications into relations within algebra $I\!I$, checking the consistency of the set of assertions and generating the expected control laws from the consistent set of specifications obtained.

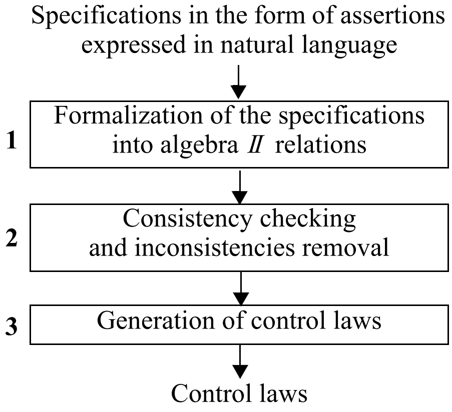

Specifications in the form of assertions
expressed in natural language
1
Formalization of the specifications into algebra $I\!I$ relations
2
Consistency checking and inconsistencies removal
3
Generation of control laws
Control laws

Fig. 1 Method overview

A large part of the method – consistency checking and control laws generation – is automated thanks to a symbolic calculus software tool able to manipulate algebra $I\!I$ statements. Conversely inconsistencies removal will always require the designer's competence.

3. Formal framework: a Boolean algebra for binary signals

3.1 Concept of binary signals

To describe the behaviour of logic systems, control engineers must refer to logic variables states, to state changes of these variables (events), and to physical time values. This explains why they are used to employ commonly timing diagrams that describe, in an intuitive way, logic signals behaviour.

Algebra $\mathbb{I}$ was defined to manipulate formally such signals. Contrary to the usual Boolean algebra which basic automatic control lectures deal with, algebra $\mathbb{I}$ takes into account temporal issues by manipulating binary signals: functions of time with Boolean values (Fig. 2).

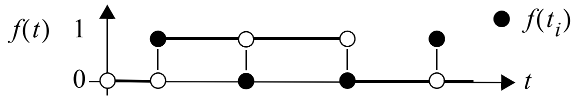


Fig. 2 Graphical representation of a binary signal

The elements of algebra $\mathbb{I}$ are indeed piecewise-continuous functions from $\mathbb{R}^{+*}$ to $\mathbb{B} = \{0, 1\}$ ($\mathbb{R}^{+*}$ means $(0, +\infty)$), formally defined as follows:

$$\mathbb{I} = \left\{ \begin{array}{l} f : \mathbb{R}^{+*} \to \mathbb{B} \mid \forall t \in \mathbb{R}^{+*} : \\ (\exists \varepsilon_t > 0 : (\forall (\varepsilon_1, \varepsilon_2) \in (0, \varepsilon_t)^2, f(t - \varepsilon_1) = f(t - \varepsilon_2))) \end{array} \right\}$$

Binary signals can be composed in a combinatory way, i.e. to obtain a signal whose value at each date is obtained from the values of the operands at the same date, with three basic operations And, Or, Not, respectively noted $.$, $+$, $\overline{}$. According to Grimaldi (2000), these three operations endow algebra $\mathbb{I}$ with a Boolean algebra structure that permits to obtain all classical results for this class of algebra (commutative laws, De Morgan's laws, ...).

3.2 Specific operations defined on algebra $\mathbb{I}$

Sequential and timed operations are to be defined to describe more complex behaviours (Roussel et al. 2004), such as those included in the specifications of industrial control systems (Fig. 3 and Fig. 4). Only the definitions of sequential operations (SR and RS) and of one timed operation (TON) are given hereinafter.

TON operation	SR operation	RS operation
$\mathbb{I} \to \mathbb{I}$	$\mathbb{I}^2 \to \mathbb{I}$	$\mathbb{I}^2 \to \mathbb{I}$
$f \to d/f$	$(s, r) \to SR(s, r)$	$(s, r) \to RS(s, r)$

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

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

$$SR(s, r)(t) = s(t) \vee$$
$$[\exists t_1 < t : ((s(t_1) = 1) \wedge (\forall d \in (t_1, t] : (r \cdot \bar{s})(d) = 0))]$$
$$RS(s, r)(t) = (s(t) \wedge \neg r(t)) \vee$$
$$[\exists t_1 < t : ((s(t_1) = 1) \wedge (\forall d \in [t_1, t], (r(d) = 0)))]$$

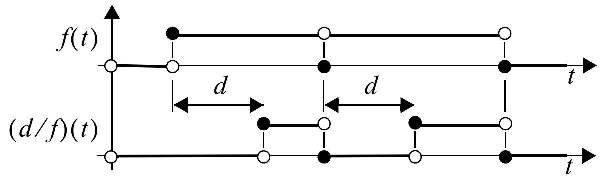


Fig. 3 Graphical representation of TON operation

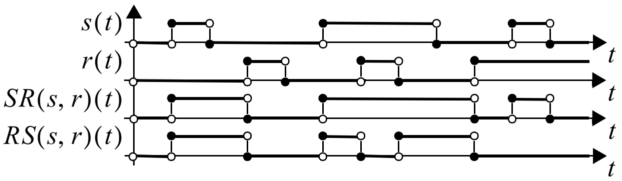
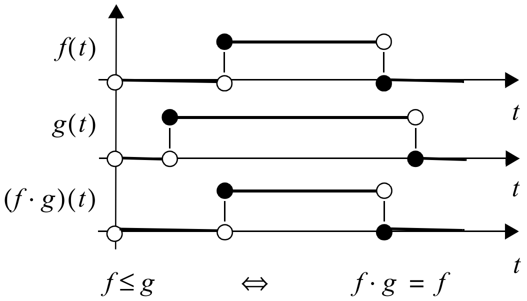


Fig. 4 Graphical representation of $SR(s, r)$ and $RS(s, r)$ operations

3.3 Relations defined on algebra $\mathbb{I}$

Two mathematical relations between binary signals are defined on algebra $\mathbb{I}$:

- the relation "equality" (noted $f = g$) which states that the values of these signals are equal, whatever the considered date.
- the relation "inclusion" (noted $f \leq g$) which states that for all dates t_i such as value $f(t_i)$ of signal f is true, value $g(t_i)$ of signal g is true too (Fig. 5).



$f \leq g \quad \Leftrightarrow \quad f \cdot g = f$

Fig. 5 Binary signals such as: $f \leq g$

From a mathematical point of view, the relation "inclusion" corresponds to the partial ordering relation which can be defined on each Boolean algebra. In (Grimaldi, 2000), this partial ordering relation is defined as follows:

If $(f, g) \in \mathbb{I}$, define $f \leq g$, if $f \cdot g = f$

This relation is reflexive, antisymmetric and transitive.

For all f, g elements of $I\!I$, the 7 following relations were proved equivalent[1]:

$$f \le g \qquad f \cdot g = f \qquad \overline{f} + g = 1^* \qquad f \cdot \overline{g} = 0^*$$
$$\overline{g} \le \overline{f} \qquad \overline{g} \cdot \overline{f} = \overline{g} \qquad f + g = g$$

The following results, that enable to split a partial ordering relation into two equivalent ones, were also proved:

$$(f+g) \le h \Leftrightarrow \begin{cases} f \le h \\ g \le h \end{cases} \qquad f \le (g \cdot h) \Leftrightarrow \begin{cases} f \le g \\ f \le h \end{cases}$$

These two relations – equality and inclusion – are the cornerstone of our approach. Assertions describing the expected behaviour of control systems in natural language can indeed be translated into formal statements thanks to these relations. Generic assertions and the equivalent formal relations are given in table 1. Sections 5 and 7 present assertions and relations describing real control specifications.

Table 1:
Formalization of assertions on $I\!I$

Assertions given in natural language	Relations on $I\!I$
Values of signals f and g are always equal.	$f = g$
Values of signals f and g are never simultaneously true.	$f \cdot g = 0^*$
At each time, at least one of the values of signals f and g is true.	$f + g = 1^*$
When the value of signal f is true, the value of signal g is true.	$f \le g$
It is sufficient that the value of signal f is true to get the value of signal g true.	$f \le g$
It is necessary that the value of signal f is true to get the value of signal g true.	$g \le f$
Value of signal f is never true more than 3 seconds.	$3\mathrm{s}/f = 0^*$

4. Symbolic calculus on algebra $I\!I$

The above operations and relations enable symbolic calculus on binary signals. This important feature can be employed for two purposes:

- to prove properties of existing logic controllers (Roussel & Denis, 2002; Roussel & Faure, 2002)
- to design dependable logic controllers.

This article addresses the second issue, the objective being control laws design from formal assertions. In this synthesis approach, each control law is obtained formally by solving a system of equations coming from assertions, as it is explained in the next section. The bases of symbolic calculus that underlie this synthesis method are given below.

4.1 Useful theorems

Performing symbolic calculus on binary signals requires theorems that were previously demonstrated on algebra $I\!I$. These theorems (Roussel et al., 2004) enable formulae simplification and rewriting relations. Up to now over 100 theorems have been demonstrated on algebra $I\!I$, some of them leading to simple, usual results, (e.g. $SR(s, \overline{s}) = s$). In this article the five following theorems are used:

$$RS(s, r) = SR((s \cdot \overline{r}), r) \qquad s \le SR(s, r)$$
$$r \cdot \overline{s} \le \overline{SR(s, r)} \qquad r \le \overline{RS(s, r)}$$
$$d/f \le f$$

4.2 Solving a system of equations on $I\!I$

Let us consider three binary signals a, b and x, a and b being known and x unknown such as:

- It is necessary that the value of signal a is true so as to obtain the value of signal x true.
- It is sufficient that the value of signal b is true so as to obtain the value of signal x true.

These two requirements can be formally stated as the following system thanks to relation "$\le$".

$$\begin{cases} x \le a \;\; (1a) \\ b \le x \;\; (1b) \end{cases} \text{or} \begin{cases} \bar{a} \le \bar{x} \;\; (1a) \\ b \le x \;\; (1b) \end{cases} \qquad (1)$$

Proposition 1 System (1) admits a solution if and only if signals a and b satisfy the following condition: $\bar{a} \cdot b = 0^*$. When this condition is satisfied, the general solution of system (1) is $x = b + a \cdot f$ with f any signal of $I\!I$.

Proof. This demonstration includes three steps.

- Existence of solutions

If x is a solution, as "$\le$" is a transitive relation, the following result becomes:

$$\begin{cases} x \le a \\ b \le x \end{cases} \Leftrightarrow \qquad b \le x \le a \qquad \Rightarrow \qquad b \le a$$

As $b \le a$ is equivalent to $\bar{a} \cdot b = 0^*$, condition $\bar{a} \cdot b = 0^*$ is therefore necessary to obtain a solution for system (1).

- $x = b + a \cdot f$, with f any signal of $I\!I$, is a solution of system (1).

As $x \le a$ is equivalent to $\bar{a} \cdot x = 0^*$, and as $b \le x$ is equivalent to $b \cdot \bar{x} = 0^*$, it is sufficient to verify that $\bar{a} \cdot x + b \cdot \bar{x} = 0^*$.

[1] 1^* : signal whose value is always 1.
0^* : signal whose value is always 0.

$$
\begin{aligned}
\bar{a}\cdot x + b\cdot\bar{x} &= \bar{a}\cdot(b + a\cdot f) + b\cdot\overline{(b + a\cdot f)}\\
&= \bar{a}\cdot b + a\cdot\bar{a}\cdot f + b\cdot(\bar{b}\cdot\overline{(a\cdot f)})\\
&= \bar{a}\cdot b + 0^* + 0^* = 0^*
\end{aligned}
$$

- if x a solution of system (1), there exists a function f of $\mathbb{I}$ for which: $x = b + a\cdot f$

For this proof, this intermediate result is need:

$$\forall(g, h)\in\mathbb{I}^2, \exists(f_1, f_2)\in\mathbb{I}^2 \qquad g = h\cdot f_1 + \bar{h}\cdot f_2$$

To prove this result, it is sufficient to find two signals f_1 and f_2 that satisfy: $h\cdot f_1 + \bar{h}\cdot f_2 = g$. Let us $(k_1, k_2)\in\mathbb{I}^2$ and assume $f_1 = h\cdot g + \bar{h}\cdot k_1$ and $f_2 = \bar{h}\cdot g + h\cdot k_2$, then:

$$
\begin{aligned}
h\cdot f_1 + \bar{h}\cdot f_2 &= h\cdot(h\cdot g + \bar{h}\cdot k_1) + \bar{h}\cdot(\bar{h}\cdot g + h\cdot k_2)\\
&= h\cdot g + 0^* + \bar{h}\cdot g + 0^* = g
\end{aligned}
$$

With this result, it is possible to assume:

$$\forall(x, a, b)\in\mathbb{I}^3, \exists(f_1, f_2, f_3, f_4)\in\mathbb{I}^4$$

$$x = a\cdot b\cdot f_1 + a\cdot\bar{b}\cdot f_2 + \bar{a}\cdot b\cdot f_3 + \bar{a}\cdot\bar{b}\cdot f_4$$

As x is solution of $b\le x$, $b\cdot x = b$

$$
\begin{aligned}
& b\cdot x = b\\
\Leftrightarrow\quad & b\cdot(a\cdot b\cdot f_1 + a\cdot\bar{b}\cdot f_2 + \bar{a}\cdot b\cdot f_3 + \bar{a}\cdot\bar{b}\cdot f_4) = b\\
\Leftrightarrow\quad & a\cdot b\cdot f_1 + \bar{a}\cdot b\cdot f_3 = b
\end{aligned}
$$

thus, $x = b + a\cdot\bar{b}\cdot f_2 + \bar{a}\cdot\bar{b}\cdot f_4$

As x is solution of $x\le a$, $\bar{a}\cdot x = 0^*$:

$$
\begin{aligned}
\bar{a}\cdot x = 0^* \quad\Leftrightarrow\quad & \bar{a}\cdot(b + a\cdot\bar{b}\cdot f_2 + \bar{a}\cdot\bar{b}\cdot f_4) = 0^*\\
\Leftrightarrow\quad & \bar{a}\cdot b + \bar{a}\cdot\bar{b}\cdot f_4 = 0^*\\
\Leftrightarrow\quad & 0^* + \bar{a}\cdot\bar{b}\cdot f_4 = 0^*\\
\Leftrightarrow\quad & \bar{a}\cdot\bar{b}\cdot f_4 = 0^*
\end{aligned}
$$

thus: $x = b + a\cdot\bar{b}\cdot f_2 = b + a\cdot f_2$

□

Remark 1 There is not a single solution for system (1) because value $x(t_i)$ of signal x is not imposed for all dates t_i such as $(a\cdot\bar{b})(t_i) = 0$. As $b\le a$, the set of solutions of system (1) admits signal b as lower bound element ($f = 0^*$) and signal a as upper bound element ($f = 1^*$).

Proposition 2 When $\bar{a}\cdot b = 0^*$ is satisfied, signal $x = SR(b, \bar{a}) = RS(b, \bar{a})$ is solution of system (1).

The main feature of the proposed solution $x = SR(b, \bar{a})$ is that it remains constant for all dates t_i such as $(a\cdot\bar{b})(t_i) = 0$ and then it keeps the last value imposed by the system (see Fig. 6).

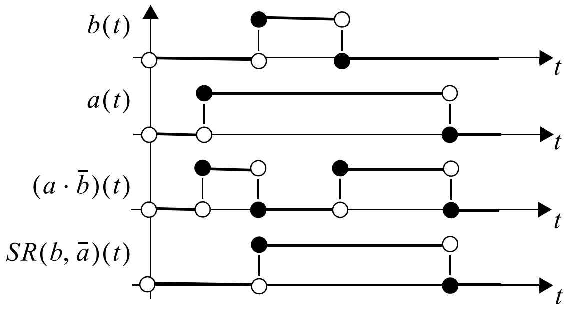


Fig. 6 Graphical representation of $SR(b, \bar{a})$ with $\bar{a}\cdot b = 0^*$

5. Algebraic synthesis of control laws

In this section, a simple example is used to present our approach. It concerns the control of an automatic gate for a car park. The inputs and the outputs of the control system to design are given in Fig. 7.

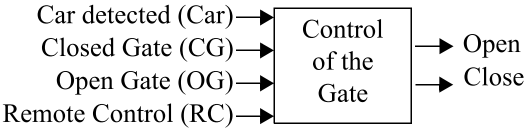


Fig. 7 Inputs and outputs of the control system to design

Two control laws are to be designed:

$$\begin{cases}\text{Open} = \text{F}_1(\text{Car, CG, OG, RC})\\ \text{Close} = \text{F}_2(\text{Car, CG, OG, RC})\end{cases}$$

where F_1 et F_2 are solutions of two systems as system (1). Each system is built from formal assertions issued from specifications given in natural language.

5.1 Control system specifications

The expected behaviour of the control system with regard to the application requirements can be expressed by the set of assertions given hereafter. Among the 8 assertions, the first three ones (A1 to A3) are related to vivacity requirements (what must be done to perform the expected task), assertions A4 and A7 express safety requirements. Assertions A5 and A6 express constraints coming from actuators features and the last one (A8) is an assumption on the correct operation of the sensors (the problem of sensors monitoring will not be dealt with in this study).

A1 When the remote control is activated, the gate opens.

A2 When the gate has been open for 3 seconds without a request from the user or the detection of a car, the gate closes.

A3 While the gate is not totally closed, the detection of a car causes the reopening of the gate.

A4 The gate must never be simultaneously controlled to open and to close.
A5 An open gate cannot be controlled to open.
A6 A closed gate cannot be controlled to close.
A7 When the remote control is activated, the gate cannot be controlled to close.
A8 The gate is never simultaneously open and closed.

5.2 Formalization of specifications

The previous set of assertions can be translated into a set of formal relations (table 2) that include input signals (Car, CG, OG, RC) and output signals (Open, Close). For more complex control systems than this example, internal signals can also be defined.

Table 2:
Formal assertions for the control system

Assertions	Assertions written on *II*
A1	$RC \le Open$
A2	$3s/(OG \cdot \overline{(RC + Car)}) \le Close$
A3	$\overline{CG} \cdot Car \le Open \cdot \overline{Close}$
A4	$Open \cdot Close = 0^*$
A5	$OG \cdot Open = 0^*$
A6	$CG \cdot Close = 0^*$
A7	$RC \cdot Close = 0^*$
A8	$OG \cdot CG = 0^*$

5.3 Equations systems design

From this set of formal assertions, two systems of equations are to be built (one for each output). The first step of this design process is aimed at labelling assertions according to the kind of binary signals that they include.

- Assertions that include only input signals. These relations are assumptions. They are used to simplify control laws.
- Assertions that include only one output or one internal signal and input signals. Each relation is used to build the respective control law.
- Assertions that include several outputs or internal signals. In that case, either the relation can be decomposed in elementary relations (relations comprising only one output or internal signal) as stated in section 3.3, that leads to the previous case (e.g. assertion A3), or the designer has to decide which signal is function of the other ones (e.g. assertion A4). This decision introduces a partial order between signals.

The result of this analysis for the control system of the automatic gate is presented in table 3. For safety reasons, opening the gate has priority over closing. Therefore signal Open must be used to compute signal Close (assertion A4).

Table 3:
Dependency relations derived from assertions labelling

Output signal	Assertions to be used	Signals involved in the assertions
Open	A1, A3, A5	Car(A3a), CG(A3a), OG(A5), RC(A1)
Close	A2, A3, A4, A6	Car(A2, A3b), CG(A3b, A6), OG(A2), RC(A2, A7), Open(A4)

From previous results a dependency graph can be built (Fig. 8). If this graph does not include any cycle, the partial order between signals is consistent. If it is not the case, part of the assertions set must be modified by the designer in order to eliminate these inconsistencies.

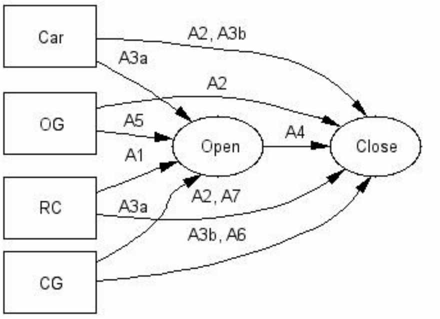


Fig. 8 Dependency graph derived from assertions labelling

The systems of equations that must be solved are presented in table 4. These systems must be solved according to the above priority rules, i.e., the system defining signal Open must be solved prior to the system defining signal Close.

Table 4:
Systems of equations to solve

Signal	Initial system
Open	$\begin{cases} Open = F_1(Car, CG, OG, RC) \\ (A1)\quad RC \le Open \\ (A3a)\quad \overline{CG} \cdot Car \le Open \\ (A5)\quad OG \le \overline{Open} \\ \text{Assumption: } OG \cdot CG = 0^* \end{cases}$
Close	$\begin{cases} Close = F_2(Car, CG, OG, RC, Open) \\ (A2)\quad 3s/(OG \cdot \overline{(RC + Car)}) \le Close \\ (A3b)\quad \overline{CG} \cdot Car \le \overline{Close} \\ (A4)\quad Open \le \overline{Close} \\ (A6)\quad CG \le \overline{Close} \\ (A7)\quad RC \le \overline{Close} \\ \text{Assumptions: } OG \cdot CG = 0^* \\ \qquad Open = F_1(Car, CG, OG, RC) \end{cases}$

5.4 Solving equations systems

Let us first hear in mind that a solution can be found for system (1) if and only if: $\bar{a} \cdot b = 0^*$. For most systems obtained from control specifications, this condition is not true. As assertions were written independently and come from different, antagonistic concerns (e.g. productivity and safety concerns), there is indeed no reason for obtaining from the start consistent systems, such as: $\bar{a} \cdot b = 0^*$. In the case of the controller studied in this section for instance, signal Open is defined by the initial system:

$$\begin{cases}(\mathrm{RC} + \overline{\mathrm{CG}} \cdot \mathrm{Car}) \le \mathrm{Open} \\ \mathrm{OG} \le \overline{\mathrm{Open}}\end{cases}$$

Proposition 1 implies that there exists a solution if and only if input signals RC, CG, Car and OG are such as:

$$(\mathrm{RC} + \overline{\mathrm{CG}} \cdot \mathrm{Car}) \cdot \mathrm{OG} = 0^*$$

This condition on inputs behaviour is not realistic; the remote control (RC) may be set when the gate is open (OG) for instance: $\mathrm{RC} \cdot \mathrm{OG} \ne 0^*$. Hence the initial system must be modified so as to ensure assertions consistency and to yield a solution whatever the behaviour of these inputs.

Therefore once the initial system of a given output is set up, the proposed synthesis method detects whether this system includes inconsistencies or not by symbolic calculus. When an inconsistency is detected, it is clearly pointed out by exhibiting the two assertions that give rise to inconsistency as well as the inconsistency condition. Then the designer must remove this inconsistency by defining which assertion has priority over the other one (a safety-related assertion has priority over an assertion describing a normal running mode, for instance). Given this information, the lower priority assertion can be modified automatically so as to obtain a consistent set.

The initial system of signal Open includes two inconsistencies:

- between A1 and A5 for condition: $\mathrm{C}_1 = \mathrm{RC} \cdot \mathrm{OG}$.
- between A3a and A5 for: $\mathrm{C}_2 = \overline{\mathrm{CG}} \cdot \mathrm{Car} \cdot \mathrm{OG}$.

To remove these inconsistencies, the designer requires that A5 has priority over A1 and over A3a. These priority rules state that the assertion for stopping the opening movement (A5) is more significant than those specifying normal operation (A1 and A3a), that is usual for that kind of automatic gate whose actuators must be shut down when the stop position is reached. The consistent final system is therefore:

$$\begin{cases}\mathrm{Open} = \mathrm{F}_1(\mathrm{Car}, \mathrm{CG}, \mathrm{OG}, \mathrm{RC}) \\ \quad (\mathrm{A1}) \quad \mathrm{RC} \cdot \overline{\mathrm{OG}} \le \mathrm{Open} \\ \quad (\mathrm{A3a}) \ \ \overline{\mathrm{CG}} \cdot \mathrm{Car} \cdot \overline{\mathrm{OG}} \le \mathrm{Open} \\ \quad (\mathrm{A5}) \quad \mathrm{OG} \le \overline{\mathrm{Open}} \\ \quad \text{Assumption: } \mathrm{OG} \cdot \mathrm{CG} = 0^*\end{cases}$$

whose solution, according to the results of section 4.2, is:

$$\begin{aligned}\mathrm{Open} &= \mathrm{SR}((\mathrm{RC} \cdot \overline{\mathrm{OG}} + \overline{\mathrm{CG}} \cdot \mathrm{Car} \cdot \overline{\mathrm{OG}}), \mathrm{OG}) \\ &= \mathrm{RS}((\mathrm{RC} + \overline{\mathrm{CG}} \cdot \mathrm{Car}), \mathrm{OG})\end{aligned}$$

Remark 2 When an inconsistency cannot be removed by introducing priority levels, one of the assertions that leads to inconsistency (or both) is (are) faulty. Then the designer must rewrite this (these) assertion(s) and check again consistency of the assertions set.

The initial system of signal Close does not include any inconsistency for:

$$\begin{aligned}&(3\mathrm{s}/(\mathrm{OG} \cdot \overline{(\mathrm{RC} + \mathrm{Car})})) \\ &\quad \cdot (\overline{\mathrm{CG}} \cdot \mathrm{Car} + \mathrm{Open} + \mathrm{CG} + \mathrm{RC}) = 0^*\end{aligned}$$

with:

$$\begin{cases}\mathrm{OG} \cdot \mathrm{CG} = 0^* \\ \mathrm{Open} = \mathrm{RS}((\mathrm{RC} + \overline{\mathrm{CG}} \cdot \mathrm{Car}), \mathrm{OG})\end{cases}$$

This result being obtained by symbolic calculus on algebra $\mathbb{I}$. According to the results of section 4.2, the solution is:

$$\mathrm{Close} = \mathrm{SR}((3\mathrm{s}/(\mathrm{OG} \cdot \overline{(\mathrm{RC} + \mathrm{Car})})), (\mathrm{Car} + \mathrm{CG} + \mathrm{RC} + \mathrm{Open}))$$

To sum up, the control laws of this automatic gate are:

$$\begin{cases}\mathrm{Open} = \mathrm{RS}((\mathrm{RC} + \overline{\mathrm{CG}} \cdot \mathrm{Car}), \mathrm{OG}) \\ \mathrm{Close} = \mathrm{SR}((3\mathrm{s}/(\mathrm{OG} \cdot \overline{(\mathrm{RC} + \mathrm{Car})})), (\mathrm{Car} + \mathrm{CG} + \mathrm{RC} + \mathrm{Open}))\end{cases}$$

5.5 Generation of PLC program

This set of formal statements can be easily translated into a Programmable Logic Controller (PLC) program written in a standardized language (IEC, 1993), like Ladder Diagram, a widespread programming language for PLCs (Fig. 9). It matters to highlight that the instruction lines of this program are ordered according to the dependency constraints previously obtained. Output Close, for instance, must be elaborated once output Open is computed.

```
   |                          +----+       |
   |        RC                | RS |  Open |
R1 +--+-----| |------+--------|S  Q|--( )--+
   |  |  CG    Car   |        |    |       |
   |  +--|/|---| |---+        |    |       |
   |       OG                 |    |       |
   +-------| |----------------|R1  |       |
   |                          +----+       |
   |                                       |
   |                 +-----+  +----+       |
   |  OG   RC   Car  | TON |  | SR | Close |
R2 +--| |--|/|--|/|--|IN  Q|--|S1 Q|--( )--+
   |              3s-|PT   |  |    |       |
   |      CG         +-----+  |    |       |
   +--+---| |---+-------------|R   |       |
   |  |   Car   |             +----+       |
   |  +---| |---+                          |
   |  |   RC    |                          |
   |  +---| |---+                          |
   |  |   Open  |                          |
   |  +---| |---+                          |
   |                                       |
```

Fig. 9 PLC program developed from the control laws

Other programming languages can be employed to develop this control program, provided that each operation of algebra $\mathbb{I}$ is translated into a set of instructions of the considered languages.

The obtained program fulfils the set of 8 assertions including initial assertions A2 and A4 to A7 as well as the following two assertions that are modified versions of initial assertions A1 and A3 so as to ensure consistency of the system of equations that defines signal Open:

- When the remote control is activated, the gate opens, if it is not already completely open.
- While the gate is not totally closed, the detection of a car causes the reopening of the gate, if it is not already completely open.

Therefore the presented formal method delivers a set of properties, stated as logic assertions, that hold true for the designed PLC program. Moreover the proof of these properties is formed by the design file that gathers all the symbolic computations leading to the control laws. These major features of the method can strongly benefit when designing controllers for critical systems with high SIL (Safety Integrity Level) (IEC, 1998); for these systems indeed, formal proof of compliance with the application requirements is often asked by customers or certification authorities.

5.6 Generalisation

Assuming that the specifications that describe the expected behaviour of the controller have been translated into formal assertions, the main steps of this algebraic method are summarised below:

1 *Setting up equations systems.* This step yields one system of equations for each output (or internal) signal by gathering all the assertions that contain this signal. A large part of this step is automated, the only role of the designer being to decide which signal is function of the other ones when an assertion contains several outputs or internal signals.

2 *Ordering equations systems.* Systems related to output (or internal) signals that depend only on input signals can be solved in any order, whereas systems related to signals that depend on input signals and on output (or internal) signals must be solved once the formal statements of these latter signals have been obtained, i.e., once their own systems have been solved. Hence the aim of this step is to determine the order in which systems of the second kind must be solved, by analysing automatically the dependency graph issued from the set of assertions. When a cycle is detected within this graph, the designer is asked to tackle this problem by modifying the specifications; in that case, the method must be restarted from the beginning.

3 *Solving equations systems.* Systems are solved one by one, according to the order determined at the previous step. Solving a system implies to check its consistency, to remove inconsistencies if needed, and last to obtain the solution. Consistency checking is performed automatically by a symbolic calculus software that detects whether Proposition 1 is satisfied or not; inconsistencies removal necessitates the designer's competence to modify assertions that give rise to these problems; obtaining the solution of a consistent system consists merely in applying Proposition 2 to this system.

This method has been applied successfully to several case studies. The detailed results of these studies can be found at http://www.lurpa.ens-cachan.fr/isa/asc/.

6. Comparison with a traditional engineering approach

To point out the advantages of this algebraic method, the result obtained will be compared with that obtained with a traditional engineering approach based only on the designer's skill. This traditional approach is assumed to yield a SFC (Sequential Function Chart) program (IEC, 1993). SFC has been chosen because its good structuring features will ease the comparison of the two methods. A similar, but longer, discussion is possible for controllers developed in other programming languages.

In a traditional approach, the designer first identifies the meaningful states of the controlled system ("Closed gate", "Opening gate", "Open gate" and "Closing gate" in this example), then defines the transitions that link these states and the transition conditions (from state "Closed gate", it is possible to move to state "Opening gate" if and only if the remote control is activated, for instance). Then, proceeding by trial and error, the designer can produce a SFC simular to that presented in Fig. 10. Steps 1, 2, 4 correspond respectively to states "Closed gate", "Opening gate" and "Closing gate". State "Open gate" gave rise to two steps (3 and 5), depending on the presence or absence of an opening request (Remote Control or Car detected), so as to facilitate timer management; when the gate is completely open indeed, the timer must count the elapsed time from the beginning of step 3 when there is no request and must be reset if a request occurs. This program seems able to control the gate; nevertheless it includes two significant flaws that come from the engineering approach.

6.1 Initial state

The program presented in Fig. 10 has been designed assuming that the controlled system is initially in a well-defined state: the gate is assumed to be closed when step 1 is active. If this assumption is no longer true, for instance because an operator lets the gate completely open at the end of

```
+----+
|    |
|  +==+==+
|  || 1 ||  (* The gate is closed. *)
|  +==+==+
|    |
|    + RC
|    |
|    +---------------------<-----------------------+
|    |                                             |
|  +--+--+  +--------+                              |
|  |  2  |--| Open   |                              |
|  +--+--+  +--------+                              |
|    |                                             |
|    + OG                                          |
|    |                                             |
|    +---------------------<--------------------+  |
|    |                                          |  |
|  +--+--+                                       |  |
|  |  3  |  (* The gate is open without *)       |  |
|  +--+--+  (* an opening request.      *)       |  |
|    *--------------------+                     |  |
|  2 |                  1 |                     |  |
|    + 3.T>T#3s           + RC OR Car           |  |
|    |                    |                     |  |
|  +--+--+  +--------+  +--+--+ (* Open with  *) |  |
|  |  4  |--| Close  |  |  5  | (* an opening *) |  |
|  +--+--+  +--------+  +--+--+ (* request     *) |  |
|    |                    |                     |  |
|    |                    + NOT (RC OR Car)     |  |
|    *--------------+     |                     |  |
|  2 |            1 |     +---------------------+  |
|    |              |                              |
|    + CG           + RC OR Car                    |
|    |              |                              |
+----+              +------------------------------+
```

Fig. 10 SFC program obtained with traditional approach

a maintenance operation and then restarts the controller from its initial state, the program will no longer run correctly; step 1 will be active while the gate is open and it will be mandatory to activate Remote Control to close the gate, which is quite unusual and hazardous. Conversely the program obtained at the end of the last section is able to control the gate whatever its initial state may be, because it has been obtained from an assertions set that does not include any assumption on the initial state of the controlled system.

Therefore the first advantage of the algebraic method, that is not state-based and therefore that does not require to identify meaningful states, is to yield a result that does not depend on the initial state of the controlled system. This feature is quite interesting when designing controllers that must run correctly after maintenance operations that may leave the controlled physical system in any state.

6.2 Unexpected behaviours

The transitions defined by the designer correspond to the basic evolutions of the program (from step 2 to step 3 once the gate is completely open, from step 3 to step 4 if the gate has been open for 3 seconds, and so on). Nevertheless other evolutions are possible by firing sequentially several transitions without any change of inputs. The SFC can evolve for instance from step 2 to step 5 going through step 3 if OG becomes true while RC is already true, and from step 4 to step 5 going through steps 2 and 3 if RC is true when the gate is still entirely open. Some of these evolutions, like the first one, are without consequence because they do not operate opposite actuators controls; other ones, like the latter evolution mentioned above, can damage actuators because they induce sequences of contradictory controls leading to hazardous energy pulses. These evolutions which must be forbidden are nevertheless difficult to detect and significant efforts must be made to correct them, particularly when dealing with large systems.

By requiring the designer to specify, at the very beginning of design, all the behaviours that he or she expects (e.g., the gate cannot be controlled to close when the remote control is activated) and by providing the formal proof of the compliance of the obtained controller to these requirements, the algebraic method presents the advantage of avoiding tedious, costly and time-consuming detection and correction tasks following usual design based on traditional engineering approaches.

Remark 3 It is possible to represent the overall behaviour of the Ladder Diagram program of Fig. 9 in the form of the SFC shown in Fig. 11 that includes only 3 steps and does not give rise to unexpected behaviours. However the condition transitions of this SFC are more complex than those of the previous model. Hence obtaining this dependable model by using a traditional approach would be an extremely hard task.

```
+----+
|    |
|  +==+==+                              VAR
|  || 1 || (* No mouvement *)             TON1 : TON ;
|  +==+==+                              END_VAR
|    |
|    +----------------------+
|    |                      |
|    + (RC OR               + TON1(
|    |    (Car & NOT CG))   |  IN:=(OG & NOT(RC OR Car)),
|    |  & (NOT OG)          |  PT:= T#3s ).Q
|    |                   +--+--+  +--------+
|    |                   |  2  |--| Close  |
|    |                   +--+--+  +--------+
|    |                      |
|    |     +----------------+----------+
|    |     |                           |
|    |     + (RC OR (Car & NOT CG))    + CG & NOT RC
|    |     | & (NOT OG)                | OR OG & RC
|    +--<--+                           | OR OG & Car
|    |                                 |
|  +--+--+  +--------+                  |
|  |  2  |--| Open   |                  |
|  +--+--+  +--------+                  |
|    |                                 |
|    + OG                              |
|    |                                 |
|    +---------+-----------------------+
|              |
+--------------+
```

Fig. 11 SFC deduced of Ladder program (Fig. 9)

7. Application to an experimental manufacturing line

Any design method should be able to deal with industrial size problems that include numerous variables. To assess scalability of the proposed synthesis method, several case studies were performed in our laboratory. Only the results obtained for an experimental manufacturing line, the example on the largest scale that we dealt with, are given below. The objectives of this section are twofold:

- To provide a methodology to obtain assertions systematically for large systems. This methodology benefits from modularity and genericity principles.
- To demonstrate the ability to design controllers far larger than the one studied in section 5.

7.1 System to control

This system (Fig. 12) is a didactic manufacturing line developed by Bosch company. It is composed of four stations controlled by a single PLC. Table 5 gives the main features of this line as regards control.

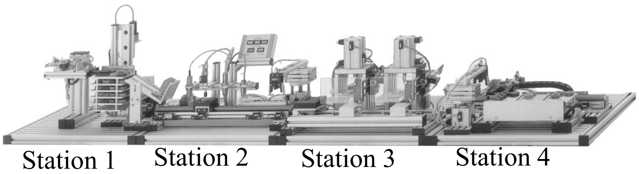


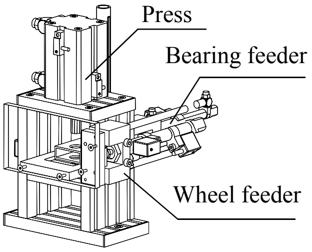
Fig. 12 Structure of the manufacturing line

Table 5:
Design features of the manufacturing line

	Station 1	Station 2	Station 3	Station 4
Inputs	17	20	21	23
Outputs	13	12	13	10
Cylinders	2	4	6	1
Motors	1	1	0	1
Electromagnets	3	0	2	0

The description of the overall operation of this line is out of the scope of this article. The design methodology is illustrated thanks to a device located at station 3: insert device, described in Fig. 13. The scalability of the synthesis method is demonstrated by presenting the results obtained for the global system.

The insert device assemblies plain bearings into gear wheels and can be decomposed into three subsystems: the press, the wheel feeder and the bearing feeder. The press and the wheel feeder are both actuated by a double-acting cylinder with a single-solenoid valve and are fitted with two position sensors. As for the bearing feeder, it is actuated by a double-acting cylinder with a single-solenoid valve and is fitted with only one position sensor.



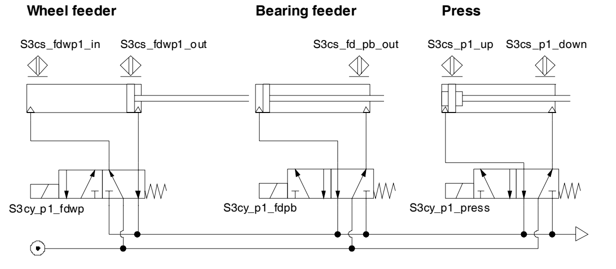


Fig. 13 Structure of the insert device (station 3)

7.2 Methodology

The input data of the synthesis method are specifications assumed to express all the users concerns (normal operation, time constraints, safety concerns, …). Any missing assertion within this specifications set will lead to control laws that will not comply totally with users requirements. Unfortunately obtaining a complete set of assertions is a tedious task for large systems. Our experiments have shown nevertheless that most of assertions could be obtained from generic ones. Hence an actuator-oriented methodology has been set up. With this methodology, assertions may be ranked into three categories:

- assertions coming from technological features,
- assertions developed for collision avoidance,
- assertions describing functionalities of sub-systems.

This approach benefits of modularity and genericity principles advocated by (Vogrig et al., 1987, Gouyon et al., 2004) and implemented by means of behaviour filters. A behaviour filter (filter in what follows) is a low-level controller that is aimed:

- At filtering observations supplied by sensors and at computing reports about the state of physical components to be sent to high-level controllers so that these reports would be consistent with the real behaviour of these components,

- At filtering functional requests submitted by high-level controllers and at computing appropriate signals to be sent to actuators in such a way that these signals comply with the current state of physical components.

Fig. 14 shows how filters may be employed to structure the control system of the insert device. Three filters (one for each actuator) were introduced. For space reasons, focus will be put mainly on the press filter in what follows.

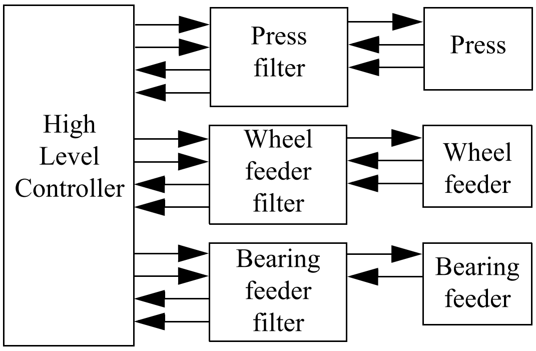


Fig. 14 Behaviour filters of the insert device

7.2.1 *Assertions coming from technological features*

These assertions allow to obtain the control laws of the outputs of filters from sensors signals and from functional requests. As manufacturing lines include numerous similar technological solutions, e.g. double-acting cylinders with single-solenoid valve, electrical motors with two rotating senses, ... , it is of interest to design specific assertions from generic ones. Generic assertions describe the behaviour of a class of technological solutions which is characterized by three parameters:

- kind of power used by the actuator (electric power, compressed air, …),
- kind of preactuator (kind of valve or contactors),
- number and location of position sensors.

The example of the press filter (Fig. 15) will illustrate design of a filter for a technological solution including a double-acting cylinder with single-solenoid valve and two position sensors. Signals S3cs_P1_up and S3cs_P1_down come from the two sensors while signal S3cy_P1_press is used to drive the valve. The other signals come from or go to the high-level controller.

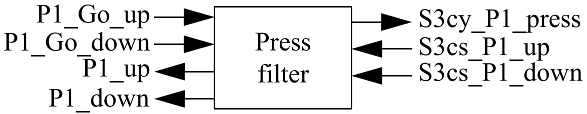


Fig. 15 Filter inputs and outputs

The interface between the filter and the high-level controller is composed of two movement requests (P1_Go_up, P1_Go_down) and of two position information (P1_up, P1_down) (Fig. 15). These latter ones are built from the states of sensors as follows:

$$\begin{cases}\text{P1\_up} = \text{S3cs\_P1\_up}\cdot\overline{\text{S3cs\_P1\_down}}\\ \text{P1\_down} = \text{S3cs\_P1\_down}\cdot\overline{\text{S3cs\_P1\_up}}\end{cases}$$

These equations mean that the states of sensors must be consistent to consider that the upper or lower positions are reached.

Signal S3cy_P1_press is obtained from the system below which means that the valve is driven according to the two movement requests and the two position information:

$$\begin{cases}\text{P1\_up}\cdot\text{P1\_Go\_down}\cdot\overline{\text{P1\_Go\_up}} \le \text{S3cy\_P1\_press}\\ \text{P1\_down}\cdot\text{P1\_Go\_up}\cdot\overline{\text{P1\_Go\_down}} \le \overline{\text{S3cy\_P1\_press}}\end{cases}$$

From previous results, the solution of this system is:

$$\text{S3cy\_P1\_press} = \text{RS}\begin{bmatrix}(\text{P1\_up}\cdot\text{P1\_Go\_down}\cdot\overline{\text{P1\_Go\_up}}),\\ (\text{P1\_down}\cdot\text{P1\_Go\_up}\cdot\overline{\text{P1\_Go\_down}})\end{bmatrix}$$

Moreover two relations between requests and position information can be stated:

$$\begin{cases}\text{P1\_up} \le \overline{\text{P1\_Go\_up}}\\ \text{P1\_down} \le \overline{\text{P1\_Go\_down}}\end{cases}$$

These relations mean that a request is satisfied when the stop position is reached.

7.2.2 *Assertions developed for collision avoidance*

To avoid collisions between mechanical components moved by actuators, safety assertions must be introduced. These assertions are independent from technological features and from functionalities of sub-systems and may be obtained by analysing the common workspaces of sub-systems. For the insert device, the workspaces of the press, of the wheel feeder and of the bearing feeder are shown in Fig. 16.

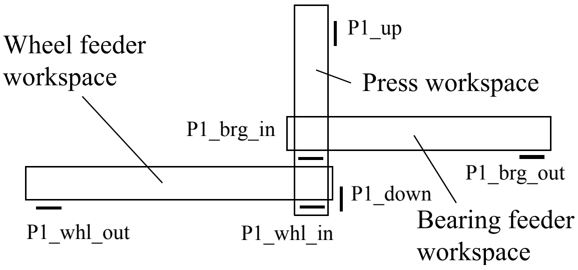


Fig. 16 Workspaces of the three sub-systems

Analysing this figure leads to two assertions for the press:

$$\begin{cases}\text{P1\_Go\_up} \le \text{P1\_brg\_in}\cdot\text{P1\_whl\_in}\\ \text{P1\_Go\_down} \le \text{P1\_brg\_in}\cdot\text{P1\_whl\_in}\end{cases}$$

Meaning that press movements are possible only if the other actuators are correctly located.

Similar assertions can be easily obtained for the other subsystems (wheel feeder and bearing feeder).

7.2.3 *Assertions describing functionalities of sub-systems*

These assertions depend upon the positions of sub-systems and upon the current state of the manufactured parts. For the insert device, six assertions are sufficient to describe normal operation:

A1 The press cylinder moves down when the two parts (wheel and bearing) are correctly located and if these parts have not been assembled previously.

A2 Once the lower position reached, the press cylinder moves up.

A3 The wheel feeder loads a wheel when the wheel-carrying conveyor stands in front of the press and carries a wheel on which no bearing has been assembled.

A4 The wheel feeder unloads the finished part (wheel and bearing assembled) when assembly is finished.

A5 When a bearing is put on the bearing feeder, the bearing feeder loads it into the press.

A6 When the bearing feeder is empty, it comes back to the bearing warehouse (right most position on Fig. 16) so as to take a new bearing.

Only the first two assertions deal with press functionnalities. They give rise to the following formal relations:

$$\begin{aligned}\text{A1}:&(\text{P1\_brg\_in}\cdot\text{P1\_brg\_load})\cdot(\text{P1\_whl\_in}\cdot\overline{\text{P1\_op\_made})}\\ &\le \text{P1\_Go\_down}\cdot\overline{\text{P1\_Go\_up}}\end{aligned}$$

A2 : $\mathrm{P1\_down} \le \mathrm{P1\_Go\_up} \cdot \overline{\mathrm{P1\_Go\_down}}$

where:

- P1_brg_in: position of bearing feeder (see Fig. 16)
- P1_brg_load: bearing feeder with one part (state of manufactured part),
- P1_whl_in: position of wheel feeder (see Fig. 16)
- P1_op_made: Operation made (state of manufactured part)

7.2.4 Control laws generation

These laws are obtained by using the method explained in section 5, from an assertions set that gathers the three kinds of assertions. Our experiments showed that introducing priority levels could be carried out in a systematic way: assertions describing safety concerns having priority over assertions describing normal operation.

The control laws for the press are given below:

$$\begin{cases}
\mathrm{P1\_up} \;=\; \mathrm{S3cs\_P1\_up} \cdot \overline{\mathrm{S3cs\_P1\_down}} \\
\mathrm{P1\_down} \;=\; \mathrm{S3cs\_P1\_down} \cdot \overline{\mathrm{S3cs\_P1\_up}} \\
\mathrm{P1\_Go\_down} \;=\; \mathrm{RS}\begin{bmatrix} (\mathrm{P1\_brg\_load} \cdot \overline{\mathrm{P1\_op\_made}})\ , \\ (\overline{\mathrm{P1\_brg\_in}} + \overline{\mathrm{P1\_whl\_in}} + \mathrm{P1\_down}) \end{bmatrix} \\
\mathrm{P1\_Go\_up} \;=\; \mathrm{RS}\begin{bmatrix} \mathrm{P1\_down}\ , \\ (\overline{\mathrm{P1\_brg\_in}} + \overline{\mathrm{P1\_whl\_in}} + \mathrm{P1\_up} + \\ \qquad (\mathrm{P1\_brg\_load} \cdot \overline{\mathrm{P1\_op\_made}})) \end{bmatrix} \\
\mathrm{S3cy\_P1\_press}{=}\mathrm{RS}\begin{bmatrix} (\mathrm{P1\_up} \cdot \mathrm{P1\_Go\_down} \cdot \overline{\mathrm{P1\_Go\_up}})\ , \\ (\mathrm{P1\_down} \cdot \mathrm{P1\_Go\_up} \cdot \overline{\mathrm{P1\_Go\_down}}) \end{bmatrix}
\end{cases}$$

7.3 Current results

To avoid tedious symbolic calculus and to help the designer during the different steps of this synthesis method, a prototype software tool has been developed in Python. This tool performs all the computations required for variables dependency analysis, inconsistencies detection and control laws generation, that enables the designer to focus only on application-related issues.

This tool was used to design the control system of the manufacturing line previously presented. The results obtained for some devices of this line that exhibit different technological features are given in table 6, where:

- C1 is the horizontal conveyor of station 1. It has three stopping positions and is actuated by an electric motor.
- T1 is the test module of station 2. It is actuated by a pneumatic cylinder and is aimed at detecting whether a bearing is inserted within a wheel or not.
- G1 is the gripper of station 2. This rotary/lift gripper is a pneumatic handling device with two motion axes and a gripper. Wheels are seized, lifted, and repositioned after a swivelling movement of up to 180°.
- C2 is the horizontal conveyor of the station 3. It has four stopping positions and is actuated by a pneumatic cylinder. Positioning is performed with contactless sensors and electromagnetically operated mechanical stops.
- P1 is the insert press of station 3 (presented before).
- G2 is the gripper of station 4, similar to gripper G1.

Table 6:
Results obtained on the manufacturing line

Number of	C1	T1	G1	C2	P1	G2	Ttl
Sub-systems	1	1	3	3	3	3	14
Control laws	10	4	14	16	14	14	72
Assertions	31	10	38	47	38	38	202
- technological features	21	4	24	29	24	24	126
- collision avoidance	3	2	6	3	5	6	25
- functionalities	7	4	8	15	9	8	51
Priorities	21	5	21	29	21	21	118

Among the 202 assertions necessary to specify the behaviours of these devices, 126 (62 %) were obtained automatically from knowledge of technological features, 25 (12 %) come from collision avoidance analysis and only 51 (26 %) describe devices functionalities. All priorities were elaborated automatically by considering that safety assertions have priority over other assertions.

The 72 control laws were automatically generated and translated into a program in Structured Text (IEC, 1993). This program was implemented on the Schneider-Electric PLC controlling the line. This program has been tested successfully. It matters to highlight that translation of control laws into an executable program was possible because the prototype tool is able to generate all data mandatory to execute a program on a PLC (e.g. mnemonic tables).

This tool generates too an analysis file that gathers:

- the list of controller signals,
- the list of variables dependencies (in textual form as shown in table 3 or in graphical form as depicted in Fig. 8),
- and, for each control law, the relations set that specifies its behaviour, the priorities introduced to obtain a consistent set as well as the formal solution on II .

8. Conclusion

The increasing productivity constraints as well as the need for compliance with standards for functional safety (IEC 61508) explain that designing dependable logic controllers becomes a more and more crucial concern for companies in many industrial domains. Formal methods can efficiently contribute to reach this objective by tackling out the usual problems of specifications inconsistency and of specifications misinterpretation leading to design faults.

The formal synthesis method presented in this article covers the whole design phase, from specifications to control laws that can be implemented into a PLC. Its ability to deal with large enough problems has been demonstrated that allows to consider application to industrial size problems, provided that the actuator-oriented methodology for assertions elicitation is used.

The Boolean algebra that underlies this method provides an integrating framework for specifications formalization and analysis as well as for control laws generation by solving equations systems. Using a single formalism guarantees the soundness of the synthesis method by avoiding semantics changes that may give rise to formal incompleteness and inconsistency.

The case studies that we achieved have yielded several investigation prospects. First the development of new operations and relations of algebra $I\!I$ is intended so as to increase the capability for expressing natural language sentences that describe specifications. Then, new solving techniques will be investigated to obtain control laws from assertions including these new operations and relations.

References

Bérard, B., et al. (1999). *Systems and Software Verification: Model-Checking techniques and tools,* Heidelberg, Springer-Verlag Editions.

Cassandras, C.G. & Lafortune S. (1999). *Introduction to Discrete Event Systems,* Boston, Kluwer Academic Publishers.

De Smet, O. & Rossi O. (2002). Verification of a controller for a flexible manufacturing line written in Ladder Diagram via model-checking, in *Proceedings of 21st American Control Conference, ACC'02*, (pp. 4147-4152).

Gouyon, D., et al. (2004). Pragmatic approach for modular control synthesis and implementation. *International Journal of Production Research*, 42(14), 2839-2858.

Grimaldi, R.P. (2000). *Discrete and Combinatorial Mathematics: An Applied Introduction*, New-York, Addison-Wesley Editions, 4th edition.

International Electrotechnical Commission (1993). IEC 61131-3*, Programmable controllers - programming languages*.

International Electrotechnical Commission (1998). IEC 61508*, Functional safety of electrical/electronic/electronic programmable safety-related systems.*

Klein S. et al. (2003). Designing fault-tolerant controllers using Model-Checking, in *Proceedings of IFAC-Safeprocess 2003*, Washington, USA, (pp. 115-120).

Nourelfath, M. & Niel, E. (2004). Modular supervisory control of an experimental automated manufacturing system. *Control Engineering Practice*, 12(2), 205-216.

Ramadge, P.J. & Wonham W.M. (1989). The control of discrete-event systems. *IEEE Transations on Automatic Control*, 77(1), 81-97.

Roussel, J.-M., & Denis B. (2002). Safety properties verification of ladder diagram programs, *Journal Européen des Systèmes Automatisés*, Hermès Editions, 36(7), 905-917.

Roussel, J.-M., & Faure J.-M. (2002). An algebraic approach for PLC programs verification. In: *Proceedings of 6th International Workshop on Discrete Event Systems (WODES'02)*, Zaragoza, Spain, 2-4 October, 2002, pp. 303-308.

Roussel, J.-M., et al. (2004). Algebraic approach for dependable logic control systems design. *International Journal of Production Research*, 42(14), 2859-2876.

Volgrig, R., et al. (1987). Flexible manufacturing system shop. *Manufacturing Systems*, 16, 43-55.

Wang, Y., (2000). *Supervisory control of Boolean discrete-event systems*, M. A. Sc. Thesis, Department of Electrical and Computer Engineering, University of Toronto.

Zaytoon, J., & Carré-ménétrier, V. (2001). Synthesis of control implementation for discrete manufacturing systems. *International Journal of Production Research*, 39(2), 329-345.