CONTROL SYNTHESIS BASED ON SAFETY BOOLEAN GUARDS FOR MANUFACTURING SYSTEMS: APPLICATION TO A SORTING SYSTEM

Bernard Riera, Raphael Coupat, David Annebicque, Philippot Alexandre, Francois Gellot

Control synthesis based on safety Boolean guards for manufacturing systems: application to a sorting system

B. RIERA*, R. COUPAT*, D. ANNEBIQUE, A. PHILIPPOT*, and F. GELLOT***

** CReSTIC (EA3804), UFR Sciences Exactes et Naturelles, Reims University (URCA), Moulin de la Housse, BP 1039, 51687 Reims - France (bernard.riera@univ-reims.fr).*
*** CReSTIC (EA3804), IUT de Troyes, 9 rue de Québec, BP 396, 10026 TROYES, Cedex, France*

Abstract: This paper presents an original approach of safe control synthesis for manufacturing systems controlled by Programmable Logic Controller (PLC). In this work, manufacturing systems are considered as Discrete Event Systems (DES) with logical Inputs (sensors) and logical Outputs (actuators). The proposed approach, which separates the functional control part from the safety control part, is easy to implement and guarantees that the designed controller is safe. The methodology is based on the use of safety constraints in order to design a safe permissive controller. This controller is then constrained by functional constraints. The approach is illustrated with a sorting boxes simulated process using the ITS PLC software from the Real Games Company (www.realgames.pt). The control algorithm is presented in details in the paper. This approach can be used with an existing PLC program in order to guarantee its safety. However, it also allows to result in a safe control, may be simpler than a conventional approach based on a complete specification for instance in GRAFCET (IEC 60848) that does not distinguish the functional aspect from the safety aspect.

Keywords: Discrete-Event Systems, Control, Safety, Programmable Logic Controllers, Manufacturing Systems.

1. INTRODUCTION

This paper presents an original approach of control synthesis for manufacturing systems controlled by PLC (Programmable Logic Controller). In this work, manufacturing systems are considered as Discrete Event Systems (DES) (Cassandras *et al.* 1999) with logical Inputs (sensors) and logical Outputs (actuators). This is an extension of the research work that the CReSTIC (Research Centre in Information and Communication Science and Technologies) has led for several years on the definition and design of guard conditions placed at the end of the PLC program which act as a logic filter in order to be robust to control errors. In previous work, these safety constraints (formally checked off line by using a model checker (Marangé *et al.* 2010)) stop on line the process in a safe state if at least one of them is violated. This idea has been extended to propose a safe control design pattern based on safety logical constraints. This approach, which separates the functional control part from the safety control part, is easy to implement and involve a new way to design the controller. The methodology is based on the use of safety constraints in order to get the most permissive safe controller allowed by the set of safety constraints. This controller is then constrained by functional constraints while respecting the safety constraints. This paper proposes several improvements of the control algorithm presented in (Riera *et al.* 2012a, 2012b) particularly in the management of Safety Combined Constraints (no infinite loop even if failures appear in the process). The approach is illustrated by using one example: a virtual sorting system using the ITS PLC software from the Real Games Company (www.realgames.pt). The main idea of this approach comes from the fact that the process is not stopped if one or several safety constraints are violated. The controller continues to work with safe outputs values. This approach to PLC programming makes safety a priority and allows for a controller to create a safe environment where functional and safety aspects are clearly separated. Consequently, this control synthesis approach allows to result in a safe control, that can be simpler than a conventional approach based on a complete specification in GRAFCET (IEC 60848) that does not distinguish the functional aspect from the safety aspect.

2. BOOLEAN SAFETY CONSTRAINTS FOR ROBUST PLC CONTROL

Since a PLC is a dedicated controller it will only process this one program over and over again. One cycle through the program is called a scan time and involves reading the inputs (i) from the other modules, executing the logic based on these inputs and then updated the outputs (o) accordingly. The memory in the CPU stores the program while also holding the status of the I/O and providing a means to store values. A controller at each PLC scan time has to compute the outputs values (controllable variables) based on inputs (uncontrollable variables) and internal memories. The notations used in the following of this paper are:

- t: current scan time (from PLC point of view), *t-1* previous PLC scan time.
- $o_k = o_k(t)$: logical variable corresponding to the value of k^{th} PLC Boolean output (actuator) at t. Outputs at t are considered as the one and only

variables that can be controlled (write variables) at each PLC scan time. All other PLC variables (inputs, previous outputs, ⋯) are uncontrollable (read only variables).

- $o_k^* = o_k(t-1)$: logical variable corresponding to the value of k^{th} PLC Boolean output (actuator) at time *t-1* (previous PLC scan time).
- "$.$", "$+$", "$\overline{\ }$" are respectively the logical operators AND, OR, and NOT.
- 0 means FALSE and 1 means TRUE.
- $\sum$ and $\prod$ are respectively the logical sum (OR) and the logical product (AND) of logical variables.
- $\sum\prod$ is a logical polynomial (sum of products expression also called SIGMA-PI).
- $\uparrow x$ means rising edge of Boolean variable x (in the PLC, $\uparrow x = \overline{x^*}.x$).
- $\downarrow x$ means falling edge of Boolean variable x (in the PLC, $\downarrow x = x^*.\bar{x}$).
- O: set of output variables at *t*
- Y : set of uncontrollable variables at *t, t-1, t-2* ...
- N_o : number of PLC Boolean outputs
- N_{CSs}: number of Simple Safety Constraints
- N_{CSc}: number of Combined Safety Constraints

The proposed methodology to design safe controllers is based on the use of logical safety constraints, which act as logical guards placed at the end of the PLC program, and forbid sending unsafe control to the plant [Marangé *et al.* 2010]. The set of safety constraints acts as a control filter.

Constraints (or guards) are always modeled with the point of view of the control part (PLC), and it is assumed that the PLC scan time is sufficient to detect any change of the input vector (synchronous operation, possible simultaneous changes of state of PLC inputs). In addition, the plant is considered functioning normally without failure.

It is considered in this work that the initial safe state for all the actuators (o_k) is defined to be 0. The constraints have to be defined in order to keep the system controllable. This means that, even with the set of safety constraints, it is possible to design a controller which matches the specifications. For example, considering the previous hypothesis about the safe initial state, a set of safety constraints which resets at each scan time all outputs is safe but does not ensure the controllability. Some guards involve a single output at time *t* (called simple safety constraints *CSs*), other constraints involve several outputs at time *t* (combined safety constraints *CSc*). Constraints require the knowledge of I/O at the current time *t* and possibly previous times (presence of edge (*t-1*) for instance). Safety constraints are not always depending only on PLC inputs at *t*. It may be necessary to define supplementary uncontrollable variables called observers. Observers are memories enabling to get a combinatory constraint (Riera *et al.* 2011).

The set of safety constraints is considered as necessary and sufficient to guarantee the safety. In this approach, it is assumed that the safety constraints can always be represented as a monomial and depend on the inputs (at *t, t-1, t-2*…), outputs (at *t, t-1, t-2*…) and observers (depending ideally on only inputs at *t, t-1, t-2*…). In the initial methodology (Marangé *et al.* 2010), the control filter is validated offline by model checking (Behrmann *et al.* 2002) and stops the process in a safe state if a safety constraint (*CSs* and *CSc*) is violated.

In this paper, *CSs* and *CSc* are represented (equations (1) and (2)) as logical monomial functions ($\prod$, products of variables but not necessarily minterms) which have always to be FALSE at the end of each scan time, before updating the outputs, in order to guarantee the safety. It is important to note that each *CSs* depends only on one controllable event (output: o_k) and that each *CSc* depends on several controllable events (outputs: o_k, o_l, …).

$$\forall m \in [1, N_{CSs}],\ \exists!\, k \in [1, N_O] \,/\, CSs_m = \prod(o_k, Y) = 0 \quad (1)$$

$$\forall n \in [1, N_{CSc}],\ \exists!\, (k, l, \dots) \in [1, N_O]\ with\ k \neq l \neq \cdots \,/\, CSc_n = \prod(o_k, o_l, \dots, Y) = 0 \quad (2)$$

There are only 2 exclusive forms of simple safety constraints (*CSs*) because they are expressed as a monomial function, and they only involve a single output at time *t* (equation (3) or (4)):

$$\forall m \in [1, N_{CSs}],\quad \exists!\, k \in [1, N_O] \,/\, CSs_m = o_k.h_{0m}(Y) \quad (3)$$

xor

$$CSs_m = \overline{o_k}.h_{1m}(Y) \quad (4)$$

These simple safety constraints (*CSs*) express the fact that if $h_{0m}(Y)$, which is a monomial (product) function of only uncontrollable variables at *t*, is TRUE, o_k must be necessarily FALSE (equation (3)) in order to keep the constraints equal to 0. If $h_{1m}(Y)$) is TRUE, o_k must be necessarily TRUE (equation (4)).

For each output, it is possible to write equation (5) corresponding to a logical OR of all simple safety constraints.

$$\sum_{i=1}^{N_{CSs}} CSs_i = \sum_{k=1}^{N_o} \left(f_{sk}(o_k, Y)\right) = 0 \quad (5)$$

$f_{sk}(o_k, Y)$ is a logical $\sum\prod$ function independent of the other outputs at *t* because only *CSs* are considered. $f_{sk}(o_k, Y)$ can be developed in equation (6) where f_{s0k} and f_{s1k} are polynomial functions (sum of products, $\sum\prod$) of uncontrollable (read only) variables. Equation (6) has always to be FALSE because all simple safety constraints must be FALSE at each PLC scan time.

$$f_{sk}(o_k, Y) = o_k.f_{s0k}(Y) + \overline{o_k}.f_{s1k}(Y) = 0 \quad (6)$$

From equation (5) and taking into account all *CSs*; it is possible to write equation (7).

$$\sum_{i=1}^{N_{CSs}} CSs_i = \sum_{k=1}^{N_o} \left(o_k.f_{s0k}(Y) + \overline{o_k}.f_{s1k}(Y)\right) = 0 \quad (7)$$

It is important to note that the simple safety constraints have to respect the following mathematical property (equation 8):

$$f_{s0k}(Y).f_{s1k}(Y) = 0 \quad (8)$$

Indeed, if it is not the case, that means that 2 *CSs* are in contradiction and one of both is necessarily not verified, thus the set of constraints is not coherent. One can notice that if f_{s0k} =0 or if f_{s1k} =0, the property is logically verified. In addition, the following proposition can be written:

Proposition: if all simple safety constraints implying output o_k are only based on the rising edge and falling edge of the output o_k, the property (8) is true (sufficient condition).

Proof: if all *CSs* implying o_k, all are only based on rising edge and falling edge, one can notice using the Shannon expansion theorem that:

$$f_{s0k}(Y) = \overline{o_k^*}.f_{s0k}(Y) \text{ and } f_{s1k}(Y) = o_k^*.f_{s1k}(Y) \quad (9)$$

Consequently, because $\overline{o_k^*}.o_k^* = 0$, and the initial state supposed by hypothesis being safe, the property (8) is verified.

$$f_{s0k}(Y).f_{s1k}(Y) = \overline{o_k^*}.f_{s0k}(Y).o_k^*.f_{s1k}(Y) = 0 \quad (10)$$

3. SAFE CONTROL SYNTHESIS FROM LOGICAL CONSTRAINTS

The control algorithm proposed separates safety requirements from functional requirements. As already noticed, a set of safety constraints is considered as necessary and sufficient. In other words, if only one safety constraint is removed, the system is unsafe. All other constraints that can be added are considered as functional constraints because they don't act on safety. The control algorithm proposed consists at each PLC scan time in authorizing functional requirements which are compatible with safety requirements. In order to present the idea, let's consider a system without *CSc*.

3.1 Taking into account the CSs and Functional specification

It is possible to define, like the *CSs*, the functional constraints (*FC*) of o_k indicating when it should be equal to 0 (g_{0k}) or when the output o_k should be equal to 1 (g_{1k}) (equation 11) from a functional point of view. In this paper only simple functional constraints *FCs* are considered.

$$g_k(o_k, Y) = o_k.g_{0k}(Y) \text{ or } g_k(o_k, Y) = \overline{o_k}.g_{1k}(Y) \quad (11)$$

Generally the specifications indicate when the output must be activated and therefore g_{1k}. These g_{1k} can of course include observers obtained from GRAFCET steps (IEC 60848) or SFC (IEC 61131-3) or be assigned with the calculated outputs from an existing PLC program. Hietter (2008) in his work about algebraic synthesis of dependable logic controllers proposed a parametric solution of the equation 6. Indeed, it is possible to write (equation 12) the parametric solution (called o'_k), where p is a Boolean parameter.

$$o'_k = \overline{f_{sok}}.p + f_{s1k} \quad (12)$$

In order to integrate the *FCs*, p has to be chosen equal to 1 and the solution becomes equation (13):

$$o'_k = \overline{f_{sok}}.g_{1k} + f_{s1k} \text{ or } o'_k = \overline{f_{sok}}.\overline{g_{ok}} + f_{s1k} \quad (13)$$

The control obtained is safe (if there are only *CSs*) because the safety is ensured regardless of the *FCs*. Indeed, if the *FCs* try to impose an output to 0, in contradiction with the safety, the term f_{s1k} continues to provide safety. Therefore the functional part can be designed without considering safety, what makes the job much easier for the control engineer.

A basic example is going to illustrate that point. Suppose the truth table, represented by a Karnaugh map, from the figure 1 where the output S'_k (controllable variable) depends on 4 inputs (uncontrollable variables) *a*, *b*, *c*, *d*. For each input vector, the output is indicated. It can be either 0 or 1 or an undetermined value (0 or 1) called "don't care" conditions. A "don't care" condition is a combination of inputs for which the designer does not care what the output is. Some of the 0 and the 1 come from safety aspects (respectively f_{s0} and f_{s1}). The others 0 and 1 come from functional requirements. From that, usually one can express the simplified output S'_k (see equation (14)) by regrouping the 1 (or the 0).

$$S'_k = b.c.\bar{d} + a.c.d \quad (14)$$

S'k	ab 00	01	11	10
cd 00	0	0 (f0)	0 (f0)	X (0 or 1)
01	0	0 (f0)	0 (f0)	X (0 or 1)
11	0	0 (f0)	1 (f1)	1 (f1)
10	0	1	1	0

Fig. 1. Simple logical control example

From the Karnaugh map, it is also possible to express the simplified expression of the output S'_k by using f_{s1k}, f_{s0k} and g_{1k} (equation (15)). It is essential to note that the one and only condition for g_{1k} is to include as a minimum, all the "1" coming from the functional requirements (figure 2).

$$\begin{cases} f_{s1k} = a.c.d \\ f_{s0k} = b.\bar{c} + \bar{a}.b.d \\ g_{1k} = b \end{cases} \quad (15)$$

S'k	ab 00	01	11	10
cd 00	0	0 (f0)	0 (f0)	X (0 or 1)
01	0	0 (f0)	0 (f0)	X (0 or 1)
11	0	0 (f0)	1 (f1)	1 (f1)
10	0	1	1	0

Fig. 2. Example with correct functional specifications

In this example, g_{1k} represents a set of 8 values. The result (equation (16)) is the same as obtained according to equation (14), and the functional requirement g_{1k} is simple to express.

$$S'_k = \overline{f_{sok}}.g_{1k} + f_{s1k}$$
$$S'_k = \overline{(b.\bar{c} + \bar{a}.b.d)}.b + a.c.d = b.c.\bar{d} + a.c.d \quad (16)$$

One of the most attractive points is that, even if g_{1k} is wrongly expressed (equation (17), figure 3), the calculation of S'_k returns a safe value. This means that even if the functional requirements are wrong the system remains safe.

$$\begin{cases} f_{s1k} = a.c.d \\ f_{s0k} = b.\bar{c} + \bar{a}.b.d \\ g_{1k} = \bar{a} \end{cases} \quad (17)$$

S'k	ab 00	01	11	10
cd 00	0	0 (f0)	0 (f0)	X (0 or 1)
01	0	0 (f0)	0 (f0)	X (0 or 1)
11	0	0 (f0)	1 (f1)	1 (f1)
10	0	1	1	0

Fig. 3. Example with wrong functional specifications

In the next part of the paper, it is shown how to deal with the combined safety constraints (*CSc*). We will only consider in the following of the paper, *FCs* defined by g_{1k}.

3.2 Taking into account the CSc

The problem with *CSc* seems to be more complex. Indeed, when a *CSc* is not verified, it is necessary to give the priority to one or several outputs and to be compliant with *CSs*. In addition, when one *CSc* is solved, it can involve problems

with other *CSc*. Taking into account these points, and using equation (13), it is possible to write equation (18).

$$o_k = \overline{f_{sok}}.\left(\overline{f_{c0k}}.g_{1k} + f_{c1k}\right) + f_{s1k} \quad (18)$$

f_{c0k} and f_{c1k} force the output o_k respectively to 0 or 1 taking into account *CSc*. It is supposed *CSc* have to be designed in order to give always the same priority to outputs. What the reader has to notice, it is that during the PLC scan time, a safe value of o_k has to be found. This means that the value of o_k has to be compliant with all *CSc* implying o_k. If f_{c0k} and f_{c1k} are badly defined, a safe value of o_k can be impossible to compute. To illustrate this problem, let's take a simple example. Suppose the 2 following *CSc* (equation 19):

$$CSc_1 = o_1.\overline{o_2};\ CSc_2 = o_2.\overline{o_3} \quad (19)$$

If when CSc_1 is TRUE the priority is given to o_1 and when CSc_2 is TRUE the priority is given to o_3, if o_1=1 and o_3=0, it is impossible to find a safe value of o_2. We propose here a simple solution to detect this problem. The idea is to check that during the PLC scan time one *CSc* is not violated 2 times. That will be the case if after having tried to find a solution (N_{CSc}+1) times, you do not get a solution. Hence, this means there is a problem of definition of *CSc*. In this case, the priority has to be given to *CSs*. Even if safety constraints are formally checked before implementation, this problem can occur if a failure appears in the process.

Let's define $\overrightarrow{f_{c0}}$ and $\overrightarrow{f_{c1}}$ as column vectors representing respectively the *k* values of f_{c0k} and f_{c1k}. $\overrightarrow{f_{c0}}$ and $\overrightarrow{f_{c1}}$ can be obtained through 2 matrices *MC0* and *MC1* that the control engineer has to define during the initial safety analysis stage to indicate the priority between outputs. *MC0* and *MC1* are matrices with N_{CSc} columns and N_o lines and indicate for each *CSc*, if the outputs ($\vec{O}$) have to be forced respectively to 0 or 1. Using the matrix logical product, one can write equations (20 and 21).

$$\overrightarrow{CSc} = \begin{pmatrix} CSc_1 \\ \cdots \\ CSc_{N_{CSc}} \end{pmatrix}, \text{ column vector of } CSc$$

$$\vec{O} = \begin{pmatrix} o_1 \\ \cdots \\ o_k \\ \cdots \\ o_{N_o} \end{pmatrix}, \text{ column vector of outputs } o_k$$

$$\overrightarrow{f_{c0}} = \begin{pmatrix} f_{c01} \\ \cdots \\ f_{c0N_o} \end{pmatrix} = MC0.\overrightarrow{CSc}$$

$$\overrightarrow{f_{c0}} = \begin{pmatrix} MC0_{11} & \cdots & MC0_{1N_{CSc}} \\ \cdots & \cdots & \cdots \\ MC0_{N_o1} & \cdots & MC0_{N_oN_{CSc}} \end{pmatrix}.\begin{pmatrix} CSc_1 \\ \cdots \\ CSc_{N_{CSc}} \end{pmatrix} \quad (20)$$

$$\overrightarrow{f_{c1}} = \begin{pmatrix} f_{c11} \\ \cdots \\ f_{c1N_o} \end{pmatrix} = MC1.\overrightarrow{CSc}$$

$$\overrightarrow{f_{c1}} = \begin{pmatrix} MC1_{11} & \cdots & MC1_{1N_{CSc}} \\ \cdots & \cdots & \cdots \\ MC1_{N_o1} & \cdots & MC1_{N_oN_{CSc}} \end{pmatrix}.\begin{pmatrix} CSc_1 \\ \cdots \\ CSc_{N_{CSc}} \end{pmatrix} \quad (21)$$

Figure 4 presents the algorithm which is detailed in order for the reader to be able to implement it in a PLC in ST language (IEC 611131-3).

```
// g1k are calculated previously (functional constraints, FC) in the PLC program. MC0 and MC1 for the CSc are known
// Each ok, fs0k, fs1k are calculated at each scan PLC
// check that the CSs respect fs0k.fs1k = FALSE
// init fc0k and fc1k
Flag_CSs = FALSE
For k=1 to No
        Flag_CSs = Flag_CSs + fs0k.fs1k
        fc0k = False // INIT
        fc1k= False //INIT
End for
Flag = not Flag_CSs
Cpt =0 // counter for the CSc

While (Flag and Cpt<NCSc)
        // each ok is calculated using ok = fsok.(fc0k.g1k + fc1k) + fs1k
        // olk is the intermediary value of ok
        For k=1 to No
                olk = fsok.(fc0k.g1k + fc1k) + fs1k
        End For
        // check if a CSc is violated
        Flag = FALSE
        For i=1 to NCSc
                Calculate CSci by using olk values
                Flag = Flag + CSci
        End For
        Cpt= Cpt +1
        // if Flag =TRUE, priority is given to a ok using MC0 and MC1
        If Flag Then
                For k=1 to No
                        fc0k = FALSE
                        fc1k = FALSE
                        For j=1 to NCSc
                                fc0k = fc0k + MC0kj.CScj
                                fc1k = fc1k + MC1kj.CScj
                        End For
                End For
        End If
End While
If Flag_CSs Then
        print "PROBLEM BAD DEFINITION CSs"
        Break // STOP with problem
End If
If (cpt= =NCSc) Then
        print "PROBLEM BAD DEFINITION CSc"
        // in case of bad definition of CSc, ok are set
        // to a safe value, with a priority to FALSE
        For k=1 to No
                olk = fsok.fs1k
        End For
End If
// The outputs are set with safe values
For k=1 to No
        ok = olk
End For
```

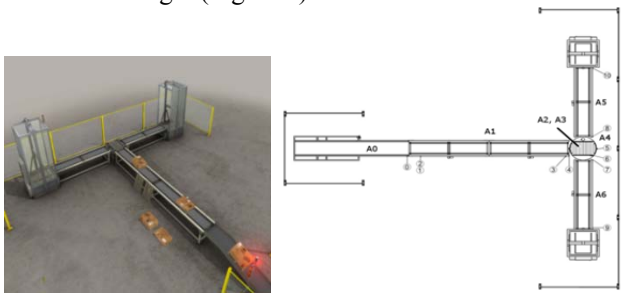
Fig. 4. Safe controller algorithm.

This algorithm is quite original because one can see there is a WHILE structure inside the PLC program in order to manage the *CSc*. This is something that could not be seen as a good practice for PLC programmer using LADDER. This algorithm is simple and the program structure (i.e. control design pattern) is always the same whatever the system to be controlled and its specification. It can also be used with an existing PLC program. Even if the functional constraints are wrong, the system remains safe. In addition, if some safety constraints become incoherent there is no problem of infinite loop with the WHILE structure. This guarantees that the PLC watchdog will never be set even if there are failures in the process. The algorithm starts with an initialization stage

where f_{c0k} and f_{c1k} are assigned to 0. After, the WHILE loop is started. The first calculation of equation (18) supplies a set of outputs respecting only the *CSs* because f_{c0k} and f_{c1k} are set initially to 0. After, the *CSc* are calculated. If one of them is not respected, new values of f_{c0k} and f_{c1k} using MC0 and MC1 are calculated and the WHILE loop is repeated. This process is repeated until the set of outputs respects the set of safety constraints (*CSs* and *CSc*) or if the number of iterations is greater than the number of *CSc*.

4. EXAMPLE ON A SORTING SYSTEM

The control algorithm will be illustrated by the mean of a virtual system from the ITS PLC collection, proposed by the Portuguese company Real Games. ITS PLC collection is a set of simulation software dedicated to automation training (Riera *et al.* 2009). Demos and technical descriptions of the five virtual industrial systems are available and freely downloadable at web address www.realgames.pt. As part of the work presented in this paper, the "sorting system" is used. The objective of this system is to transport boxes from entry conveyor to exit conveyor by sorting them according to for instance their height (Figure 5).



Inputs (Sensors):
C0: Feeder belt exit detector, C1: Lower case detector, C2: Higher case detector, C3 Exit detector of the entry conveyor, C4-C5: Detectors of the turntable position, C6: Turntable pallet detector, C7: Entry detector of the left exit conveyor, C9: Exit detector of the left exit conveyor, C8: Entry detector of the right exit conveyor, C10: Exit detector of the right exit conveyor
Outputs (Actuators): A0: Feeder belt, A1: Entry conveyor, A2: Turntable rollers (loading), A3: Turntable rollers, A4: Turntable, A5: Left exit conveyor, A6: Right exit conveyor

Fig. 5. Virtual sorting system from ITS PLC collection

The system is instrumented using 11 sensors to determine the size of the boxes (small or large) and the entry or exit of a box in different conveyors (feeding, intermediate, evacuation) or turntable. The seven outputs of the PLC can activate the various conveyors and the turntable. The specification used is as follows. After pressing the "start" button, the boxes are sent successively one to the left elevator and one to the right elevator. After pressing the "stop" button, boxes in transit are evacuated. The safety analysis has resulted in 17 *CSs* (equation (22)) and 5 *CSc* (equation (23)), formally checked using the UPPAAL model checker (Behrmann *et al.* 2002) and the methodology proposed in (Riera *et al.* 2011, Marangé *et al.* 2010). With this set of safety constraints (*CSs* and *CSc*), whatever the controller, the collisions between boxes and falling down of boxes, are avoided (figure 6). Explanation about *CSs* and *CSc* can be found in (Benlorhfar *et al.* 2011). This set of constraints ensure the controllability (there is at least one controller allowing to bring boxes to the left elevator and the right elevator), and the safety regardless of the control. It should be noted that these constraints are permissive (large control space allowed) but require five observers (*2P*, *P36*, *P67, P79 and P810*). This example is interesting because the separation of safety and functional aspects simplifies a lot the control design. Indeed, from a functional point of view, the problem consists in only deciding if the box goes to the right or to the left.

Fig. 6. Unsafe situations avoided

P36, *P67*, *P79*, *P810* are observers which allow knowing that a box is respectively present between the sensors *C3* and *C6*, *C6* and *C7*, *C7* and *C9*, and *C8* and *C10* (sensors excluded). For example, *P36* is set to 1 on the falling edge of the sensor *C3* and reset to 0 on a rising edge of the sensor *C6*. In this system, the distance between the sensors *C0* and *C1* is smaller than the size of a box. The observer *2P* (figure 8) indicates if *C0=C1=1*, 2 boxes are present and not only one (figure 7).

$$CSs1 = 2P.A0; CSs2 = C3.\overline{C4}.A1; CSs3 = C3.C4.C6.A1$$
$$CSs4 = C3.P36.A1;\ CSs5 = \overline{C5}.A3;\ CSs6 = C4.C6.A2$$
$$CSs7 = \overline{C4}.\overline{C5}.A2;\ CSs8 = C5.C6.\overline{A4};\ CSs9 = C5.C8.\overline{A4}$$
$$CSs10 = C5.C7.\overline{A4};\ CSs11 = C5.P67.\overline{A4}$$
$$CSs12 = C5.C7.A2;\ CSs13 = C4.C9.A4$$
$$CSs14 = C4.\overline{C6}.A4;\ CSs15 = C4.\mathrm{P79}.A4;$$
$$CSs16 = C4.\mathrm{P810}.A4;\ CSs17 = C4.C10.A4 \quad (22)$$
$$CSc1 = C0.A0.\overline{A1};\ CSc2 = C3.C4.A1.\overline{A2}$$
$$CSc3 = C5.C8.A2.\overline{A5};\ CSc4 = C5.C7.A3.\overline{A6}$$
$$CSc5 = \mathrm{A2}.A3 \quad (23)$$

2P = 0 *2P* = 1

Fig. 7. Observers *2P*

Concerning *CSc*, following the path of boxes, *A2* has priority over *A1*, and *A1* has priority over *A0* (*CSc1=1* implies *A0=0*, *CSc2=1* implies *A1=0*). *A5* and *A6* have priority over *A2* and *A3* (*CSc3=1* implies *A2=0*, *CSc4=1* implies *A3=0*). At least, when *A2* = *A3* = 1, there is no priority, *A2* and *A3* are reset to *0* (*CSc5=1* implies *A2=A3=0*). The specification of the functional part is presented figure 8 using GRAFCET (IEC60848) which is easy to implement in one of the PLC languages (IEC 61131-3). The variable *cpt_conv1* is a counter which indicates the number of boxes on the entry conveyor (controlled by *A1*). *PC* is an observer whose value is complemented on a falling edge of the sensors *C7* or *C8*, and allows directing the boxes to the left elevator or the right elevator. One can notice that a complete specification in GRAFCET is much more difficult to get and to read because safety and functional aspects have to be mixed. One can also note, that theoretically the motion of the turntable must be maintained in steps 14 and 15. This will be managed by the safety guards. Now it is possible to write f_{s0k}, f_{s1k} g_{1k}, *MC0* and *MC1* for each output from the *CSs* (equation 24). The control algorithm has been

implemented successfully in a PLC with version ITS PLC PE using a PLC M340 from Schneider Electric.

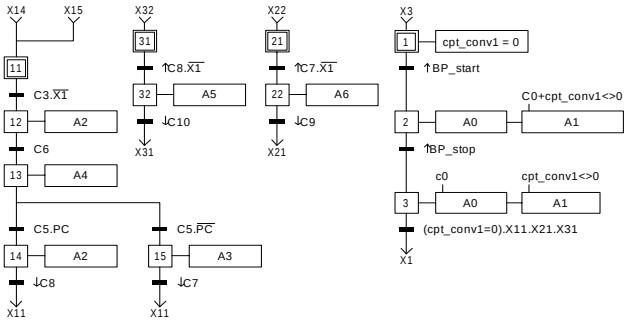


Fig. 8. Functional specification of the sorting system

$$A0\begin{cases} f_{s00} = 2P \quad f_{s10} = 0 \\ g10 = X2 + X3.C0 \end{cases}$$

$$A1\begin{cases} f_{s01} = C3.\overline{C4} + C3.C4.C6 + C3.P36 \\ f_{s11} = 0 \\ g_{11} = X2.\left(C0 + (cpt_{conv1} <> 0)\right) + X3.cpt_{conv1} <> 0 \end{cases}$$

$$A2\begin{cases} f_{s02} = C4.C6 + \overline{C4}.\overline{C5} + C5.C7 \quad f_{s12} = 0 \\ g_{12} = X12 + X14 \end{cases}$$

$$A3\begin{cases} f_{s03} = \overline{C5} \quad f_{s13} = 0 \\ g_{13} = X15 \end{cases}$$

$$A4\begin{cases} f_{s04} = C4.\overline{C6} + C4.P79 + C4.P810 + C4.C9 + C4.C10 \\ f_{s14} = C5.C6 + C5.C8 + C5.C7 + C5.P67 \\ g_{14} = X13 \end{cases}$$

$$A5\begin{cases} f_{s05} = 0 \quad f_{s15} = 0 \\ g_{15} = X32 \end{cases}$$

$$A6\begin{cases} f_{s06} = 0 \quad f_{s16} = 0 \\ g_{16} = X22 \end{cases}$$

$$\mathrm{MC0} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \quad \mathrm{MC1} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \qquad (24)$$

5. CONCLUSION

This paper proposed a control synthesis method based on the use of safety guards (represented as a set of logical constraints which can be simple or combined). The result is a control design pattern easy to implement in a PLC. If the safety constraints are well defined and eventually formally checked, the programmed controller is safe even if the functional constraints are wrong because only one control respecting the safety constraints is allowed. Contrary to SCT approach (supervisory control theory (Ramadge *et al.* 1989)), the algorithm has been designed to be implemented in a PLC. The separation of "safety" and "functional" aspects allows interesting perspectives, like better process performances and flexibility, easier management of several operating modes linked to a Manufacturing Execution System (MES) or simpler management of the manual modes through Human-Machine Interfaces (HMI) or Supervisory Control and Data Acquisition (SCADA) systems. In addition, the prospects of this work also seem to be important because the obtained results could change the "traditional" way to design controllers of automated production system.

ACKNOWLEDGEMENT

This work is integrated in the frame of the CPER project MOSYP and is supported by Région Champagne-Ardenne, FRANCE.

REFERENCES

Behrmann G., Bengtsson J., David A., Larsen K.G., Pettersson P., Yi W. (2002). Uppaal implementation secrets. *7th International Symposium on Formal Techniques in Real-Time and Fault Tolerant Systems*.

Benlorhfar R., Annebicque D., Gellot F., Riera B. (2011). Robust filtering of PLC program for automated systems of production, *18th World Congress of the International Federation of Automatic Control* , Milano, Italy, august.

Cassandras C. G., Lafortune S., (1999). *Introduction to discrete event systems*. Boston, MA: Kluwer Academic Publishers.

Hietter Y., Roussel J.-M., Lesage J.-J. (2008). Algebraic synthesis of dependable logic controllers *17th IFAC World Congress, Seoul (Korea)*, pp. 4132-4137, July.

IEC INTERNATIONAL STANDARD 60848 (2002), Second edition 2003-01, Programmable controllers – Part 3: Programming languages GRAFCET specification language for sequential function charts Reference number CEI/IEC 60848: 2002.

IEC INTERNATIONAL STANDARD 61131-3 (2003), Second edition 2002-02, GRAFCET specification language for sequential function charts Reference number CEI/IEC 61131-3: 2003.

Marangé P., Benlorhfar R., Gellot F., Riera B. (2010). Prevention of human control errors by robust filter for manufacturing system, *11th IFAC/IFIP/IFORS/IEA Symposium on Analysis, Design, and Evaluation of Human-Machine Systems*, Valenciennes, France.

Ramadge G., Wonham W. M. (1989). The control of discrete event systems, Proc. IEEE, *Special issue on DEDSs*, 77, pp.81-98.

Riera B., Annebicque D., Gellot F., Philippot A. and Benlorhfar R. (2012a). Control synthesis based on logical constraints for safe manufacturing systems, *14th IFAC Symposium on INformation COntrol problems in Manufacturing (INCOM 2012)*, Bucarest, Romania, mai 2012.

Riera B., Benlorhfar R., Annebicque D., Gellot F. , Vigario B., (2011). Robust control filter for manufacturing systems: application to PLC training, *18th World Congress of the International Federation of Automatic Control* , Milano, Italy.

Riera B., Gellot F., Philippot A., Vigario B., et D. Annebicque (2012b). Synthèse de commande sure de fonctionnement à base de contraintes logiques pour les systèmes manufacturiers, *9th International Conference on Modeling, Optimization & SIMulation (MOSIM'2012)*, Bordeaux, France, juin 2012.

Riera B., Marangé P., Gellot F., Nocent O., Magalhaes A., Vigario B. (2009). Complementary usage of real and virtual manufacturing systems for safe PLC training, *8th IFAC Symposium on Advances in Control Education (ACE'09)*. Kumamoto, Japan.