Safe PLC Controller Implementation IEC 61131-3 Compliant based on a Simple SAT Solver: Application to Manufacturing Systems

Romain Pichard, Alexandre Philippot and Bernard Riera
CReSTIC (EA3804), University of Reims Champagne Ardenne Moulin de la Housse, BP 1039, 51687 Reims CEDEX 2, France

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

Abstract: In this study, manufacturing systems are considered as Discrete Event Systems (DES) with logical Inputs (sensors) and logical Outputs (actuators). In previous work, an original implementation of safe controllers (using safety logical constraints) for manufacturing systems, based on the use of a CSP (constraint satisfaction problem) solver, was proposed. However, the proposed solution was not IEC 61131-3 compliant. In other words, it was not possible to implement it in a PLC (Programmable Logic Controller). In this paper, a proof of concept IEC 61131-3 compliant has been carried out. To perform this challenge, an original simple CSP - SAT solver in ST (Structured Text) has been developed and programmed. The algorithm has been tested and validated by using a M340 Schneider Electric PLC and a box sorting simulated process using the FACTORY I/O software from the Real Games Company (www.realgames.co). It seems to be the first time that a SAT solver developed for PLC, is used in real time as a part of a PLC program to get a safe controller.

1 INTRODUCTION

In this work, manufacturing systems are considered as Discrete Event Systems (DES) (Cassandras et al., 1999) with logical Inputs (sensors) and logical Outputs (actuators). The proposed approach for control synthesis separates the functional control part from the safety control part. The methodology is based on the use of safety constraints or guards placed at the end of the PLC program which act as a logic filter in order to be robust to control errors. Safety and functional requirements, separately defined, provide an intuitive and natural way to represent the safety constraints as well as a means to simplify the definition of functional aspects (Zaytoon and Riera, 2017). The safety requirements are expressed as logic functions to set/reset the PLC outputs. These logic functions should be formally checked offline to verify their sufficiency (Marangé et al., 2010) and their consistency (Pichard et al., 2017). However, this approach cannot guarantee deadlock-freeness. Furthermore, since the safety aspects have priority over the functional aspects, the execution of the resulting PLC program may not be compliant to the functional specifications when they violate the safety constraints (Pichard et al., 2018).

In a previous paper (Pichard et al., 2016), an original implementation of this safe control synthesis approach based on the use of a CSP (Constraint Satisfaction Problem) solver was proposed. The principle consisted of, at each scan time, to get all outputs vectors respecting the set of safety constraints and to select the closest, in the sense of Hamming distance, from the functional outputs vector. The proof of concept was performed using a soft PLC developed in Python, which was not IEC 61131-3 standard compliant.

In this paper, we propose a PLC implementation IEC 61131-3 compliant. It is based on the development, of a simple CSP solver in ST (Structured Text). This work introduces the possibility to control manufacturing systems by constraint programming.

The first part of the paper is dedicated to the concept of Boolean guards for safe PLC program. In the second part, the definition and mathematical formalism used for the safety guards are detailed then, it is shown that the problem is a SAT problem

Pichard, R., Philippot, A. and Riera, B.
Safe PLC Controller Implementation IEC 61131-3 Compliant based on a Simple SAT Solver: Application to Manufacturing Systems.
DOI: 10.5220/0006885502310239
In *Proceedings of the 15th International Conference on Informatics in Control, Automation and Robotics (ICINCO 2018) - Volume 1*, pages 231-239
ISBN: 978-989-758-321-6

(SATisfiability). The third part presents the SAT solver algorithm developed for PLC. At last, an experimental platform using a real PLC and a virtual plant (sorting system) is used to validate the approach.

It seems to be the first time that a SAT solver algorithm developed for PLC, is used in real time as a part of a PLC program to get a safe controller.

2 BOOLEAN GUARDS FOR SAFE PLC PROGRAM

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 (input scan), executing the logic based on these inputs (logic scan) and then updated the outputs (O) accordingly (output scan). 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 use of a memory map enables to guarantee that all the calculations are performed with inputs values which are not modified during a PLC scan time. Outputs update is performed with the last outputs calculation in the PLC program. These three basic stages of operations (input scan, logic solve and output scan) are repeated at each scan time.

The idea proposed by (Marangé et al., 2010) is to place a logic filter between the logic solve and the output scan. The goal of this filter is to detect and compensate control errors (Figure 1).

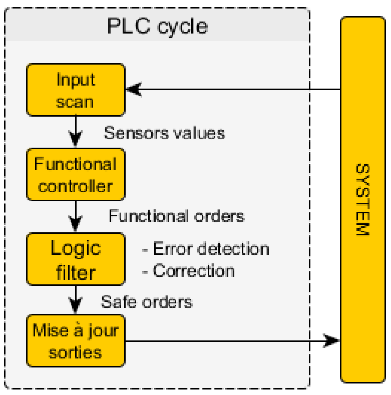

Figure 1: Principle of the logic filter.

Three use cases can be thought of doing with the logic filter: safe blocking, supervisor, and controller (Riera et al., 2015). In the first case, when a safety constraint is violated, the controller is frozen in a safe state which is supposed known. The supervisory approach consists in correcting the control errors without blocking the controller. This enables for instance to safe existing PLC program without changing the code. The controller approach is similar to the supervisor approach. The main difference is that in the design of the controller, it is taken into account by the designer that the safety part is managed by the safety constraints. Hence, there is a separation between functional and safety aspects of the controller. In addition, even if the functional part is badly defined, the system remains safe (Riera et al., 2015). Contrary to the supervisor approach, the fact to violate a safety constraint can be seen as normal behavior of the controller. This last approach modifies the way to design a PLC program but presents several advantages (tasks synchronization, management of running modes, connection to a Manufacturing Execution System …). However, control design based on logical constraints involves 2 main difficulties:

1) Constraints definition and validation which are not always easy to manage. We suppose in this paper that the designer has got a correct set of safety constraints.
2) The proposal of a control algorithm which defines, when one or several constraints are violated, a safe outputs vector compliant with all the safety constraints.

We have already proposed several algorithms to compute at each PLC scan time a safe outputs vector (Pichard et al., 2016, Pichard et al., 2018). One of them is based on CSP to perform the detection and the correction stages. The main advantage of this approach is that it is not necessary to define priority between outputs when a constraint is violated. However, this approach has not yet been implemented and tested with a PLC.

3 BOOLEAN SAFETY CONSTRAINTS FORMALISM

The notations used in this paper are based on the Boolean algebra and PLC programming. 0 means *False* and 1 means *True*. Σ and Π are respectively the logical sum (OR) and the logical product (AND) of logical variables. $\Sigma\Pi$ is a logical polynomial (sum of products expression also called SIGMA-PI). ".", "+", "⊕" "¯" are respectively the logical operators AND, OR, XOR and NOT. *t* is the current scan time (from PLC point of view), *t-1* is the previous PLC scan time. $o_k = o_k(t)$ is the logical variable corresponding to the kth variables at the tth PLC scan time. 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 is the set of output variables at *t*. Y is the set of uncontrollable variables at *t*, *t-1*, *t-2*… N_o is the PLC Boolean outputs number. N_{CSs} is the Simple Safety Constraints number. N_{CSc} is the Combined Safety Constraints number.

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 (or guards) acts as a control filter. 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*). 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.

In this approach, it is assumed that the safety constraints can always be represented as a monomial and depend on the inputs (at time *t, t-1, t-2…*), outputs (at time *t, t-1, t-2…*) and observers (depending ideally on only inputs (at time *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 (∏, logical products of variables but not necessarily minterms) which have always to be *False* at the end of each PLC 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 variable (output: o_k) at time *t* and that each *CSc* depends on several controllable variables (outputs: $o_k, o_l,…$) at time *t*.

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

$$\forall n \in [1, \mathrm{N_{CSc}}],$$
$$\exists!\,(k, l, \dots) \in [1, \mathrm{N_o}]\ avec\ \ k \neq l \neq \cdots /$$
$$CSc_n = \prod(o_k, o_l, \dots, \mathrm{Y}) \tag{2}$$

To guarantee the safety, *CSs* and *CSc* must be *False* (=0) in the PLC program before updating outputs, the logical sum of safety constraints computed with all o_k has to be *False* (equation 3). It is the detection function of the logic filter. A PLC program can be considered as safe if, for the outputs vector $(o_1, \dots, o_k, \dots, o_{No})$, equation (3) is verified before output scan.

$$\sum_{i=1}^{N_{CSs}} CSs_i + \sum_{\mathrm{j}=1}^{N_{CSc}} CSc_{\mathrm{j}} = 0 \tag{3}$$

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 (4)):

$$\forall m \in [1, \mathrm{N_{CSs}}], \qquad \exists!\, k \in [1, \mathrm{N_O}] \ /$$
$$CSs_m = o_k \cdot h_{0m}(\mathrm{Y}) + \overline{o_k} \cdot h_{1m}(\mathrm{Y})$$
$$\text{with } h_{0m}(\mathrm{Y}) \oplus h_{1m}(\mathrm{Y}) = 1 \tag{4}$$

These simple safety constraints (*CSs*) express the fact that if $h_{0m}(\mathrm{Y})$, which is a monomial (product) function of only uncontrollable variables at *t*, is *True*, o_k must be necessarily *False* in order to keep the constraints equal to 0. If $h_{1m}(\mathrm{Y})$ is *True*, o_k must be necessarily *True*. In addition, it is not possible to have $h_{0m}(\mathrm{Y})$ and $h_{1m}(\mathrm{Y})$ true simultaneously. For each output, it is possible to write equation (5) corresponding to a logical OR of all simple safety constraints.

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

$f_{sk}(o_k, \mathrm{Y})$ is a logical ΣΠ function independent of the other outputs at *t* because only *CSs* are considered. $f_{sk}(o_k, \mathrm{Y})$ can be developed in equation (6) where f_{s0k} and f_{s1k} are polynomial functions (sum of products, ΣΠ) of uncontrollable (read-only) variables. Equation (6) has always to be *False* because all simple safety constraints must be *False* at the end of each PLC scan time. To simplify equations, a logical function can be represented by a logical variable having the same name.

$$f_{sk}(o_k, Y) = o_k \cdot f_{s0k}(Y) + \overline{o_k} \cdot f_{s1k}(Y)$$
$$f_{sk} = f_{sk}(o_k, Y) = o_k \cdot f_{s0k} + \overline{o_k} \cdot f_{s1k} \tag{6}$$

From equations (5) and (6), it is possible to write equation (7).

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

The outputs vector can be considered as safe at the end of the PLC scan time if equation (8) is checked.

$$\sum_{k=1}^{N_o} f_{sk} + \sum_{j=1}^{N_{CSc}} CSc_j = 0 \tag{8}$$

One can notice that we have got a set of safety constraints and a formalism which is compliant with a constraints satisfaction problem (CSP) to find a safe outputs vector. To be more precise, it is a Boolean satisfiability problem (sometimes called propositional satisfiability problem and abbreviated as SATISFIABILITY or SAT (Vizel *et al.*, 2015)). The problem consists of determining if there exists an interpretation that satisfies a given Boolean formula. In other words, it asks whether the variables of a given Boolean formula can be consistently replaced by the values *True* or *False* in such a way that the formula evaluates to *True*. If this is the case, the formula is called satisfiable. On the other hand, if no such assignment exists, the function expressed by the formula is *False* for all possible variable assignments and the formula is unsatisfiable. For example, the formula "NOT a AND NOT b" is satisfiable because one can find the values a = *False* and b = *False*, which make (NOT a AND NOT b) = TRUE. In contrast, "b AND NOT b" is unsatisfiable.

4 SAFE PLC CONTROLLER BASED ON A SIMPLE SAT SOLVER

CSP are mathematical problems defined as a set of objects whose state must satisfy a number of constraints or limitations (Hooker, 2000) (Krzysztof, 2003) (Tsang, 1993). CSP represent the entities in a problem as a homogeneous collection of finite constraints over variables, which is solved by constraint satisfaction methods. CSP are the subject of intense research in both artificial intelligence and operations research, since the regularity in their formulation provides a common basis to analyze and to solve problems of many seemingly unrelated families. CSP often exhibit high complexity, requiring a combination of heuristics and combinatorial search methods to be solved in a reasonable time. Formally, in this work, a constraint satisfaction problem is defined as a triple:

$O = \{o_1, \dots, o_n\}$ is the set of outputs variables,

$D = \{True, False\}$ is a set of the respective domains of values,

$C = \{CSs_1, \dots, CSs_{N_{CSs}}, CSc_1, \dots, CSc_{N_{CSc}}\} = \{C_1, \dots, C_{N_{CSs}+N_{CSc}}\}$ is the set of simple safety constraints and combined safety constraints.

Each variable o_i can take a value in the nonempty domain $\{True, False\}$. Every constraint Cr_k is, in turn, a pair $\langle t_j, R_j \rangle$ where $t_j \subset X$ is a subset of k variables and R_j is an k-ary relation on the corresponding subset of domains $\{True, False\}$. An evaluation of the variables o is a function from a subset of variables to a particular set of values in the corresponding subset of domains. An evaluation v satisfies $\langle t_j, R_j \rangle$ if the values assigned to the variables t_j satisfies the relation R_j. An evaluation is consistent if it does not violate any of the constraints. An evaluation is complete if it includes all variables. An evaluation is a solution if it is consistent and complete.

A CSP solver, at each PLC scan time, can supply all the safe output vectors based on the safety constraints. From these, in our approach, we select the first one which is the closest from the functional output vectors. For that, the Hamming distance is used. In information theory, the Hamming distance between two strings of equal length is the number of positions at which the corresponding symbols are different. In another way, it measures the minimum number of substitutions required to change one string into the other, or the minimum number of errors that could have transformed one string into the other. This heuristic is simple and seems appropriate. Indeed, if the Hamming distance is null, this means that the functional outputs vector is safe and can be updated. If the Hamming distance is different from 0, this means that the functional outputs vector is not safe. One can suppose that the functional part of the controller performed by the expert is not out of sense. Hence, selecting the safe outputs vector which has got the smallest Hamming distance from the functional one enables to select the closest outputs vector has got the maximum chance to achieve the production (i.e. functional) goals. Of course, that will work if the functional part of the controller is partially correctly designed. However, whatever the functional part (even if it is really badly designed), the system will remain safe.

5 IMPLEMENTATION IN A PLC

Today, PLC does not include CSP solver. In a previous paper (Pichard et al., 2016), a soft PLC in IronPython was used to preliminary test the idea and to get a proof of concept. We used the package "logilab-constraint", an open source constraint solver written in pure Python with constraint propagation algorithms. The proposed control algorithm calculated at each scan all the safe outputs vectors and selects the one with the minimum Hamming distance compared to the Functional Output Vector (*FOV*). The control algorithm based on CSP has been implemented successfully, with no problem of time calculation. However, it is important to test the concept with real PLC. For that, it is necessary to develop a SAT solver in ST (Structured Text) compliant with the IEC 61131-3. This development seems possible because the structure of the safety constraints is known and simple (monomial), moreover only 1 solution is required. In addition, because of the structure of manufacturing systems (subsystems interconnected), the number of *CSc* violated at each PLC scan time is low. In this paper, we propose a simple SAT solver algorithm which can be implemented easily in ST, whatever the PLC brand in order to satisfy the Boolean safety constraints problem.

5.1 The Proposed Hamming-based SAT Solver

Classical SAT solver algorithm is based on recursive functions. However, it is not possible to perform a recursive program in a PLC by using languages from IEC 61131-3 standard. In addition, it is necessary to avoid to trigger the PLC watchdog.

The objective of the proposed SAT solver algorithm (Algorithm 1) is to test, at each PLC scan time, the functional outputs vector (array *FOV*) given by the PLC program (cf. Figure 1). If at least 1 constraint is violated by FOV, new values of actuators must be computed (array *solution*). The sensors values and internal variables values are grouped in vector I, this vector is given as entry of the algorithm to compute the constraints values.

The main idea of the proposed algorithm is to find the closest values to FOV values (i.e. changing minimum values' number of FOV). This is carried out by using the Hamming distance. Indeed, all the possible vectors with an increasing Hamming distance are computed, then as soon as a vector solved every constraint, this vector is used as the solution and applied to the outputs values.

In order to improve the algorithm efficiency, 2 vectors are tested simultaneously: the *closest* (minimum Hamming distance) and the *farthest* (maximum Hamming distance). The farthest is computed by complementing the closest's values. If the closest solved the problem, the algorithm is stopped and the closest is used as the solution. Else, if the farthest solved the constraints, it is memorized. With this approach, the computation time is almost divided by 2. At last, if no closest vector has solved the constraints, the latest farthest vector is used as the solution. Indeed, the last memorized farthest vector has the minimum hamming distance.

Algorithm 1: Hamming-based SAT solver algorithm.

```
function HammingFilter(Boolean[] FOV, Boolean[] I)
    Compute the values of FS0 and FS1;
    solution := Initialize the solution vector by applying the FS to the vector FOV;
    Test the CSc with solution
    If CSc are not solved
        index := Find the index of the free actuators
        For k = 1 to dim(index)/2
            closest := Compute the first closest k-subset
            Test the CSc with closest
            If CSc are not solved
                farthest := Compute the first farthest k-subset by inversing closest
                Test the CSc farthest
                farthestSolution := Store the farthest if it solved the CSc

Repeat

                    If a new k-subset exists
                        closest := Compute the next closest k-subset
                        Test the CSc with closest
                        If CSc are not solved
                            farthest := Compute the next farthest k-subset by inversing closest
                            Test the CSc with farthest
                            farthestSolution := Store the farthest if it solved the CSc
                        endif
                        Until the problem is solved or all the k-subset are tested
                    endif
                    If closest solved the CSc
                        solution := closest //Use closest as solution
                        exit
                    endif
        endfor
        If the closest doesn't solve the CSc
            If the farthest solved the CSc
                solution := farthest //Use farthest as solution
            Else
                solution := Compute the worst case by complementing the values in FOV of the free variables
            endif
        endif
    endif
    return solution
end
```

We proposed in the next section (section 5.2) an implementation of the proposed algorithm in Structured Text language.

5.2 Implementation in ST (Structured Text, IEC 61131-3)

The implementation respects the algorithm previously presented. Hence, the algorithm stops as soon as a solution (array of Boolean: *mem*) is found. The entry of the algorithm is vector I (sensors and internal variables values) and the functional outputs vector (array *FOV*). The set and reset functions *F0s* and *F1s* are computed by using the I values. At each scan time, from the number of output variables (*No*), the number of output variables that can be modified to solve the set of *CS*c is determined (*Noc*). This is done through the subroutine *initCSC* where the result (*GG*) is an array of integers) which indicates outputs variables implied in violated *CSc*. An array (*tabMot2*) of integers with a size of *Noc* stores the index of these output variables. For generating all combinatorial combinations of *tabMot2*, incrementing the Hamming distance, we have adapted an algorithm found in (Cameron, 1994). For instance, if one considers a word of 3 bits, corresponding respectively to 3 output variables that can be changed, the sequence, where each object is represented by the array *HamMot*, will generate in this order:

- Hamming distance of 1: 100, 010, 001
- Hamming distance of 2: 110, 101, 011
- and finally, Hamming distance of 3: 111.

For instance, let's suppose that *No*=5 (5 outputs: O0, O1, O2, O3, O4), *Noc* =3, with *tabMot2*[0]=1 (corresponding to O1), *tabMot2*[1]=3 (corresponding to O3) and *tabMot2*[2]=4 (corresponding to O4). If *HamMot*[0]=0, HamMot[1]=1 and HamMot[2]=1, this means that the solution inverting O3 and O4 is going to be tested. In addition, in this case *GG*[0]=*False*, *GG*[1]=*True*, *GG*[2]=*False*, *GG*[3]=*True* and *GG*[4]=*True*.
The 2 subroutines *test_CSC* and *calcul_CSC* respectively test if a CSC is violated, and calculate the CSC. As already noticed, in order to improve the algorithm performance speed, 2 solutions: *k* and *Noc-k* Hamming distances are calculated (arrays *mem* and *membis1*) at each loop.

```
FOR k := 0 TO (No-1) DO
    mem[k] := NOT F0s[k] AND FOV[k] OR F1s[k];
    membis1[k]:= NOT F0s[k] AND NOT FOV[k] OR F1s[k];
END_FOR;
tabMot:=mem;
calcul_CSC();
test_CSC();
IF Flag THEN
    Flagbis:=TRUE;
    initCSC();
    Noc := 0;
    FOR i := 0 TO (No-1) DO
        IF NOT F0s[i] AND NOT F1s[i] and GG[i] THEN
            tabMot2[Noc]:=i;
            Noc:=Noc+1;
        END_IF;
    END_FOR;
    maxHam:=Noc;
    FOR k := 1 to DIV(Noc,2) do
        Flag1:=TRUE;
        FOR ii := 0 TO k-1 DO (* First k-subset *)
            HamMot[ii]:=TRUE;
        END_FOR;
        FOR ii := k TO Noc-1 DO
            HamMot[ii]:=FALSE;
        END_FOR;
        FOR i := 0 TO (Noc-1) DO  (* test first k-subset *)
            IF HamMot[i] THEN
                mem[tabMot2[i]]:=NOT tabMot[tabMot2[i]];
            ELSE
                mem[tabMot2[i]]:=tabMot[tabMot2[i]];
            END_IF;
        END_FOR;
        calcul_CSC();
        test_CSC();
        IF Flag and (maxHam>Noc-k) THEN
            FOR i := 0 TO (Noc-1) DO  (* test first k-subset *)
                IF HamMot[i] THEN
                    mem[tabMot2[i]]:=tabMot[tabMot2[i]];
                ELSE
                    mem[tabMot2[i]]:=NOT tabMot[tabMot2[i]];
                END_IF;
            END_FOR;
            calcul_CSC();
            test_CSC(); (* bis*)
            IF NOT Flag THEN
                Flagbis:=FALSE;
            membis1:=mem;
            maxHam:=Noc-k;
            Flag:=TRUE;
        END_IF;
      REPEAT  (* Next k-subset *)
cpt := 0;
            FOR i:=0 TO Noc-2 DO
            IF HamMot[i] THEN
               cpt:=cpt+1;
               IF NOT HamMot[i+1] THEN EXIT;
               END_IF;
            END_IF;
            END_FOR;
             IF i=Noc-1 THEN flag1:=FALSE;
             ELSE
                HamMot[i]:=FALSE;
                HamMot[i+1]:=TRUE;
FOR j := 0 TO i-1 DO
                   HamMot[j]:=FALSE;
                           END_FOR;
                 WHILE (cpt>1) DO
                    HamMot[cpt-2]:=TRUE;
                    cpt:=cpt-1;
                 END_WHILE;
                 FOR i := 0 TO (Noc-1) DO
                     IF HamMot[i] THEN
                        mem[tabMot2[i]]:=
                                   NOT
tabMot[tabMot2[i]];
                      ELSE
                   mem[tabMot2[i]]:=
                                   tabMot[tabMot2[i]];
                       END_IF;
                  END_FOR;
                  calcul_CSC();
                             test_CSC();
                  IF    Flag    and    (maxHam>Noc-k)
THEN                      FOR i := 0 TO (Noc-1) DO
                        IF NOT HamMot[i] THEN
                     mem[tabMot2[i]]:=
                                      NOT
tabMot[tabMot2[i]];
                        ELSE
                                 mem[tabMot2[i]]:=

tabMot[tabMot2[i]];                       END_IF;

                     END_FOR;
                     calcul_CSC();
                     test_CSC(); (* bis*)
                     IF NOT Flag THEN
Flagbis:=FALSE;
                 membis1:=mem;
                 maxHam:=Noc-k;
                        Flag:=TRUE;
                            END_IF;
                         END_IF;
END_IF;
                          UNTIL NOT Flag OR NOT Flag1
END_REPEAT;
                   END_IF;
                   IF NOT Flag THEN EXIT; END_IF;
         END_FOR;
         IF Flag THEN
            IF  NOT  Flagbis  THEN  mem:=membis1;

            ELSE
                FOR i:= 0 TO (Noc-1) DO
                    mem[tabMot2[i]]:=NOT
tabMot[tabMot2[i]];
                   END_FOR;
                   calcul_CSC();
                   test_CSC();
                   IF Flag THEN
        FOR i := 0 TO (No-1) DO
            mem[i] := NOT F0s[i] AND F1s[i];
         END_FOR;
                   END_IF;
              END_IF;
         END_IF;
     END_IF;
     (* update outputs with mem*)
```

Figure 2: simple SAT solver in ST.

The control algorithm has been implemented in a real PLC and tested by the mean of a virtual system from the software FACTORY I/O

5.3 Sorting System Application

FACTORY I/O (https://factoryio.com/) is a new generation of 3D factory simulation for learning automation technologies. It integrates most of the features described in the paper "Virtual systems to train and assist control applications in future factories" (Riera and Vigario, 2013). Designed to be easy to use, it allows to quickly build a virtual factory using a selection of common industrial parts. FACTORY I/O also includes many scenes inspired by typical industrial applications ranging from beginner to advanced difficulty levels. We propose in this paper to use the same benchmark as in the previous paper (Pichard et al., 2016): the sorting system. The main goal of the "sorting system" is to transport and sort cardboard boxes by height using a turntable (Figure 3).

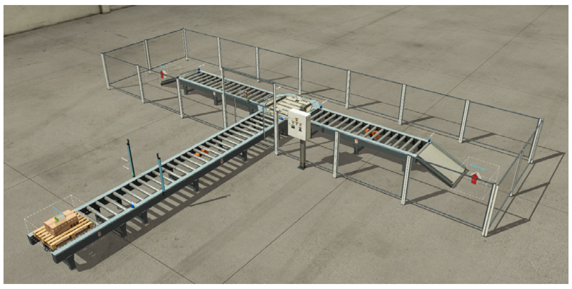

Figure 3: Sorting system from FACTORY I/O.

The descriptions of sensors, actuators and safety constraints used for this example are presented in the previous paper (Pichard et al., 2016).
The control algorithm based on CSP has been successfully implemented in a real M340 PLC. The connection between the PLC and FACTORY I/O is performed using USB I/O DAQ (cf. Figure 4). With this device, the PLC does not see difference between real and virtual plant.

We did not have any problem with time calculation and a scan time of 5 ms was respected for the PLC. In this example, with the functional part of the controller, the maximum Hamming distance is 2, and the time to execute the SAT solver algorithm is always less than 1 ms.

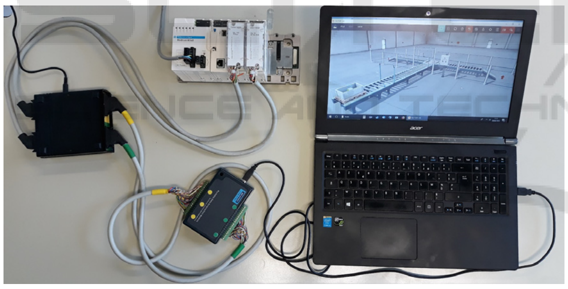

Figure 4: Experimental platform with M340 PLC, FACTORY I/O and USB DAQ Advantech 4750.

6 CONCLUSION

This paper has proposed an implementation of a safe control synthesis method based on the use of safety guards (represented as a set of logical constraints which can be simple or combined) with a SAT solver developed in ST (Structured Text) compliant with the IEC 61131-3 standard for PLC. 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. The algorithm has been successfully tested with a real M340 PLC and a virtual sorting system. The controller code is efficient. However, even if the controller is safe, it is not deterministic and it has to be proved that the minimum Hamming distance compared to the functional output vector is suitable in the sense of the specification of the functional control. It seems to be the first time that, a controller based on the use in real time of a SAT solver, is implemented in a real PLC. Even if the idea of using a SAT solver in a PLC presents several advantages, the proposed control methodology is very different from the "traditional" way to design controllers of the automated production system. However, it seems interesting to the control of cyber physical systems (CPS) in the framework of Industry 4.0.

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*. In Springer, *Verlag London, UK 2002*: 3-22.

Cameron, P. J. *Combinatorics: Topics, Techniques, Algorithms*. Cambridge University Press, 1994 (reprinted 1996). ISBN 0521457610.

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

Hooker J (2000). *Logic-Based Methods for Optimization - Combining Optimization and Constraint Satisfaction.* Wiley-Interscience series in discrete mathematics and optimization. John Wiley and Sons, 2000.

IEC INTERNATIONAL STANDARD 61131-3 (2003). *Programmable controllers – Part 3: Programming languages.* Reference number CEI/IEC 61131-3: 2003.

Krzysztof A. (2003). *Principles of Constraint Programming*. Cambridge University Press, ISBN: 0521825830, New York, NY, USA.

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.

Pichard, R., Rabah, N. B., Carre-Menetrier, V., & Riera, B. (2016). CSP solver for Safe PLC Controller: Application to manufacturing systems. *IFAC-PapersOnLine, 49(12), 402-407.*

Pichard, R., Philippot, A., & Riera, B. (2017). Consistency Checking of Safety Constraints for Manufacturing Systems with Graph Analysis. *IFAC-PapersOnLine, 50(1), 1193-1198.*

Pichard, R., Philippot, A., Saddem, R., & Riera, B. (2018). Safety of Manufacturing Systems Controllers by Logical Constraints with Safety Filter. *IEEE Transactions on Control Systems Technology.*

Riera B., Philippot A., Coupat R., Gellot F., Annebicque D. (2015). A non-intrusive method to make safe existing

PLC Program, *9th IFAC Symposium on Fault Detection, Supervision and Safety for Technical Processes (SAFEPROCESS'15)*, Paris, France, September 2015.

Riera, B. Vigario, B (2013). Virtual Systems to Train and Assist Control Applications in Future Factories. *IFAC Analysis, Design, and Evaluation of Human-Machine Systems, Volume # 12 | Part# 1, pp 76-81, Elsevier*, Las Vegas, 2013.

Tsang E. P. K. (1993). *Foundations of Constraints Satisfaction*, Academic Press Limited, UK, 1993.

Vizel, Y., Weissenbacher, G., Malik, S. (2015). *Boolean Satisfiability Solvers and Their Applications in Model Checking*. Proceedings of the IEEE 103 (11). doi:10.1109/JPROC.2015.2455034.

Zaytoon, J. and Riera, B. *Synthesis and implementation of logic controllers – A review.* Annual Reviews in Control, Volume 43, 2017, Pages 152-168, 2017.