Control of an automatic parking gate

Jean-Marc ROUSSEL
jean-marc.roussel@lurpa.ens-cachan.fr

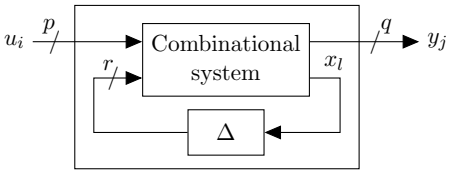
LURPA,ENS Cachan
61, avenue du Prsident Wilson
94235 CACHAN Cedex

April 20, 2012

1 Introduction

This document presents a case study made according to the algebraic synthesis method developed in LURPA. We propose to obtain the control law to implement into a Programmable Logic Controller (PLC) from its specifications given in natural language, by solving a Boolean equations system of switching functions.

We suppose that the expected control law can be expressed with recurrent Boolean equations as presented Figure 1. This generic model has p Boolean inputs (u_i), q Boolean outputs (y_j) and r Boolean state variables (x_l). These inputs and outputs correspond to the inputs and outputs of the controller for which the control laws must be designed. The state variables, used to express the sequential behavior, will be represented with internal variables of the controller.



$$\begin{cases} y_j[k] = F_j(u_1[k], \cdots, u_p[k], x_1[k-1], \cdots, x_r[k-1]) \\ x_l[k] = F_{q+l}(u_1[k], \cdots, u_p[k], x_1[k-1], \cdots, x_r[k-1]) \end{cases}$$

Figure 1: Generic model of sequential systems expressed with recurrent Boolean equations

The behavior of this model can be fully defined according to the definition of $(q+r)$ switching functions of $(p+r)$ variables. Even if this representation is very compact (the r Boolean state variables allow the representation of 2^r different states), the construction by hands of these switching functions has always been a very tedious and error-prone task [Huf54]: the model presented Figure. 1, admits 2^p inputs combinations, can send 2^q outputs combinations and can express $(2^{2^{(p+r)}})^{(q+r)}$ sequential behaviors.

Nevertheless, thanks to recent mathematical results obtained for Boolean algebras [Rud01], [Bro03], the automatic algebraic synthesis of switching functions is now possible. We propose to obtain the control law to implement into a PLC by solving a Boolean equations system of switching functions. Details of the proposed method can be found in [HRL08a] [HRL08b] [Hie09].

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[1] performs all the computations required for inconsistencies detection and control laws generation. This enables the designer to focus only on application-related issues. For ergonomic reasons, complementary works were also developed in order be able to represent the synthesized control law with a state model.

[1] Case studies are available: http://www.lurpa.ens-cachan.fr/isa/asc/case_studies.html

1.1 Notations used

To avoid confusion between Boolean variables and Boolean functions of Boolean variables, each Boolean variable b_i is denoted as ${}_b b_i$. The set of the two Boolean values ${}_b 0$ and ${}_b 1$ is denoted as: $B = \{{}_b 0, {}_b 1\}$. The classical two-element Boolean Algebra is $(\{{}_b 0, {}_b 1\}, \vee, \wedge, \neg, {}_b 0, {}_b 1)$.

Let $F_n(B)$ be the set of the 2^{2^n} n-variable switching functions. The Boolean Algebra of n-variable switching functions is $(F_n(B), +, \cdot, \bar{}, 0, 1)$:

- 0 and 1 are the 2 constant functions:

$$\begin{array}{rcl} 0: & B^n \to B & \qquad 1: \quad B^n \to B \\ & ({}_b b_1, \ldots, {}_b b_n) \mapsto {}_b 0 & \qquad ({}_b b_1, \ldots, {}_b b_n) \mapsto {}_b 1 \end{array}$$

- $+$, $\cdot$, $\bar{}$ are three closed operations defined as follows:

$$\begin{array}{rcl} \text{Op. } +: & F_n(B)^2 \to F_n(B) & \qquad \text{Op. } \cdot: \quad F_n(B)^2 \to F_n(B) \qquad \text{Op. } \bar{}: \quad F_n(B) \to F_n(B) \\ & (f, g) \mapsto f + g & \qquad (f, g) \mapsto f \cdot g \qquad\qquad f \mapsto \overline{f} \end{array}$$

$$\begin{array}{l} \text{where } \forall ({}_b b_1, \ldots, {}_b b_n) \in B^n, \\ (f+g)({}_b b_1, \ldots, {}_b b_n) = f({}_b b_1, \ldots, {}_b b_n) \vee g({}_b b_1, \ldots, {}_b b_n) \\ (f \cdot g)({}_b b_1, \ldots, {}_b b_n) = f({}_b b_1, \ldots, {}_b b_n) \wedge g({}_b b_1, \ldots, {}_b b_n) \\ \overline{f}({}_b b_1, \ldots, {}_b b_n) = \neg f({}_b b_1, \ldots, {}_b b_n) \end{array}$$

$F_n(B)$ can be equipped with a partial order relation, called *Inclusion-Relation* defined as follows:

$$x \leq y \quad \Leftrightarrow \quad x \cdot y = x$$

2 Control system specifications

The studied system is the controller of automatic parking gate. Users could open the gate by activating the remote control. The gate moves thanks to an electrical asynchronous motor controlled by two distinct contactors.

2.1 Inputs and outputs of the controller

The Boolean inputs and outputs of this controller are given in Fig. 2. Each movement of the gate is controlled thanks to a distinct contactor (outputs: 'open' and 'close'). The controller is informed of the position of the gate thanks to two on-off switches (inputs: 'go' and 'gc'). A sensor permits to detect the presence of a car in front of the gate (input: 'car'). Users could open the gate by activating the remote control (input: 'rc').

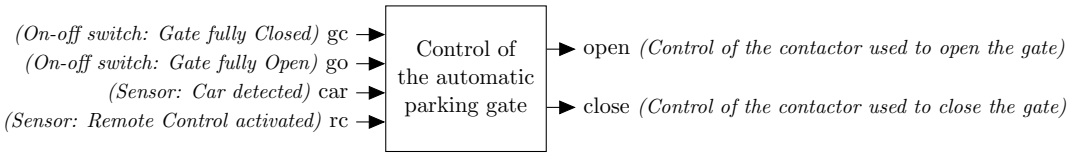


Figure 2: Inputs and outputs of the controller to design

2.2 Expected behavior

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

- **F1:** When the remote control is activated, the gate opens.
- **F2:** If the gate is not fully closed, the detection of a car brings about the opening of the gate.
- **F3:** The opening of the gate must be full: to stop the opening of the gate, it is necessary that the gate is fully open.
- **F4:** When the remote control is activated, the gate can not be control to close.
- **S1:** To close the gate, it is necessary that no car is detected.
- **T1:** The gate must never be simultaneously controlled to open and to close.
- **T2:** When the gate is fully open, the gate must never be controlled to open.
- **T3:** When the gate is fully closed, the gate must never be controlled to close.

Assertions F1 to F4 are functional requirements. Assertion S1 was introduced to protect users. Assertions T1 to T3 are technical requirements imposed by the use of a electrical motor controlled by two distinct contactors.

2.3 Control laws to design

Our approach does not permit to identify automatically which state variables must be used. They are given by the designer according to its interpretation of the specification.

For the automatic gate, we propose to use two state variables: one for each output. According to this choice, we have only two switching functions to synthesize. However, each function is a 6-variable switching functions as the control laws has 4 inputs and 2 state variables. The generic form of the control law we want to design is:

$$\begin{cases} \text{open}[k] = \text{Open } (\text{gc}[k], \text{go}[k], \text{car}[k], \text{rc}[k], \text{open}[k-1], \text{close}[k-1]) \\ \text{close}[k] = \text{Close } (\text{gc}[k], \text{go}[k], \text{car}[k], \text{rc}[k], \text{open}[k-1], \text{close}[k-1]) \\ \text{open}[0] = {}_b0 \\ \text{close}[0] = {}_b0 \end{cases}$$

This simple model permits to express 2^{128} ($(2^{2^6})^2$) different control laws. We propose to find the control law which satisfies the expected behavior given Section 2.2 by solving a Boolean equations system of 6-variable switching functions.

3 Algebraic synthesis of the control laws

The first step of the proposed method consists to formalize the expected behavior with relations between formula of 6-variable switching functions. For this case study, we have 6 specific switching functions[2]:

- The 4 switching functions (GC, GO, Car and RC) which characterize the behavior of the inputs of the controller and are defined as follows:

$$\begin{aligned} \mathrm{GC}: \quad & B^6 \to B \\ & (\mathrm{gc}[k], \ldots, \mathrm{close}[k-1]) \mapsto \mathrm{gc}[k] \end{aligned}$$

- The 2 switching functions ($_p$Open and $_p$Close) which characterize the previous behavior of the state variables of the controller and are defined as follows:

$$\begin{aligned} {}_p\mathrm{Open}: \quad & B^6 \to B \\ & (\mathrm{gc}[k], \ldots, \mathrm{close}[k-1]) \mapsto \mathrm{open}[k-1] \end{aligned}$$

In our case, only 2 switching functions must be designed (Open and Close). They represent the unknowns or our problem.

Remark: As Open and $_p$Open represent the behavior of 'open' at respectively times $[k]$ and $[k-1]$, the starting of the opening of the gate corresponds to $(\mathrm{Open} \cdot \overline{{}_p\mathrm{Open}})$ and the stopping of the opening of the gate corresponds to $(\overline{\mathrm{Open}} \cdot {}_p\mathrm{Open})$.

3.1 Formalization of requirements

Assertions describing the expected behavior of control systems in natural language can be translated into formal statements thanks to the relations Equality and Inclusion.

- **F1:** When the remote control is activated, the gate opens.

$$\mathrm{RC} \leq \mathrm{Open}$$

- **F2:** While the gate is not fully closed, the detection of a car brings about the opening of the gate.

$$\overline{\mathrm{GC}} \cdot \mathrm{Car} \leq \mathrm{Open}$$

- **F3:** The opening of the gate must be full: to stop the opening of the gate, it is necessary that the gate is fully open.

$$(\overline{\mathrm{Open}} \cdot {}_p\mathrm{Open}) \leq \mathrm{GO}$$

- **F4:** When the remote control is activated, the gate can not be control to close.

$$\mathrm{RC} \leq \overline{\mathrm{Close}}$$

- **S1:** To close the gate, it is necessary that no car is detected.

$$\mathrm{Close} \leq \overline{\mathrm{Car}}$$

- **T1:** The gate must never be simultaneously controlled to open and to close.

$$\mathrm{Open} \cdot \mathrm{Close} = 0$$

[2]These functions are the 6 projection-functions of $F_6(B)$.

- **T2:** When the gate is fully open, the gate must never be controlled to open.

$$\mathrm{GO} \leq \overline{\mathrm{Open}}$$

- **T3:** When the gate is fully closed, the gate must never be controlled to close.

$$\mathrm{GC} \leq \overline{\mathrm{Close}}$$

3.2 Consistency checking

The result of the formalization of all the requirements is composed of a set of relations:

$$\begin{cases} \text{F1:} & \mathrm{RC} \leq \mathrm{Open} \\ \text{F2:} & \overline{\mathrm{GC}} \cdot \mathrm{Car} \leq \mathrm{Open} \\ \text{F3:} & (\overline{\mathrm{Open}} \cdot_p \mathrm{Open}) \leq \mathrm{GO} \\ \text{F4:} & \mathrm{RC} \leq \overline{\mathrm{Close}} \\ \text{S1:} & \mathrm{Close} \leq \overline{\mathrm{Car}} \\ \text{T1:} & \mathrm{Open} \cdot \mathrm{Close} = 0 \\ \text{T2:} & \mathrm{GO} \leq \overline{\mathrm{Open}} \\ \text{T3:} & \mathrm{GC} \leq \overline{\mathrm{Close}} \end{cases}$$

For this problem, the method we propose permits to prove that the given requirements are inconsistent. the result given by our software tool was the following inconsistency condition:

$$\mathcal{I} = (\mathrm{GO} \cdot \mathrm{RC}) + (\overline{\mathrm{GC}} \cdot \mathrm{GO} \cdot \mathrm{Car})$$

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

- $(\mathrm{GO} \cdot \mathrm{RC})$: What appends if the remote control is activated when the gate is fully open? We consider that the requirement T2 has priority on requirement F1 (for security reasons) and we have added the priority rule: $\mathrm{T2} \gg \mathrm{F1}$.
- $(\overline{\mathrm{GC}} \cdot \mathrm{GO} \cdot \mathrm{Car})$: What appends if a car is detected when the gate is fully open? We consider that the requirement T2 has priority on requirement F2 (for security reasons) and we have added the priority rule: $\mathrm{T2} \gg \mathrm{F2}$.

The result of the formalization of all the requirements is composed of a set of relations and priority rules:

$$\left[\begin{array}{l} \textbf{Requirements:} \\ \quad \begin{cases} \text{F1:} & \mathrm{RC} \leq \mathrm{Open} \\ \text{F2:} & \overline{\mathrm{GC}} \cdot \mathrm{Car} \leq \mathrm{Open} \\ \text{F3:} & (\overline{\mathrm{Open}} \cdot_p \mathrm{Open}) \leq \mathrm{GO} \\ \text{F4:} & \mathrm{RC} \leq \overline{\mathrm{Close}} \\ \text{S1:} & \mathrm{Close} \leq \overline{\mathrm{Car}} \\ \text{T1:} & \mathrm{Open} \cdot \mathrm{Close} = 0 \\ \text{T2:} & \mathrm{GO} \leq \overline{\mathrm{Open}} \\ \text{T3:} & \mathrm{GC} \leq \overline{\mathrm{Close}} \end{cases} \\ \textbf{Priority rules:} \\ \quad \begin{cases} \mathrm{T2} \gg \mathrm{F1} \\ \mathrm{T2} \gg \mathrm{F2} \end{cases} \end{array}\right.$$

3.3 Equation solving

Thanks to the last mathematical results, we are able to obtain automatically the different solutions of this problem. The parametric for of the solutions is:

$$\begin{cases} \text{Open} = \overline{\text{GO}} \cdot (\text{RC} +_p \text{Open} + \overline{\text{GC}} \cdot \text{Car}) + p_1 \cdot \overline{\text{GO}} & p_1 \in F_6(B) \\ \text{Close} = p_2 \cdot \overline{\text{GC}} \cdot \overline{\text{Car}} \cdot \overline{\text{RC}} \cdot (\text{GO} + \overline{{}_p\text{Open}} \cdot \overline{p_1}) & p_2 \in F_6(B) \end{cases}$$

A particular solution has to be chosen among the set of solutions. For that, a specific value of each parameter of the general solution has to be fixed. In some cases, optimal solutions, according to given criteria, can be automatically found ([Ler11]). For the automatic gate, we can consider:

- The opening of the gate is made only if it is necessary.
- The closing of the gate is made when it is possible.

We have chosen the solution which minimizes the opening of the gate (the parameter p_1 is fixed to 0) and maximizes the closing of the gate (the parameter p_2 is fixed to 1). The solution of this problem is:

$$\begin{cases} \text{Open} = \overline{\text{GO}} \cdot (\text{RC} +_p \text{Open} + \overline{\text{GC}} \cdot \text{Car}) \\ \text{Close} = \overline{\text{GC}} \cdot \overline{\text{Car}} \cdot \overline{\text{RC}} \cdot (\text{GO} + \overline{{}_p\text{Open}}) \end{cases}$$

4 Obtained contol laws

4.1 Representation with recurrent Boolean equations

The control laws presented hereafter was obtained by translating the expression of the unknowns according to the projection-functions into relations between recurrent Boolean equations.

$$\begin{cases} \text{open}[k] = \neg\text{go}[k] \wedge (\text{rc}[k] \vee \text{open}[k-1] \vee \neg\text{gc}[k] \wedge \text{car}[k]) \\ \text{close}[k] = \neg\text{gc}[k] \wedge \neg\text{car}[k] \wedge \neg\text{rc}[k] \wedge (\text{go} \vee \neg\text{open}[k-1]) \\ \text{open}[0] =_b 0 \\ \text{close}[0] =_b 0 \end{cases}$$

This control law was implemented into a PLC with the Ladder Diagram language [IEC03]. The code is composed of only three rungs (Figure 3).

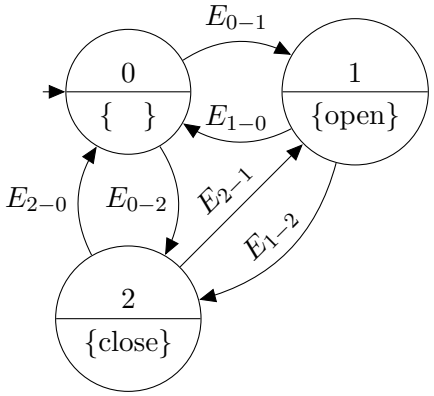
```
Rung 1: Command of Open                    Rung 3: Update previous value of Open
|  go      rc                   open  |    |  open                          pOp   |
+--|/|--+--| |-------+----------( )---+    +--| |--------------------------( )---+
|       |  gc   car  |                |    |                                      |
|       +--|/|--| |--+                |
|       |  pOp       |                |
|       +--| |-------+                |
|                                     |

Rung 2: Command of Close
|  gc   car  rc      go         close  |
+--|/|--|/|--|/|--+--| |--+------( )---+
|                 |  pOp  |            |
|                 +--|/|--+            |
```

Figure 3: Ladder Diagram of the code to implement into the PLC

4.2 Representation with a state model

If recurrent Boolean equations are well-adapted for an implementation, the representation of the control law with a state model simplifies the work of the designer. For this control law, the equivalent state model (automatically built thanks to [Gui11]) is composed of three states only (Figure 4). Each state is defined according to the set of emitted outputs. The six transition conditions are a Boolean expression of the inputs. By construction, this state model satisfies all the requirements given Section 2.2.



$$\begin{cases} E_{0-1} = \neg \text{go} \wedge (\text{rc} \vee \neg \text{gc} \wedge \text{car}) \\ E_{0-2} = \neg \text{gc} \wedge \neg \text{car} \wedge \neg \text{rc} \\ E_{1-0} = \text{go} \wedge (\text{gc} \vee \text{rc} \vee \text{car}) \\ E_{1-2} = \text{go} \wedge \neg \text{gc} \wedge \neg \text{rc} \wedge \neg \text{car} \\ E_{2-0} = \text{go} \wedge (\text{rc} \vee \text{car}) \vee \text{gc} \wedge \neg \text{rc} \\ E_{2-1} = \neg \text{go} \wedge (\text{rc} \vee \neg \text{gc} \wedge \text{car}) \end{cases}$$

Figure 4: State model of the obtained control law

5 Conclusions

For this case study, the method we propose has allowed to find the control law for the pumping system. The use of priority rules and optimization criteria has simplified greatly the formalization of the requirements.

References

[Bro03] Frank Markham Brown. *Boolean Reasoning: The Logic of Boolean Equations*. Dover Publications, 2003.

[Gui11] Anaïs Guignard. Symbolic generation of the automaton representing an algebraic description of a logic system. Master's thesis, ENS Cachan, July 2011.

[Hie09] Yann Hietter. *Synthèse algébrique de lois de commande pour les systèmes à évènements discrets logiques*. PhD thesis, ENS Cachan, May 2009.

[HRL08a] Yann Hietter, Jean-Marc Roussel, and Jean-Jacques Lesage. Algebraic synthesis of dependable logic controllers. In *Proceedings of 17th IFAC World Congress, 2008*, pages pp. 4132–4137, Seoul, South Korea, July 2008.

[HRL08b] Yann Hietter, Jean-Marc Roussel, and Jean-Jacques Lesage. Algebraic synthesis of transition conditions of a state model. In *Proceedings of 9th International Workshop On Discrete Event Systems (WODES'08)*, pages 187–192, Göteborg, Sweden, May 2008.

[Huf54] D. A. Huffman. The synthesis of sequential switching circuits. *J. of the Franklin Institute*, 257(3-4):161–190 and 275–303, 1954.

[IEC03] IEC 61131-3. *IEC 61131-3 Standard: Programmable controllers - Part 3: Programming languages*. International Electrotechnical Commission, 2 edition, 2003.

[Ler11] Hélène Leroux. Algebraic synthesis of logical controllers with optimization criteria. Master's thesis, ENS Cachan, July 2011.

[Rud01] Sergiu Rudeanu. *Lattice Functions and Equations (Discrete Mathematics and Theoretical Computer Science)*. Springer, 2001.