Incremental Design of a Po wer Transformer Station Controller using a Controller Synthesis Methodology[⋆]

Hervé Marchand[1] and Mazen Samaan[2]

[1] IRISA / INRIA - Rennes,
F-35042 RENNES, France
e-mail: hmarchan@irisa.fr
[2] EDF/DER, EP, dept. CCC,
6 quai Watier, 78401 CHATOU, France
e-mail: Mazen.Samaan@der.edf.fr

Abstract. In this paper, we describe the incremental specification of a pow er transformer station controller using a *controller synthesis methodology*. We specify the main requirements as simple properties, named *control objectives*, that the controlled plant has to satisfy. Then, using algebraic tec hniques, the controller is automatically derived from these set of con trol objectiv es. In our case, the plant is specified at a high level, using the data-flow synchronous SIGNAL language and then by its logical abstraction, named polynomial dynamical system. The control objectives are specified as *invariance, reachability, attractivity* properties, as well as *partial order relations* to be checked by the plant. The control objectives equations are then synthesized using algebraic transformations.

Key-words: Discrete Even t Systems, Polynomial Dynamical System, Supervisory Control Problem, SIGNAL, Pow er Plant.

1 Introduction & Motivations

The SIGNAL language [8] is developed for precise specification of real-time reactive systems [2]. In such systems, requirements are usually chec ked *a posteriori* using property verification and/or simulation techniques. Control theory of Discrete Even t Systems (DES) allows to use constructive methods, that ensure, *a priori*, required properties of the system behavior. The validation phase is then reduced to properties that are not guaranteed by the programming process.

There exist different theories for control of Discrete Even t Systems since the 80's [14, 1, 5, 13]. Here, we choose to specify the plant in SIGNAL and the control synthesis as well as v erification are performed on a logical abstraction of this program, called a polynomial dynamical system (PDS) ov er $\mathbb{Z}/_{3\mathbb{Z}}$. The control

⋆ This work was partially supported by Électricité de France (EDF) under contract n umber M64/7C8321/E5/11 and by the Esprit SYRF project 22703.

of the plant is performed by restricting the controllable input values with respect to the control objectives (logical or optimal). These restrictions are obtained by incorporating new algebraic equations into the initial system. The theory of PDS uses classical tools in algebraic geometry, such as ideals, varieties and morphisms. This theory sets the basis for the verification and the formal calculus tool, SIGALI built around the SIGNAL environment. SIGALI manipulates the system of equations instead of the sets of solutions, avoiding the enumeration of the state space. This abstract level avoids a particular choice of set implementations, such as BDDs, even if all operations are actually based on this representation for sets.

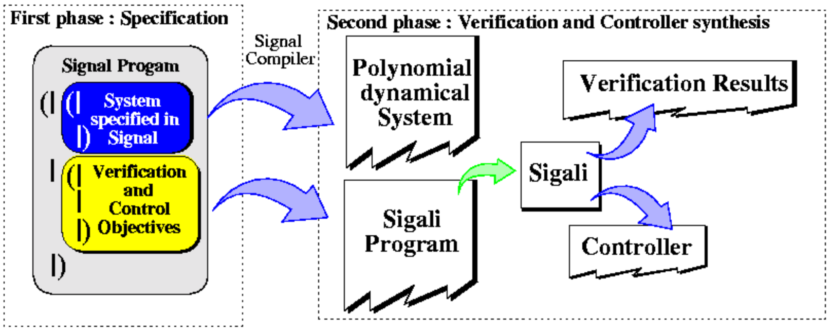

First phase : Specification
Signal Progam
System specified in Signal
Verification and Control Objectives
Signal Compiler
Second phase : Verification and Controller synthesis
Polynomial dynamical System
Sigali Program
Sigali
Verification Results
Controller

Fig. 1. Description of the tool

The methodology is the following (see Figure 1). The user first specifies in SIGNAL both the physical model and the control/verification objectives to be ensured/checked. The SIGNAL compiler translates the SIGNAL program into a PDS, and the control/verification objectives in terms of polynomial relations/operations. The controller is then synthesized using SIGALI. The result is a controller coded by a polynomial and then by a Binary Decision Diagram.

To illustrate our approach, we consider in this paper the application to the specification of the automatic control system of a power transformer station. It concerns the response to electric faults on the lines traversing it. It involves complex interactions between communicating automata, interruption and preemption behaviors, timers and timeouts, reactivity to external events, among others. The functionality of the controller is to handle the power interruption, the redirection of supply sources, and the re-establishment of the power following an interruption. The objective is twofold: the safety of material and uninterrupted best service. The safety of material can be achieved by (automatic) triggering circuit-breakers when an electric fault occurs on lines, whereas the best quality service can be achieved by minimizing the number of costumers concerned by a power cut, and re-establishment of the current as quickly as possible for the customers hit by the fault (i.e, minimizing the failure in the distribution of power in terms of duration and size of the interrupted sub-network).

2 Overview of the power transformer station

In this section, we make a brief description of the power transformer station network as well as the various requirements the controller has to handle.

2.1 The power transformer station description

Électricité de France has hundreds of high voltage networks linked to production and medium voltage networks connected to distribution. Each station consists of one or more power transformer stations to which circuit-breakers are connected. The purpose of an electric power transformer station is to lower the voltage so that it can be distributed in urban centers to end-users. The kind of transformer (see Figure 2) we consider, receives high voltage lines, and feeds several medium voltage lines to distribute power to end-users.

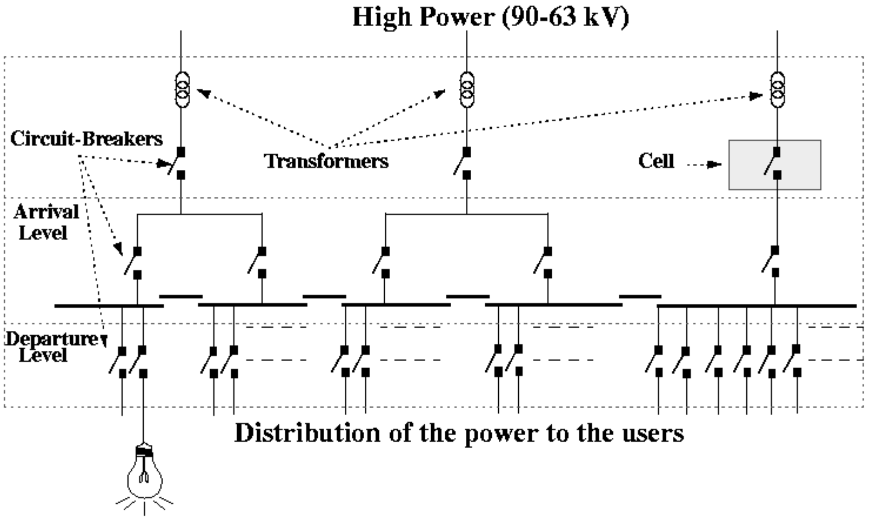
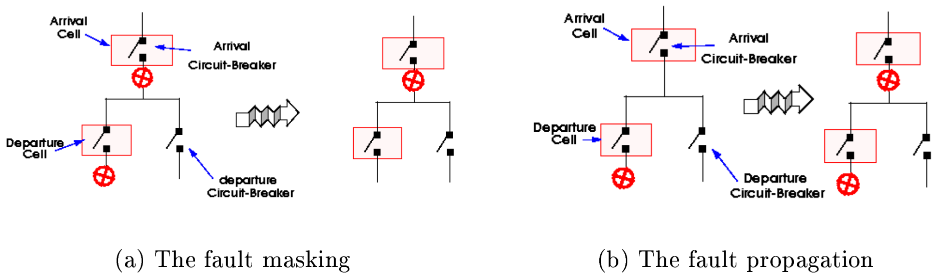


Fig. 2. The power transformer station topology.

For each high voltage line, a transformer lowers the voltage. During operation of this system, several faults can occur (three types of electric faults are considered: phase PH, homopolar H, or wattmetric W), due to causes internal or external to the station. To protect the device and the environment, several circuit breakers are placed in a network of *cells* in different parts of the station (on the arrival lines, link lines, and departure lines). These circuit breakers are informed about the possible presence of faults by sensors.

Power and Fault Propagation: We discuss here some physical properties of the power network located inside the power transformer station controller. It is obvious that the power can be seen by the different cells if and only if all the upstream circuit-breakers are closed. Consequently, if the link circuit-breaker is opened, the power is cut and no fault can be seen by the different cells of the power transformer station. The visibility of the fault by the sensors of the cells is less obvious. In fact, we have to consider two major properties:

- On one hand, if a physical fault, considered as an input of our system, is seen by the sensors of a cell, then all the downstream sensors are not able to see some physical faults. In fact, the appearance of a fault at a certain level (the departure level in Figure 3(a) for example) increases the voltage on the downstream lines and masks all the other possible faults.



(a) The fault masking (b) The fault propagation

Fig. 3. The Fault properties

- On the other hand, if the sensors of a cell at a given level (for example the sensors of one of the departure cells as illustrated in Figure 3(b)) are informed about the presence of a fault, then all the upstream sensors (here the sensors of the arrival cell) detect the same fault. Consequently, it is the arrival cell that handle the fault.

2.2 The controller

The controller can be divided into two parts. The first part concerns the local controllers (*i.e.*, the cells). We chose to specify each local controller in SIGNAL, because they merge logical and numerical aspects. We give here only a brief description of the behavior of the different cells (more details can be found in [12, 7]). The other part concerns more general requirements to be checked by the global controller of the power transformer station. That specification will be described in the following.

The Cells: Each circuit breaker controller (or cell) defines a behavior beginning with the confirmation and identification of the type of the fault. In fact, a variety of faults are transient, i.e., they occur only for a very short time. Since their duration is so short that they do not cause any danger, the operation of the circuit-breaker is inhibited. The purpose of this confirmation phase is let the transient faults disappear spontaneously. If the fault is confirmed, the handling consists in opening the circuit-breaker during a given delay for a certain number of periods and then closing it again. The circuit-breaker is opened in consecutive cycles with an increased duration. At the end of each cycle, if the fault is still present, the circuit-breaker is reopened. Finally, in case the fault is still present at the end of the last cycle, the circuit-breaker is opened definitively, and control is given to the remote operator.

The specification of a large part of these local controllers has been performed using the SIGNAL synchronous language [12] and verified using our formal calculus system, named SIGALI [7].

Some global requirements for the controller: Even if is quite easy to specify the local controllers in SIGNAL, some other requirements are too informal, or their behaviors are too complex to be expressed directly as programs.

1. One of the most significant problems concerns the appearance of two faults (the kind of faults is not important here) at two different departure cells, at the same time. Double faults are very dangerous, because they imply high defective currents. At the place of the fault, this results in a dangerous path voltage that can electrocute people or cause heavy material damages. The detection of these double faults must be performed as fast as possible as well as the handling of one of the faults.
2. Another important aspect is to know which of the circuit breakers must be opened. If the fault appears on the departure line, it is possible to open the circuit breaker at departure level, at link level, or at arrival level. Obviously, it is in the interest of users that the circuit be broken at the departure level, and not at a higher level, so that the fewest users are deprived of power.
3. We also have to take into account the importance of the departure circuit-breaker. Assume that some departure line, involved in a double faults problem, supplies a hospital. Then, if the double faults occur, the controller should not open this circuit-breaker, since electricity must always delivered to a hospital.

The transformer station network as well as the cells are specified in SIGNAL. In order to take into account the requirements (1), (2) and (3), with the purpose of obtaining an optimal controller, we rely on automatic controller synthesis that is performed on the logical abstraction of the global system (network + cells).

3 The SIGNAL equational data flow real-time language

SIGNAL [8] is built around a minimal kernel of operators. It manipulates *signals* `X`, which denote unbounded series of typed values $(\mathtt{x}_t)_{t \in T}$, indexed by time t in a time domain T. An associated clock determines the set of instants at which values are present. A particular type of signals called `event` is characterized only by its presence, and has always the value *true* (hence, its negation by `not` is always $false$). The clock of a signal `X` is obtained by applying the operator `event X`. The constructs of the language can be used in an equational style to specify the relations between signals i.e. , between their values and between their clocks. Systems of equations on signals are built using a composition construct, thus defining *processes.* Data flow applications are activities executed over a set of instants in time. At each instant, input data is acquired from the execution environment; output values are produced according to the system of equations considered as a network of operations.

3.1 The SIGNAL language.

The kernel of the SIGNAL language is based on four operations, defining primitive processes or equations, and a composition operation to build more elaborate processes in the form of systems of equations.

Functions are instantaneous transformations on the data. The definition of a signal Y_t by the function f: $\forall t,\ Y_t = f(X_{1_t}, X_{2_t}, \ldots, X_{n_t})$ is written in SIGNAL: `Y :=` f`{ X1, X2, ... , Xn}`. `Y`, `X1`, ... , `Xn` are required to have the same clock.

Selection of a signal `X` according to a boolean condition `C` is: `Y := X when C`. If `C` is present and *true*, then `Y` has the presence and value of `X`. The clock of `Y` is the *intersection* of that of `X` and that of `C` at the value *true*.

Deterministic merge noted: `Z := X default Y` has the value of `X` when it is present, or otherwise that of `Y` if it is present and `X` is not. Its clock is the *union* of that of `X` and that of `Y`.

Delay gives access to past values of a signal. E.g., the equation $ZX_t = X_{t-1}$, with initial value V_0 defines a *dynamic process.* It is encoded by: `ZX := X$1` with initialization `ZX init V0`. `X` and `ZX` have equal clocks.

Composition of processes is noted "`|`" (for processes P_1 and P_2, with parenthesizing: `(|` P_1 `|` P_2 `|)`). It consists in the composition of the systems of equations; it is associative and commutative. It can be interpreted as parallelism between processes.

The following table illustrates each of the primitives with a trace:

n	3	2	1	0	3	2	...
zn := n$ 1 init 0	0	3	2	1	0	3	....
p := zn-1	-1	2	1	0	-1	2	...
x := true when (zn=0)	t				t		
y := true when (n=0) default (not x)	f			t	f		

Derived features: Derived processes have been defined on the base of the primitive operators, providing programming comfort. E.g., the instruction `X ^= Y` specifies that signals `X` and `Y` are synchronous (i.e., have equal clocks); `when B` gives the clock of *true*-valued occurrences of `B`.

For a more detailed description of the language, its semantic, and applications, the reader is referred to [8]. The complete programming environment also features a block-diagram oriented graphical user interface and a proof system for dynamic properties of SIGNAL programs, called SIGALI (see Section 4).

3.2 Specification in Signal of the power transformer station

The transformer station network we are considering contains four departure, two arrival and one link circuit-breakers as well as the cells that control each circuit-breaker [7]. The process `Physical_Model` in Figure 4 describes the power and fault propagation according to the state of the different circuit-breakers. It is composed of nine subprocesses. The process `Power_Propagation` describes the propagation of power according to the state of the circuit-breakers (Open/Closed). The process `Fault_Visibility` describes the fault propagation and visibility according to the other faults that are potentially present. The remaining seven processes encode the different circuit-breakers.

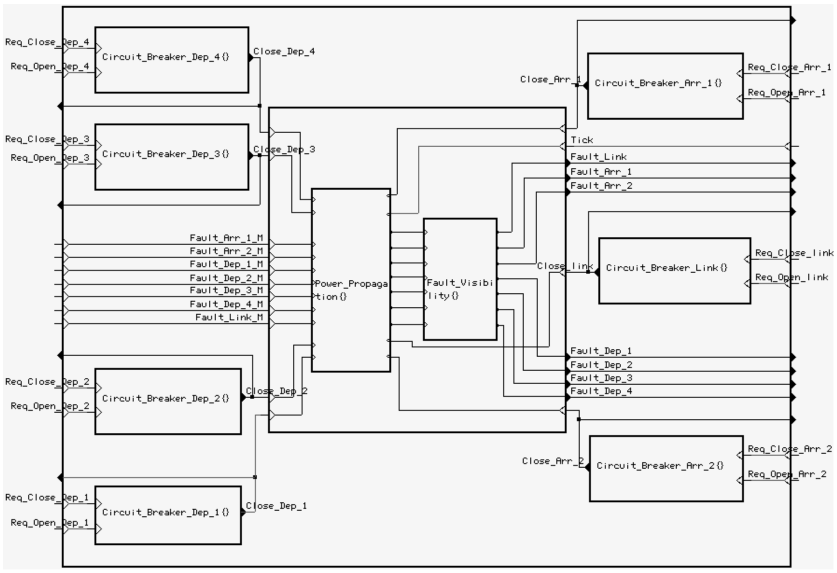


Fig. 4. The main process in Signal

The inputs of this main process are booleans that encode the physical faults: `Fault_Link_M`, `Fault_Arr_i_M` (i=1,2), `Fault_Dep_j_M` (j =1,..,4). They encode faults that are really present on the different lines. The event inputs `req_close_...` and `req_open_...` indicate opening and closing requests of the various circuit-breakers. The outputs of the main process are the booleans `Fault_Link`, `Fault_Arr_i`, `Fault_Dep_j`, representing the signals that are sent to the different cells. They indicate whether a cell is faulty or not. These outputs represents the knowledge that the sensors of the different cells have.

We will now see how the subprocesses are specified in Signal.

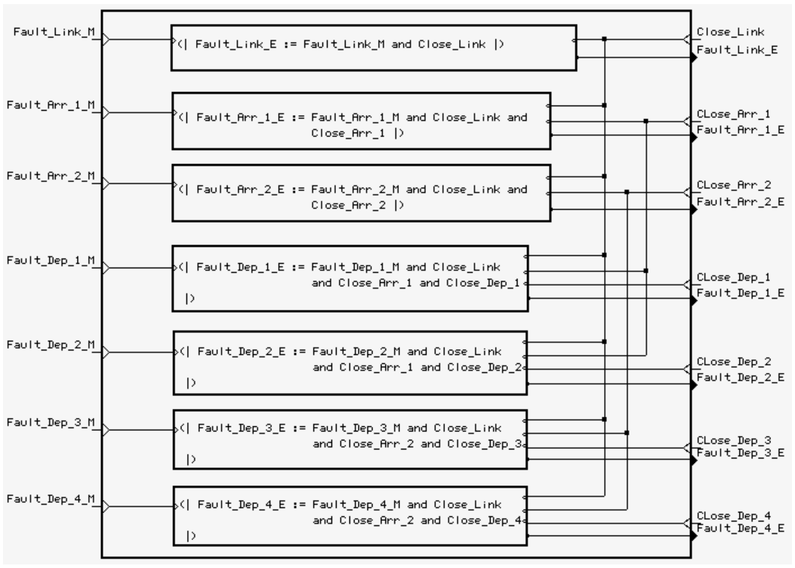
The circuit-breaker: A circuit-breaker is specified in Signal as follows: The process `Circuit-Breaker` takes two sensors inputs: `Req_Open` and `Req_Close`. They represent opening and closing requests. The output `Close` represents the status of the circuit-breaker.

```
(| Close := (Req_Close default (false when Req_Open) default Z_Close
 | Z_Close := Close $1 init true
 | Close ^= Tick
 | (Req_Close when Req_Open) ^= when (not Req_Open) |)
```

Fig. 5. The Circuit-breaker in SIGNAL

The boolean `Close` becomes *true* when the process receives the event `req_close`, and *false* when it receives the event `Req_open`, otherwise it is equal to its last value (i.e. `Close` is *true* when the circuit-breaker is closed and *false* otherwise). The constraint `Req_Close when Req_Open ^= when not Req_Close` says that the two events `Req_Close` and `Req_Open` are exclusive.

Power Propagation: It is a filter process using the state of the circuit-breakers. Power propagation also induces a visibility of possible faults. If a circuit-breaker is open then no fault can be detected by the sensors of downstream cells.

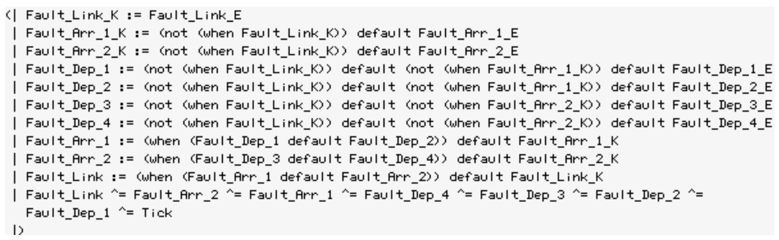


```
(| Fault_Link_E := Fault_Link_M and Close_Link |)
(| Fault_Arr_1_E := Fault_Arr_1_M and Close_Link and
                    Close_Arr_1 |)
(| Fault_Arr_2_E := Fault_Arr_2_M and Close_Link and
                    Close_Arr_2 |)
(| Fault_Dep_1_E := Fault_Dep_1_M and Close_Link
                    and Close_Arr_1 and Close_Dep_1
 |)
(| Fault_Dep_2_E := Fault_Dep_2_M and Close_Link
                    and Close_Arr_1 and Close_Dep_2
 |)
(| Fault_Dep_3_E := Fault_Dep_3_M and Close_Link
                    and Close_Arr_2 and Close_Dep_3
 |)
(| Fault_Dep_4_E := Fault_Dep_4_M and Close_Link
                    and Close_Arr_2 and Close_Dep_4
 |)
```

Fig. 6. specification in SIGNAL of the power propagation

This is specified in the process `Power_Propagation` shown in Figure 6. The inputs are booleans that code the physical faults and the status of the circuit-breakers. For example, a fault could be detected by the sensor of the departure cell 1 (i.e. `Fault_Dep_1_E` is *true*) if there exists a physical fault (`Fault_Dep_1_M`=*true*) and if the upstream circuit-breakers are closed (ie, `Close_Link`=*true* and `Close_Arr_1`=*true* and `Close_Dep_1`=*true*).

Fault visibility and propagation: The `Fault_Visibility` process in Figure 7, specifies fault visibility and propagation. As we explained in Section 2.1, a fault could be seen by the sensors of a cell only if no upstream fault is present.



```
(| Fault_Link_K := Fault_Link_E
 | Fault_Arr_1_K := (not (when Fault_Link_K)) default Fault_Arr_1_E
 | Fault_Arr_2_K := (not (when Fault_Link_K)) default Fault_Arr_2_E
 | Fault_Dep_1 := (not (when Fault_Link_K)) default (not (when Fault_Arr_1_K)) default Fault_Dep_1_E
 | Fault_Dep_2 := (not (when Fault_Link_K)) default (not (when Fault_Arr_1_K)) default Fault_Dep_2_E
 | Fault_Dep_3 := (not (when Fault_Link_K)) default (not (when Fault_Arr_2_K)) default Fault_Dep_3_E
 | Fault_Dep_4 := (not (when Fault_Link_K)) default (not (when Fault_Arr_2_K)) default Fault_Dep_4_E
 | Fault_Arr_1 := (when (Fault_Dep_1 default Fault_Dep_2)) default Fault_Arr_1_K
 | Fault_Arr_2 := (when (Fault_Dep_3 default Fault_Dep_4)) default Fault_Arr_2_K
 | Fault_Link := (when (Fault_Arr_1 default Fault_Arr_2)) default Fault_Link_K
 | Fault_Link ^= Fault_Arr_2 ^= Fault_Arr_1 ^= Fault_Dep_4 ^= Fault_Dep_3 ^= Fault_Dep_2 ^=
   Fault_Dep_1 ^= Tick
 |)
```

Fig. 7. Specification in SIGNAL of the fault propagation and visibility

For example, a fault cannot be detected by the sensor of the departure cell 1 (i.e. `Fault_Dep_1` is *false*), even if a physical fault exists at this level (`Fault_Dep_1_E`=*true*[1]), when another physical fault exists at the link level (`Fault_Link_1_K`=*true*) or at the arrival level 1 (`Fault_Arr_1_K`=*true*). It is thus, *true* just when the departure cell 1 detects a physical fault (`Fault_Dep_1_E`) and no upstream fault exists. *A contrario*, if a fault is picked up by a cell, then it is also picked up by the upstream cells. This is for example the meaning of `Fault_Link := (when (Fault_Arr_1 default Fault_Arr_2)) default Fault_link_K`.

4 Verification of SIGNAL programs

The SIGNAL environment contains a verification and controller synthesis toolbox, SIGALI. This tool allows us to prove the correctness of the dynamical behavior of the system. The equational nature of the SIGNAL language leads to the use of polynomial dynamical equation systems (PDS) over $\mathbb{Z}/_{3\mathbb{Z}}$, *i.e.* integers modulo 3: {-1,0,1}, as a formal model of program behavior. The theory of PDS uses classical concepts of algebraic geometry, such as ideals, varieties and comorphisms [6]. The techniques consist in manipulating the system of equations instead of the sets of solutions, which avoids enumerating state spaces.

To model its behavior, a SIGNAL process is translated into a system of polynomial equations over $\mathbb{Z}/_{3\mathbb{Z}}$ [7]. The three possible states of a boolean signal `X` (*i.e.* , *present* and *true*, *present* and *false*, or *absent*) are coded in a *signal variable* x by (*present* and *true* $\rightarrow 1$, *present* and *false* $\rightarrow -1$, and *absent* $\rightarrow 0$). For the non-boolean signals, we only code the fact that the signal is *present* or *absent*: (*present* $\rightarrow 1$ and *absent* $\rightarrow 0$).

Each of the primitive processes of SIGNAL are then encoded as polynomial equations. Let us just consider the example of the *selection* operator. `C := A when B` means "*if* $b = 1$ *then* $c = a$ *else* $c = 0$". It can be rewritten as a polynomial equation: $c = a(-b - b^2)$. Indeed, the solutions of this equation are the set of possible behaviors of the primitive process `when`. For example, if the signal B is *true* (*i.e.* , b=1), then $(-b - b^2) = (-1 - 1) = 1$ in $\mathbb{Z}/_{3\mathbb{Z}}$, which leads to $c = a$.

[1] Note that this fault has already be filtered. It can only be present if all the upstream circuit-breakers are closed

The delay `$`, which is dynamical, is different because it requires memorizing the past value of the signal into a *state variable* x. In order to encode `B := A$1 init B0`, we have to introduce the three following equations:

$$\begin{cases} x' = a + (1 - a^2)x & (1) \\ b = xa^2 & (2) \\ x_0 = b_0 & (3) \end{cases}$$

where x' is the value of the memory at the next instant. Equation (1) describes what will be the next value x' of the state variable. If a is *present*, x' is equal to a (because $(1-a^2) = 0$), otherwise x' is equal to the last value of a, memorized by x. Equation (2) gives to b the last value of a (*i.e.* the value of x) and constrains the clocks b and a to be equal. Equation (3) corresponds to the initial value of x, which is the initial value of b.

Table 1 shows how all the primitive operators are translated into polynomial equations. Remark that for the non boolean expressions, we just translate the synchronization between the signals.

Boolean expressions	
`B :=` `not A`	$b = -a$
`C :=` `A and B`	$c = ab(ab - a - b - 1)$ $a^2 = b^2 = c^2$
`C :=` `A or B`	$c = ab(1 - a - b - ab)$ $a^2 = b^2 = c^2$

`C :=` `A default B`
 `C :=` `A when B`
	$c = a + (1 - a^2)b$
	 $c = a(-b - b^2)$

`B :=` `A $1 (init` b_0`)`	$x' = a + (1 - a^2)x$ $b = a^2x$ $x_0 = \mathrm{b}_0$
non-boolean expressions	
`B :=` $f(A_1, \dots, A_n)$	$b^2 = a_1^2 = \dots = a_n^2$

`C :=` `A default B`
 `C :=` `A when B`
	$c^2 = a^2 + b^2 - a^2b^2$
	 $c^2 = a^2(-b - b^2)$

`B :=` `A $1 (init` b_0`)`	$b^2 = a^2$

Table 1. Translation of the primitive operators.

Any SIGNAL specification can be translated into a set of equations called polynomial dynamical system (PDS), that can be reorganized as follows:

$$S = \begin{cases} X' = P(X, Y) \\ Q(X, Y) = 0 \\ Q_0(X) = 0 \end{cases} \qquad (1)$$

where X, Y, X' are vectors of variables in $\mathbb{Z}/_{3\mathbb{Z}}$ and $dim(X) = dim(X')$. The components of the vectors X and X' represent the states of the system and are called *state variables*. They come from the translation of the delay operator. Y is a vector of variables in $\mathbb{Z}/_{3\mathbb{Z}}$, called *event variables*. The first equation is the *state transition equation*; the second equation is called the *constraint equation* and specifies which events may occur in a given state; the last equation gives the initial states. The behavior of such a PDS is the following: at each instant t, given a state x_t and an admissible y_t, such that $Q(x_t, y_t) = 0$, the system evolves into state $x_{t+1} = P(x_t, y_t)$.

Verification of a SIGNAL program: We now explain how verification of a SIGNAL program (in fact, the corresponding PDS) can be carried out. Using algebraic operations, it is possible to check properties such as *invariance*, *reachability* and *attractivity* [7]. Note that most of them will be used in the sequel as control objectives for controller synthesis purposes. We just give here the basic definitions of each of this properties.

Definition 1. *1. A set of states E is* **invariant** *for a dynamical system if for every x in E and every y admissible in x, $P(x,y)$ is still in E.*
2. A subset F of states is **reachable** *if and only if for every state $x \in F$ there exists a trajectory starting from the initial states that reaches x.*
3. A subset F of states is **attractive** *from a set of states E if and only if every state trajectory initialized in E reaches F.* •

For a more complete review of the theoretical foundation of this approach, the reader may refer to [6, 7].

Specification of a property: Using an extension of the SIGNAL language, named SIGNAL+, it is possible to express the properties to be checked, as well as the control objectives to be synthesized (see section 5.2), in the SIGNAL program. The syntax is

```
(| Sigali(Verif_Objective(PROP)) |)
```

The keyword `Sigali` means that the subexpression has to be evaluated by SIGALI. The function `Verif_Objective` (it could be `invariance`, `reachability`, `attractivity`, etc) means that SIGALI has to check the corresponding property according to the boolean `PROP`, which defines a set of states in the corresponding PDS. The complete SIGNAL program is obtained composing the process specifying the plant and the one specifying the verification objectives in parallel. Thus, the compiler produces a file which contains the polynomial dynamical system resulting from the abstraction of the complete SIGNAL program and the algebraic verification objectives. This file is then interpreted by SIGALI. Suppose that, for example, we want, in a SIGNAL program named "`system`", to check the attractivity of the set of states where the boolean `PROP` is *true*. The corresponding SIGNAL+ program is then:

```
(| system() (the physical model specified in Signal)
 | PROP: definition of the boolean PROP in Signal
 | Sigali(Attractivity(True(PROP))) |)
```

The corresponding SIGALI file, obtained after compilation of the SIGNAL program, is:

```
read(‘‘system.z3z’’);  => loading of the PDS
Set_States : True(PROP); => Compute the  states where PROP is true
Attractivity(S,Set_States);
 => Check for the attractivity of Set_States from the initial states
```

The file “system.z3z” contains in a coded form the polynomial dynamical system that represents the system. Set_States is a polynomial that is equal to 0 when the boolean PROP is *true*. The methods consist in verifying that the set of states where the polynomial Set_States takes the value 0 is attractive from the initial states (the answer is then *true* or *false*): Attractivity(S, Set_States). This file is then interpreted by SIGALI that checks the verification objective.

4.1 Verification of the power transformer network

In this section, we apply the tools to check various properties of our SIGNAL implementation of the transformer station. After the translation of the SIGNAL program, we obtain a PDS with 60 state variables and 35 event variables. Note that the compiler also checks the causal and temporal concurrency of our program and produces an executable code. We will now describe some of the different properties, which have been proved.

(1) *“There is no possibility to have a fault at the departure, arrival and link level when the link circuit-breaker is opened.”* In order to check this property, we add to the original specification the following code

```
(| Error:= ((Fault_Link or Fault_Arr_1 or Fault_Arr_1 or
      Fault_Dep_1 or Fault_Dep_2 or Fault_Dep_3 or Fault_Dep_4)
          when Open_Link)  default false
 | Error ^= Tick
 | Sigali(Reachable(True(Error))) |)
```

The Error signal is a boolean which takes the value *true* when the property is violated. In order to prove the property, we have to check that there does not exist any trajectory of the system which leads to the states where the Error signal is *true* (Reachable(True(Error))). The produced file is interpreted by SIGALI that checks whether this set of states is reachable or not. In this case, the result is *false*, which means that the boolean Error never takes the value *true*. The property is satisfied[2] . In the same way, we proved similar properties when one of the arrival or departure circuit-breakers is open.

(2) *“If there exists a physical fault at the link level and if this fault is picked up by its sensor then the arrival sensors can not detect a fault”*. We show here the property for the arrival cell 1. It can be expressed as an invariance of a set of states.

```
(| Error:= (Fault_Arr_1 when Fault_Link_E) default false
 | Error ^= Tick
 | Sigali(Invariance(False(Error))) |)
```

[2] Alternatively, this property could be also expressed as the invariance of the boolean False(Error), namely Sigali(Invariance(False(Error))).

We have proved similar properties for a departure fault as well as when a physical fault appears at the arrival level and at the departure level at the same time.

(3) We also proved using the same methods the following property: *"If a fault occurs at a departure level, then it is automatically seen by the upstream sensors when no other fault exists at a higher level."*

All the important properties of the transformer station network have been proved in this way. Note that the cell behaviors have also been proved (see [7] for more details).

5 The automatic controller synthesis methodology

5.1 Controllable polynomial dynamical system

Before speaking about control of polynomial dynamical systems, we first need to introduce a distinction between the events. From now on, we distinguish between the *uncontrollable* events which are sent by the system to the controller, and the *controllable* events which are sent by the controller to the system.

A polynomial dynamical system S is now written as:

$$S : \begin{cases} Q(X,Y,U) = & 0 \\ X' & = P(X,Y,U) \\ Q_0(X_0) & = \quad 0 \end{cases} \tag{2}$$

where the vector X represents the state variables; Y and U are respectively the set of *uncontrollable* and *controllable event variables.* Such a system is called a controllable polynomial dynamic system. Let n, m, and p be the respective dimensions of X, Y, and U. The trajectories of a controllable system are sequences (x_t, y_t, u_t) in $(\mathbb{Z}/_{3\mathbb{Z}})^{n+m+p}$ such that $Q_0(x_0) = 0$ and, for all t, $Q(x_t, y_t, u_t) = 0$ and $x_{t+1} = P(x_t, y_t, u_t)$. The events (y_t, u_t) include an uncontrollable component y_t and a controllable one u_t[3]. We have no direct influence on the y_t part which depends only on the state x_t, but we observe it. On the other hand, we have full control over u_t and we can choose any value of u_t which is admissible, *i.e.* , such that $Q(x_t, y_t, u_t) = 0$. To distinguish the two components, a vector $y \in (\mathbb{Z}/_{3\mathbb{Z}})^m$ is called an *event* and a vector $u \in (\mathbb{Z}/_{3\mathbb{Z}})^p$ a *control*. From now on, an event y is *admissible* in a state x if there exists a control u such that $Q(x, y, u) = 0$; such a control is said *compatible* with y in x.

The controllers: A PDS can be controlled by first selecting a particular initial state x_0 and then by choosing suitable values for $u_1, u_2, \dots, u_n, \dots$. We will here consider control policies where the value of the control u_t is instantaneously computed from the value of x_t and y_t. Such a controller is called a *static controller*. It is a system of two equations: $C(X,Y,U) = 0$ and $C_0(X) = 0$, where

[3] This particular aspect constitutes one of the main differences with [14]. In our case, the events are partially controllable, whereas in the other case, the events are either controllable or uncontrollable.

the equation $C_0(X) = 0$ determines initial states satisfying the control objectives and the other one describes how to choose the instantaneous controls; when the controlled system is in state x, and when an event y occurs, any value u such that $Q(x, y, u) = 0$ and $C(x, y, u) = 0$ can be chosen. The behavior of the system S composed with the controller is then modeled by the system S_c:

$$S_c = \begin{cases} X' = P(X, Y, U) & \\ Q(X, Y, U) = 0 & C(X, Y, U) = 0 \\ Q_0(X_0) = 0 & C_0(X_0) = 0 \end{cases} \tag{3}$$

However, not every controller (C, C_O) is acceptable. First, the controlled system S_C has to be initialized ; thus, the equations $Q_0(X) = 0$ and $C_0(X) = 0$ must have common solutions. Furthermore, due to the uncontrollability of the events Y, any event that the system S can produce must be admissible by the controlled system S_C. Such a controller is said to be *acceptable.*

5.2 Traditional Control Objectives

We now illustrate the use of the framework for solving a traditional control synthesis problem we shall reuse in the sequel.

Suppose we want to ensure the *invariance* of a set of states E. Let us introduce the operator $\widetilde{pre}$, defined by: for any set of states F,

$$\widetilde{pre}\ (F) = \{x \in (\mathbb{Z}/_{3\mathbb{Z}})^n \mid \forall y \text{ admissible}, \exists u, Q(x, y, u) = 0 \text{ and } P(x, y, u) \in F\}$$

Consider now the sequence $(E_i)_{i \in \mathbb{N}}$ defined by:

$$\begin{cases} E_0 &= E \\ E_{i+1} &= E_i \cap \widetilde{pre}\ (E) \end{cases} \tag{4}$$

The sequence (4) is decreasing. Since all sets E_i are finite, there exists a j such that $E_{j+1} = E_j$. The set E_j is then the greatest control-invariant subset of E. Let g_j be the polynomial that has E_j as solution, then $C_0(X) = g_j$ and $C(X, Y, U) = P^*(g_j)$[4] is an admissible feed-back controller and the system S_C : $S + (C_0, C)$ verifies the invariance of the set of states E.

Using similar methods, we are also able to to compute controllers (C, C_0) that ensure

- the *reachability* of a set of states from the initial states of the system,
- the *attractivity* of a set of states E from a set of states F.
- the *recurrence* of a set of states E.

We can also consider control objectives that are conjunctions of basic properties of state trajectories. However, basic properties cannot, in general, be combined in a modular way. For example, an invariance property puts restrictions on the

[4] the solutions of the polynomial $P^*(g)$ are the triples (x, y, u) that satisfy the relation "$P(x, y, u)$ is solution of the polynomial g".

set of state trajectories which may be not compatible with an attractivity property. The synthesis of a controller insuring both properties must be effected by considering both properties *simultaneously* and not by combining a controller insuring safety with a controller insuring attractivity independently. For more details on the way controllers are synthesized, the reader may refer to [4].

Specification of the control objectives: As for verification (Section 4), the control objectives can be directly specified in SIGNAL+ program, using the key-word `Sigali`. For example, if we add in the SIGNAL program the line `Sigali(S_Attractivity(S,PROP))`, the compiler produces a file that is interpreted by SIGALI which computes the controller with respect to the control objective. In this particular case, the controller will ensure the attractivity of the set of states `Set_States`, where `Set_States` is a polynomial that is equal to zero when the boolean `PROP` is *true*. The result of the controller synthesis is a polynomial that is represented by a Binary Decision Diagram (BDD). This BDD is then saved in a file that could be used to perform a simulation [11].

Application to the transformer station: We have seen in the previous section, that one of the most critical requirements concerns the double fault problem. We assume here that the circuit-breakers are ideal, i.e. they immediately react to actuators (*i.e.* , when a circuit-breaker receives an opening/closing request, then at the next instant the circuit-breaker is opened/closed). With this assumption, the double fault problem can be rephrased as follows:

"if two faults are picked up at the same time by two different departure cells, then at the next instant, one of the two faults (or both) must disappear."

In order to synthesize the controller, we assume that the only controllable events are the opening and closing requests of the different circuit-breakers. The other events concern the appearance of the faults and cannot be considered controllable. The specification of the control objective is then:

```
(| 2_Fault :=          when (Fault_Dep_1 and Fault_Dep_2)
              default  when (Fault_Dep_1 and Fault_Dep_3)
              default  when (Fault_Dep_1 and Fault_Dep_4)
              default  when (Fault_Dep_2 and Fault_Dep_3)
              default  when (Fault_Dep_2 and Fault_Dep_4)
              default  when (Fault_Dep_3 and Fault_Dep_4) default false
 | Z_2_Fault := 2_Fault $1 init false
 | Error := 2_Fault and Z_2_Fault
 | Sigali(S_Invariance(S,False(Error)) |)
```

The boolean `2_Fault` is *true*, when two faults are present at the same time and is *false* otherwise. The boolean `Error` is *true* when two faults are present at two consecutive instants. We then ask SIGALI to compute a controller that forces the boolean `Error` to be always *false* (i.e., whatever the behavior, there is no possibility for the controlled system to reach a state where `Error` is *true*).

The SIGNAL compiler translates the SIGNAL program into a PDS, and the control objectives in terms of polynomial relations and polynomial operations. Applying the algorithm, described by the fixed-point computation (4), we are able to synthesize a controller (C_1, C_0), that ensures the invariance of the set of states where the boolean *Error* is *true*, for the controlled system $S_{C_1} = S + (C_1, C_0)$. The result is a controller coded by a polynomial and a BDD.

Using the controller synthesis methodology, we solved the double fault problem. However, some requirements have not been taken into account (importance of the lines, of the circuit-breakers,...). This kind of requirements cannot be solved using traditional control objectives such as invariance, reachability or attractivity. In the next section, we will handle this kind of requirements, using control objectives expressed as order relations.

5.3 Numerical Order Relation Control Problem

We now present the synthesis of control objectives that considers the way to reach a given logical goal. This kind of control objectives will be useful in the sequel to express some properties of the power transformer station controller, as the one dealing with the importance of the different circuit-breakers. For this purpose we introduce cost functions on states. Intuitively speaking, the cost function is used to express priority between the different states that a system can reach in one transition. Let S be a PDS as the one described by (2). Let us suppose that the system evolves into a state x, and that y is an admissible event at x. As the system is generally not deterministic, it may have several controls u such that $Q(x, y, u) = 0$. Let u_1 and u_2 be two controls compatible with y in x. The system can evolve into either $x_1 = P(x, y, u_1)$ or $x_2 = P(x, y, u_2)$. Our goal is to synthesize a controller that will choose between u_1 and u_2, in such a way that the system evolves into either x_1 or x_2 according to a given choice criterion. In the sequel, we express this criterion as a cost function relation.

Controller synthesis method: Let $X = (X_1, \dots, X_n)$ be the state variables of the system. Then, a cost function is a map from $(\mathbb{Z}/_{3\mathbb{Z}})^n$ to $\mathbb{N}$, which associates to each x of $(\mathbb{Z}/_{3\mathbb{Z}})^n$ some integer k.

Definition 2. *Given a PDS S and a cost function c over the states of this system, a state x_1 is said to be c-better than a state x_2 (denoted $x_1 \succeq_c x_2$), if and only if, $c(x_2) \geq c(x_1)$.* ●

In order to express the corresponding order relation as a polynomial relation, let us consider $k_{max} = sup_{x \in (\mathbb{Z}/_{3\mathbb{Z}})^n}(c(x))$. The following sets of states are then computed $A_i = \{x \in (\mathbb{Z}/_{3\mathbb{Z}})^n \mid c(x) = i\}$. The sets $(A_i)_{i=0..k_{max}}$ form a partition of the global set of states. Note that some A_i could be reduced to the empty set. The proof of the following property is straightforward:

Proposition 1. $x_1 \succeq_c x_2 \Leftrightarrow \exists i \in [0, .., k_{max}],\ x_1 \in A_i \wedge x_2 \in \bigcup_{j=i}^{k_{max}} A_j$ ○

Let $g_0, \ldots, g_{k_{max}}$ be the polynomials that have the sets $A_1, \ldots, A_{k_{max}}$ as solutions[5]. The order relation $\succeq_c$ defined by the proposition 1 can be expressed as polynomial relation:

Corollary 1. *$x \succeq_c x' \Leftrightarrow R_{\succeq_c}(x, x') = 0$, where*

$$R_{\succeq_c}(X, X') = \prod_{i=1}^{n} \{g_i^2(X) \oplus (\prod_{j=i}^{n} (g_j^2(X')))\} \text{ with } f \oplus g = (f^2 + g^2)^2.$$

As we deal with a non strict order relation, from $\succeq_c$, we construct a strict order relation, named $\succ_c$ defined as: $x \succ_c x' \Leftrightarrow \{x \succeq_c x' \wedge \neg(x' \succeq_c x)\}$. Its translation in terms of polynomial equation is then given by:

$$R_{\succ_c}(X, X') = R_{\succeq_c}(X, X') \oplus (1 - R^2_{\succeq_c}(X', X)). \tag{5}$$

We now are interested in the direct control policy we want to be adopted by the system; *i.e.* , how to choose the right control when the system S has evolved into a state x and an uncontrollable event y has occurred.

Definition 3. *A control u_1 is said to be* better *compared to a control u_2, if and only if $x_1 = P(x, y, u_1) \succ_c x_2 = P(x, y, u_2)$. Using the polynomial approach, it gives $R_{\succ_c}(P(x, y, u_1), P(x, y, u_2)) = 0$.* •

In other words, the controller has to choose, for a pair (x, y), a compatible control with y in x, that allows the system to evolve into one of the states that are maximal for the relation $R_{\succ_c}$. To do so, let us introduce a new order relation $\sqsupset_c$ defined from the order relation $\succ_c$.

$$(x, y, u) \sqsupset_c (x', y', u') \Leftrightarrow \begin{cases} x = x' \\ y = y' \\ P(x, y, u) \succ_c P(x, y, u') \end{cases} \tag{6}$$

In other words, a triple (x, y, u) is “better” than a triple (x, y, u') whenever the state $P(x, y, u)$ reached by choosing the control u is better than the state $P(x, y, u')$ reached by choosing the control u'.

We will now compute the maximal triples of this new order relation among all of the triples. To this effect, we use $I = \{(x, y, u) \in (\mathbb{Z}/_{3\mathbb{Z}})^{n+m+p} \mid Q(x, y, u) = 0\}$ the set of admissible triples (x, y, u). The maximal set of triples I_{max} is then provided by the following relation:

$$I_{max} = I - \{(x, y, u) \mid \exists (x, y, u') \in I, (x, y, u') \sqsupset_c (x, y, u)\} \tag{7}$$

The characterization of the set of states I_{max} in terms of polynomials is the following:

[5] To compute efficiently such polynomials, it is important to use the Arithmetic Decision Diagrams (ADD) developed, for example, by [3].

Proposition 2. *The polynomial C that has I_{max} as solutions is given by:*

$$C(X,Y,U) = Q(X,Y,U) \oplus (1 - \exists elim_{U'}(Q(X,Y,U') \oplus R_{\succ_c}(P(X,Y,U'), P(X,Y,U))))$$

where the solutions of $\exists elim_{U'}(Q(X,Y,U')$ are given by the set $\{(x,y)/\exists u', Q(x,y,u') = 0\}$.

Using this controller, the choice of a control u, compatible with y in x, is reduced such that the possible successor state is maximal for the (partial) order relation $\succ_c$. Note that if a triple (x,y,u) is not comparable with the maximal element of the order relation $\sqsupset_c$, the control u is allowed by the controller (*i.e.* , u is compatible with the event y in the state x).

Without control, the system can start from one of the initial states of $I_0 = \{x \ / \ Q_0(x) = 0\}$. To determine the new initial states of the system, we will take the ones that are the maximal states (for the order relation $R_{\succ_c}$) among all the solutions of the equation $Q_0(X) = 0$. This computation is performed by removing from I_0 all the states for which there exist at least one smaller state for the strict order relation $\succ_c$. Using the same method as the one previously described for the computation of the polynomial C, we obtain a polynomial C_0. The solutions of this polynomial are the states that are maximal for the order relation $\sqsupset_c$.

Theorem 1. *With the preceding notations, (C, C_0) is an acceptable controller for the system S. Moreover, the controlled system $S_C = (S + (C, C_0))$ adopts the control policy of Definition 3.* ◦

Some others characterization of order relations in terms of polynomials can be found in [10]. Finally, note that the notion of numerical order relation has been generalized over a bounded states trajectory of the system, retrieving the classical notion of *Optimal Control* [9].

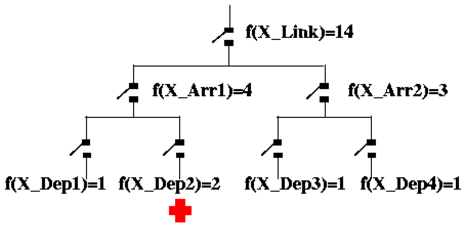
Application to the power transformer station controller: We have seen in Section 5.2 how to compute a controller that solves the double fault problem. However, even if this particular problem is solved, other requirements had not been taken into account. The first one is induced by the obtained controller itself. Indeed, several solutions are available at each instant. For example, when two faults appear at a given instant, the controller can choose to open all the circuit-breakers, or at least the link circuit-breaker. This kind of solutions is not admissible and must not be considered. The second requirements concerns the importance of the lines. The first controller (C_1, C_0) does not handle this kind of problems and can force the system to open the bad circuit-breakers.

As consequences, two new requirements must be added in order to obtain a real controller:

1. The number of opened circuit-breaker must be minimal
2. The importance of the lines (and of the circuit-breakers) has to be different.

These two requirements introduce a quantitative aspect to the control objectives. We will now describe the solutions we proposed to cope with these problems.

First, let us assume that the state of a circuit-breaker is coded with a state variable according to the following convention: the state variable i is equal to 1 if and only if the corresponding circuit-breaker i is closed. CB is then a vector of state variables which collects all the state variables encoding the states of the circuit-breakers. To minimize the number of open circuit-breaker and to take into account the importance of the line, we use a cost function . We simply encode the fact that the more important is the circuit-breaker, the larger is the cost allocated to the state variable which encodes the circuit-breaker. The following picture summarizes the way we allocate the cost.



The cost allocated to each state variable corresponds to the cost when the corresponding circuit-breaker is opened. When it is closed, the cost is equal to 0. The cost of a global state is simply obtained by adding all the circuit-breaker costs. With this cost function, it is always more expensive to open a circuit-breaker at a certain level than to open all the downstream circuit-breakers. Moreover, the cost allocated to the state variable that encodes the second departure circuit-breaker (encoded by the state variable X_{dep2})) is bigger than the others because the corresponding line supplies a hospital (for example). Finally note that the cost function is minimal when the number of open circuit-breaker is minimal.

Let us consider the system S_{C_1}. We then introduce an order relation over the states of the system: a state x_1 is said to be better compared to a state x_2 ($x_1 \sqsupseteq x_2$) if and only if for their corresponding sub-vectors CB_1 and CB_2, we have $CB_1 \sqsupseteq_c CB_2$. This order relation is then translated in an algebraic relation $R_{\sqsupseteq_c}$, following Equation (5) and by applying the construction described in proposition 2 and 1, we obtain a controller (C_2, C'_0) for which the controlled system $S_{C_2} = (S_{C_1} + (C_2, C'_0))$ respects the control strategy.

6 Conclusion

In this paper, we described the incremental specification of a power transformer station controller using the control theory concepts of the class of polynomial dynamical systems over $\mathbb{Z}/_{3\mathbb{Z}}$. As this model results from the translation of a SIGNAL program [8], we have a powerful environment to describe the model for a synchronous data-flow system. Even if classical control can be used, we have shown that using the algebraic framework, optimal control synthesis problem is possible. The order relation controller synthesis technique can be used to synthesize control objectives which relate more to the way to get to a logical goal, than to the goal to be reached.

Acknowledgment: The authors gratefully acknowledge relevant comments from the anonymous reviewers of this paper.

References

1. S. Balemi, G. J. Hoffmann, H. Wong-Toi, and G. F. Franklin. Supervisory control of a rapid thermal multiprocessor. *IEEE Transactions on Automatic Control*, 38(7):1040–1059, July 1993.
2. A. Benveniste and G. Berry. Real-time systems designs and programming. *Proceedings of the IEEE*, 79(9):1270–1282, September 1991.
3. R.E. Bryant and Chen Y. Verification of Arithmetic Functions with Binary Diagrams. Research Report, School of Computer Science CMU, May 1995.
4. B. Dutertre and M. Le Borgne. Control of polynomial dynamic systems: an example. Research Report 798, IRISA, January 1994.
5. L.E. Holloway, B.H. Krogh, and A. Giua. A survey of Petri net methods for controlled discrete event systems. *Discrete Event Dynamic Systems: Theory and Application*, 7:151–190, 1997.
6. M. Le Borgne, A. Benveniste, and P. Le Guernic. Polynomial dynamical systems over finite fields. In *Algebraic Computing in Control*, volume 165, pages 212–222. LNCIS, G. Jacob et F. Lamnabhi-lagarrigue, March 1991.
7. M. Le Borgne, H. Marchand, E. Rutten, and M. Samaan. Formal verification of signal programs: Application to a power transformer station controller. In *Proceedings of AMAST'96*, pages 271–285, Munich, Germany, July 1996. Springer-Verlag, LNCS 1101.
8. P. Le Guernic and T. Gautier. Data-flow to von Neumann: the SIGNAL approach. In Jean-Luc Gaudiot and Lubomir Bic, editors, *Advanced Topics in Data-Flow Computing*, chapter 15, pages 413–438. Prentice-Hall, 1991.
9. H. Marchand and M. Le Borgne. On the optimal control of polynomial dynamical systems over $\mathbb{Z}/_{p\mathbb{Z}}$. In *4th International Workshop on Discrete Event Systems*, pages 385–390, Cagliari, Italy, August 1998.
10. H. Marchand and M. Le Borgne. Partial order control of discrete event systems modeled as polynomial dynamical systems. In *1998 IEEE International Conference On Control Applications*, Trieste, Italia, September 1998.
11. H. Marchand, Bournai P., M. Le Borgne, and P. Le Guernic. A design environment for discrete-event controllers based on the signal language. In *1998 IEEE International Conf. On Systems, Man, And Cybernetics*, pages 770–775, San Diego, California, USA, October 1998.
12. H. Marchand, E. Rutten, and M. Samaan. Synchronous design of a transformer station controller with Signal. In *4th IEEE Conference on Control Applications*, pages 754–759, Albany, New-York, September 1995.
13. H. Melcher and K. Winkelmann. Controller synthesis for the production cell case study. In *Proceedings of the 2nd Workshop on Formal Methods in Software Practice (FMSP-98)*, pages 24–33, New YOrk, March 4–5 1998. ACM Press.
14. P. J. Ramadge and W. M. Wonham. The control of discrete event systems. *Proceedings of the IEEE; Special issue on Dynamics of Discrete Event Systems*, 77(1):81–98, 1989.