Incremental Design of a Power Transformer Station Controller using a Controller Synthesis Methodology

Hervé Marchand, Mazen Samaan

Incremental Design of a Power Transformer Station Controller using a Controller Synthesis Methodology

Hervé Marchand and Mazen Samaan

Abstract—In this paper, we describe the incremental specification of a power 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 techniques, the controller is automatically derived from this set of control objectives. 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, ...* properties, as well as *partial order relations* to be checked by the plant. The control objectives equations are synthesized using algebraic transformations.

Index Terms—Discrete event systems, polynomial dynamical system, supervisory control problem, optimal control, SIGNAL, SIGALI, power plant.

---

1 INTRODUCTION & MOTIVATIONS

THE SIGNAL language [1] is developed for precise specification of real-time reactive systems [2]. In such systems, requirements are usually checked a posteriori using property verification and/or simulation techniques. The control theory of Discrete Event Systems allows us 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.

Different theories for control of Discrete Event Systems have existed since the 1980s [3], [4], [5]. Here, we choose to specify the plant in SIGNAL and the control synthesis, as well as verification performed on a logical abstraction of this program, called a polynomial dynamical system (PDS), over $\mathbb{Z}/_3\mathbb{Z}$ (i.e., integers modulo 3: {-1,0,1}). The control 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 uses classical tools in algebraic geometry, such as ideals, varieties, and morphisms. This theory sets the basis for our 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.

The methodology is the following (see Fig. 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 Ternary Decision Diagram (TDD), a slight extension of the Binary Decision Diagrams (BDDs).

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 interaction between communicating automata, interruption and preemption behaviors, timers and timeouts, and reactivity to external events, among others. The functionality of the controller is to handle the power interruption, the redirection of supply sources, and the reestablishment of the power following an interruption. The objective is twofold: the safety of material and the best uninterrupted service. The safety of material can be achieved by triggering circuit breakers when an electric fault occurs, whereas the best quality service can be achieved by minimizing the number of costumers concerned by a power cut and reestablishment 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 subnetwork).

2 OVERVIEW OF THE POWER TRANSFORMER STATION

In this section, we give a brief description of the power transformer station network, as well as the various requirements the controller has to handle.
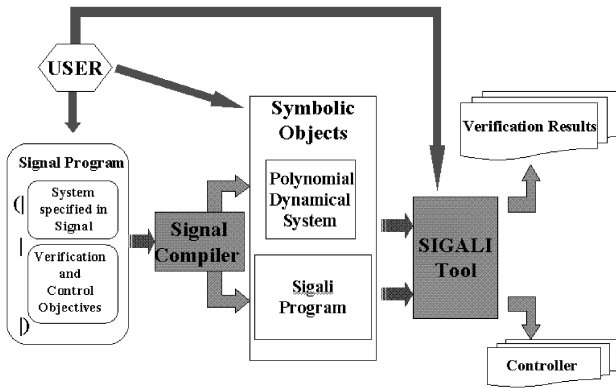
- H. Marchand is with IRISA/INRIA-Rennes, F-35042 Rennes, France. E-mail: hmarchan@irisa.fr.
- M. Samaan is with EDF, Unité National Technique Système, CAP AMPERE 1, Place Pleyel, 93282, Saint-Denis cedex, France. E-mail: Mazen.Samaan@edf.fr.

Fig. 1. Description of the tool.

2.1 The Power Transformer Station

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 Fig. 2) we consider receives high voltage lines and feeds several medium voltage lines to distribute power to end-users.

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 link lines, arrival lines, and departure lines). They are informed about the possible presence of faults by sensors.

2.1.1 Power and Fault Propagation

Here, we discuss 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
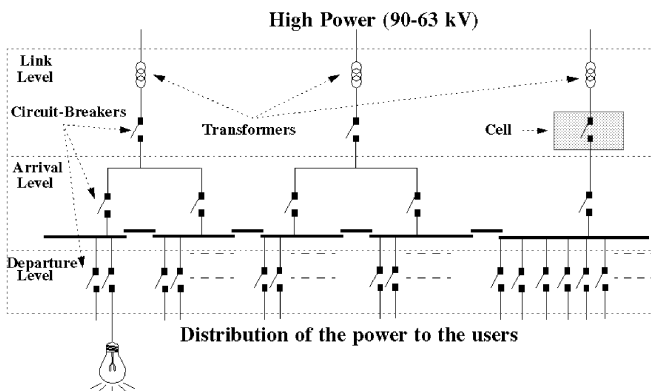

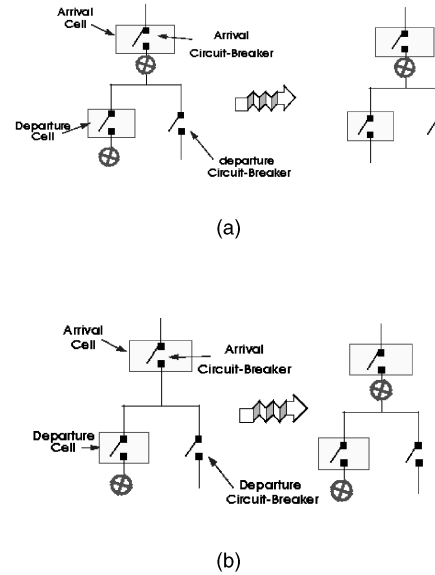Fig. 2. The power transformer station topology.

Fig. 3. The fault properties. (a)The fault masking. (b)The fault propogation.

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 Fig. 3a, for example) increases the voltage on the downstream lines and masks all the other possible faults.
- 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 Fig. 3b) 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 handles 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. Here, we give only a brief description of the behavior of the different cells (more details can be found in [6], [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 next section.

2.2.1 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 to let the transient faults disappear

spontaneously. If the fault is confirmed, it is handled by opening the circuit-breaker for a certain number of periods during a given delay 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 [6] and verified using our formal calculus system, named SIGALI [7].

2.2.2 Some Global Requirements for the Controller

Even if it 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 the fault location, 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 a departure line, involved in a double faults problem, supplies a hospital. Then, if 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 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 LANGUAGE

SIGNAL [1] is built around a minimal kernel of operators. It manipulates *signals* X, which denote unbounded series of typed values $(x_t)_{t \in T}$, 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 always has the value *true* (hence, its negation by not is always *false*).

TABLE 1

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

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 of the composition of the systems of equations; it is associative and commutative. It can be interpreted as parallelism between processes.

Table 1 shows each of the primitives with a trace.

3.1.1 Derived Features

Derived processes have been defined on the base of the primitive operators, providing programming comfort. For example, the instruction $X^\wedge = Y$ 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 [1]. The complete programming environment also features a block-
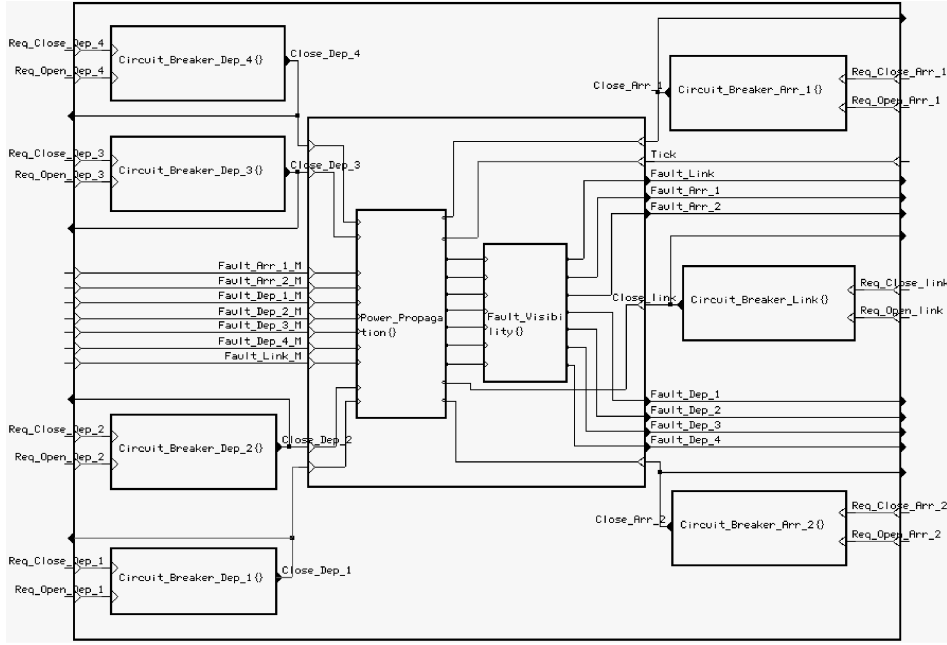
Fig. 4. The main process in SIGNAL.

diagram oriented graphical user interface and a proof system for dynamic properties of SIGNAL programs, called SIGALI (see Section 4).

3.2 The Graphical Interface

The complete programming environment of SIGNAL also contains a graphical, block-diagram oriented, user interface [8], as illustrated in Fig. 4. This environment allows the user to have graphical and textual representations of the language structures. These representations may be used together during the building or the "reading" of the program. A SIGNAL expression may be considered as a set of components (represented as boxes) with connection points (ports represented with triangles) joined by links. The same SIGNAL expression is also a system of equations on series of values; this system is represented by a phrase built with the language operators (definitions of variables, composition, renaming, etc.)

This graphical interface will be used further in order to perform the specification of the system to be controlled, as well as the automatic simulation of the controlled system.

3.3 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 Fig. 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.

The inputs of this main process are Booleans that encode the physical faults: Fault_Link_M, Fault_Arr_i_M (i=1,2), and Fault_Dep_j_M (j=1,..,4). They encode faults that are actually 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, and Fault_Dep_j, representing the signals that are sent to the different cells. They indicate whether a cell is faulty or not. These outputs represent the knowledge that the sensors have of the different cells. We will now see how the subprocesses are specified in SIGNAL.

3.3.1 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. (See Fig. 5.)

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.

```
(| 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)
 |)
```

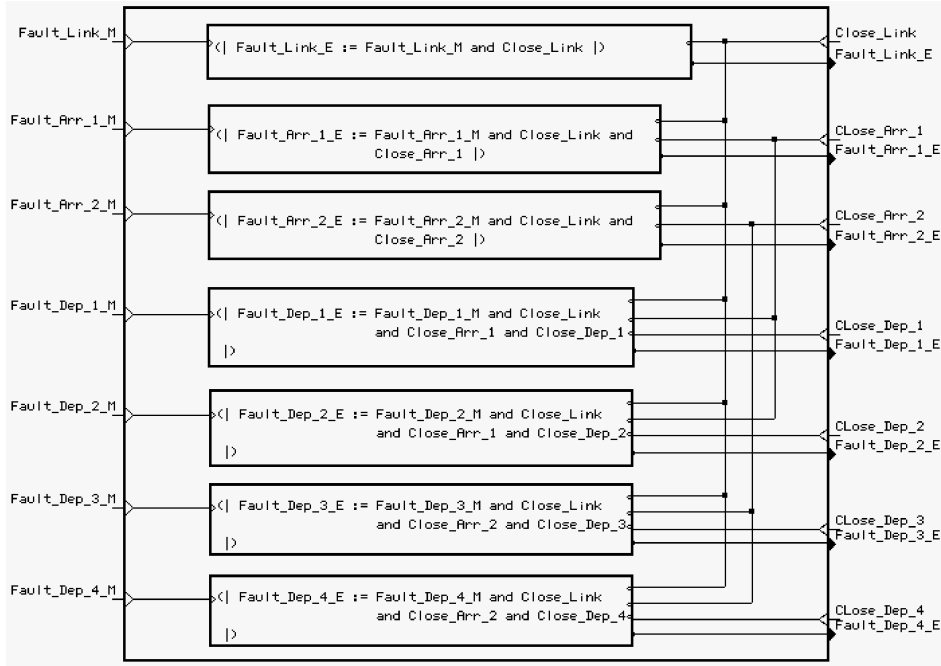Fig. 5. The circuit-breaker in SIGNAL.

Fig. 6. Specification in SIGNAL of the power propagation.

```
(| 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.

3.3.2 Power Propagation

Power Propagation is a filter process using the state of the circuit-breakers. It 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.

This is specified in the process `Power_Propagation` shown in Fig. 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 (i.e., `Close_Link` = *true* and `Close_Arr_1`=*true* and `Close_Dep_1` = *true*).

3.3.3 Fault Visibility and Propagation

The process in Fig. 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.

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 *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 tool-box, SIGALI. This tool allows us to prove the correctness of the dynamical behavior of the system. The equational nature of the SIGNAL language leads naturally to the use of a method, based on polynomial dynamical equation systems (PDSs) 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 [9]. The techniques consist of manipulating the system of equations instead of the sets of solutions, which avoids enumerating state spaces. More precisely, a set of states and/or events can actually be represented by a unique polynomial named *principal generator*. This way we can perform operations on sets while still remaining in the domain of polynomial functions while not having to

---

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

enumerate them. The tool, SIGALI, implements the basic operators; set theoretic operators, fix-point computation, and quantifiers [10]. It relies on an implementation of polynomials by Ternary Decision Diagram (TDD) (for three valued logics) in the same spirit of BDD [11], but where the paths in the data structures are labeled by values in $\{-1, 0, 1\}$ instead of $\{0, 1\}$.

4.1 From a SIGNAL Program to a Polynomial Dynamical System

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 the 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 $c = a$.

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, 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 2 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.

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

$$S = \begin{cases} X' &= P(X, Y) \\ Q(X, Y) &= 0 \\ Q_0(X_0) &= 0, \end{cases} \quad (1)$$

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*. In the following, x, x_t, y, y_t will denote particular

TABLE 2
Translation of the Primitive Operators

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	$=$	$a + (1 - a^2)b$
C	:=	A when B	c	$=$	$a(-b - b^2)$
B	:=	A \$1 (init b_0)	x'	$=$	$a + (1 - a^2)x$
			b	$=$	$a^2 x$
			x_0	$=$	b_0
non-boolean expressions					
B	:=	$f(A_1, \ldots, A_n)$	b^2	$=$	$a_1^2 = \cdots = a_n^2$
C	:=	A default B	c^2	$=$	$a^2 + b^2 - a^2 b^2$
C	:=	A when B	c^2	$=$	$a^2(-b - b^2)$
B	:=	A \$1 (init b_0)	b^2	$=$	a^2

instantiations of vector X and Y. 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 $x_{t+1} = P(x_t, y_t)$.

4.2 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. Here, we just give 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 [9], [7].

4.2.1 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, attractivty, etc.) means that SIGNALI 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 by 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 a SIGNAL program named "system" to check the attractivity of the set of states where the Boolean PROP is *true*. Then the corresponding SIGNAL + program is:

```
(| system() (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 of 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 which checks the verification objective.

4.3 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. Its computation is realized in less than five seconds. 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 proven.

4.3.1 Property One

"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 which 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.

4.3.2 Property Two

"If there exists a physical fault at the link level and if this fault is picked up by its sensor, then the arrival sensors cannot 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)))
 |).
```

We have proven 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.

4.3.3 Property Three

Using the same methods, we also proved 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 proven in this way. Note that the cell behaviors have also been proven (see [7], [12] for more details).

5 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:

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

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

---

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

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.

5.1.1 The Controllers

A PDS can be controlled by first selecting a particular initial state x_0 and then by choosing suitable values for $u_1, u_2, \ldots, u_n, \ldots$. Here we will 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 $C_0(X_0) = 0$, where the equation $C_0(X_0) = 0$ 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.

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

However, not every controller (C, C_O) is acceptable. First, the controlled system S_C has to be initialized. Thus, the equations $Q_0(X_0) = 0$ and $C_0(X_0) = 0$ 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. This remark leads to the definition of acceptable controller: be *acceptable*.

Definition 2. *An acceptable controller for a system S, defined by (2), is given by an initial constraint equation $C_0(X_0) = 0$ and a control equation $C(X, Y, U)$ such that:*

1. *The initial constraints $C_0(X_0) = 0$ and $Q_0(X_0) = 0$ have common solutions;*
2. *For all the states x which can be reached during the evolution of the controlled system, any uncontrollable event y admissible in x for the initial system S is also admissible for the controlled system S_C.*

5.2 Traditional Control Objectives

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

Suppose we want to ensure the *invariance* of a set of states $\mathcal{O}$. Let $\mathcal{O}_{S_c}$ be the orbit[4] of the controller system. The problem is to compute (C, C_0) such that $\mathcal{O}_{S_c} \subseteq \mathcal{O}$. Assume that there exists an acceptable controller (C, C_0) which satisfies this property. In this case, we know that,

1. $\mathcal{O}_{S_c} \subseteq \mathcal{O}$
2. $\mathcal{O}_{S_c}$ invariant for S_c, which means that

$$\forall x \in \mathcal{O}_{S_c}, \forall y \in (\mathbb{Z}/_{3\mathbb{Z}})^m, \forall u \in (\mathbb{Z}/_{3\mathbb{Z}})^p, Q(x, y, u) = 0 \\ \text{and } C(x, y, u) = 0 \Rightarrow P(x, y, u) \in \mathcal{O}_{S_c}. \quad (4)$$

Now, let x be a state of $\mathcal{O}_{S_c}$ and y an event admissible in x by the system S. Since (C, C_0) constitutes an acceptable controller, y is also admissible in x for the controlled system S_c. Then, there exists a command u such that $Q(x, y, u) = 0$ and $C(x, y, u) = 0$, and, for this command, $P(x, y, u)$ belongs to $\mathcal{O}_{S_c}$. For the system S, $\mathcal{O}_{S_c}$ possesses a property similar to invariance. We say that $\mathcal{O}_{S_c}$ is *U-invariant under control* for the system S.

Definition 3. *A set of states $\mathcal{O}$ is* U-invariant under control *for a system S if, for every state x in $\mathcal{O}$ and every event y admissible in x, there exists a command u such that $Q(x, y, u) = 0$ and $P(x, y, u) \in \mathcal{O}$.*

If there exists an acceptable controller which ensures the invariance of $\mathcal{O}$, then the orbit $\mathcal{O}_{S_c}$ of S_c is included in $\mathcal{O}$ and is U-invariant under control for S. It is also necessary to have some initial states of S included in $\mathcal{O}$ ($\mathcal{O} \cap \mathcal{V}(<Q_0>) \neq \emptyset$), where $\mathcal{V}(<Q_0>) = \{x/Q_0(x_0) = 0\}$. The essential result is that these three properties are sufficient.

Theorem 1. *Given a controlled system S and $\mathcal{O}$, a set of states of S, there exists an acceptable controller which ensures the invariance of $\mathcal{O}$ if and only if there exists a set of states E such that:*

$\alpha 1$. *$E \subseteq \mathcal{O}$,*
$\alpha 2$. *$E \cap \mathcal{V}(<Q_0>) \neq \emptyset$,*
$\alpha 3$. *E is U-invariant under control for S.*

Proof. We have already shown that the conditions are necessary. Conversely, assume there exists a set E which satisfies the three properties. Let C_0 be a principal generator of E and $C = P^*(C_0)$, where $P^*(g)$ is the polynomial which has, as solutions, the set $\{(x, y, u) / P(x, y, u) \text{ is solution of } g\}$, which is called comorphism of g. By construction, we have

$$\begin{cases} C_0(x_0) = 0 & \Leftrightarrow & x_0 \in E, \\ C(x, y, u) = 0 & \Leftrightarrow & P(x, y, u) \in E. \end{cases} \quad (5)$$

It follows that the orbit of S_c is included in E and, hence, in $\mathcal{O}$, the controller (C, C_0) ensures the invariance of $\mathcal{O}$. Now, let x be a state in the orbit of S_c and y be an event admissible in x for S. x is also an element of E and, since E is U-invariant under control, there exists a command u such that $Q(x, y, u) = 0$ and $P(x, y, u) \in E$. This is equivalent to $Q(x, y, u) = 0$ and $C(x, y, u) = 0$. y is

---

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

4. We recall that the orbit of a polynomial dynamical system S is the set of states that are reachable from one of the initial states (i.e., the solutions of the polynomial $Q_0(X_0) = 0$).

then also admissible in x for S_c. Since the condition $E \cap \mathcal{V}(<Q_0>) \neq \emptyset$ means that $C_0(X_0) = 0$ and $Q_0(X_0) = 0$ have common solutions, the controller (C, C_0) is acceptable. □

The proof gives an algorithm to obtain a controller, ensuring the invariance of a set of states $\mathcal{O}$. It suffices to find a U-invariant under control subset of $\mathcal{O}$ which satisfies the initialization condition ($\alpha 2$). $\mathcal{O}$ contains at least one U-invariant under control, the empty set. It is easy to see that the union of two U-invariant under control sets is also U-invariant under control. As a consequence, there exists a greatest U-invariant under control subset of $\mathcal{O}$. Let E be this subset. If E satisfies the condition ($\alpha 2$), then control equations can be obtained from the principal generator of E, else no subset of $\mathcal{O}$ can satisfy condition ($\alpha 2$) and the problem has no solution.

The computation of E uses the operator $\widetilde{pre}$ defined by, for any set of states F,

$$\widetilde{pre}(F) = \{x \in (\mathbb{Z}/_3\mathbb{Z})^n \ / \ \forall y \in (\mathbb{Z}/_3\mathbb{Z})^m, Q'(x,y) = 0 \\ \Rightarrow \exists u, Q(x,y,u) = 0 \text{ and } P(x,y,u) \in F\}, \quad (6)$$

where the solutions of the polynomials Q' are the set $\{(x,y)/\exists u, Q(x,y,u) = 0\}$. From Definition 1, it is clear that a set of states F is U-invariant under control if and only if $F \subseteq \widetilde{pre}(F)$. The greatest U-invariant under control subset of $\mathcal{O}$ is obtained by constructing the sequence $(E_i)_{i \in \mathbb{N}}$ defined by,

$$\begin{cases} E_0 & = & \mathcal{O} \\ E_{i+1} & = & E_i \cap \widetilde{pre}(E_i). \end{cases} \quad (7)$$

The sequence is decreasing. Since all sets E_i are finite, there exists an index j such that $E_{j+1} = E_j$. The set E_j is the greatest U-invariant under control subset of $\mathcal{O}$; E is equal to E_j. In practice, we transform this computation to an equivalent sequence of polynomials $(g_i)_{i \in \mathbb{N}}$, where g_j is a polynomial having as solutions the set of states E_j [13], [10]. Once obtained, the greatest U-invariant under control subset E of $\mathcal{O}$, the controller is simply given by the system of equations described by (5).

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

- the *global reachability* (resp. *attractivity*) of a set of states from the initial states of the system.
- the *persistence of a set of states*. A set of states E if it is attractive from the initial states and if E is invariant.
- the *recurrence of a set of states*. A set of states E is recurrent if it is visited infinitely often.

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 set of state trajectories which may not be 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 [10].

5.2.1 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 a SIGNAL program the line `Sigali(S_Reachability(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 reachability 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 Ternary Decision Diagram (TDD). This TDD is then saved in a file that could be used to perform a simulation [14].

5.2.2 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 always be *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 (7), 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, therefore, by a TDD.

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, etc., ...). These 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. These 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.

5.3.1 Controller Synthesis Method

Let $X = (X_1, \ldots, X_n)$ 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 4. *Given a PDS S and a cost function c, 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 Proposition 1 can be expressed as:

---

5. To compute efficiently such polynomials, it is important to use the Arithmetic Decision Diagrams (ADD) developed, for example, by [15].

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

$$R_{\succeq_c}(X, X') = \prod_{i=1}^n \{g_i^2(X) \oplus (\prod_{j=i}^n (g_j^2(X')))\}, \qquad (8)$$

with $f \oplus g = (f^2 + g^2)^2$.

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

$$R_{\succ_c}(X, X') = R_{\succeq_c}(X, X') \oplus (1 - R_{\succeq_c}^2(X', X)) \qquad (9)$$

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 5. *For a given state x and admissible event y, a control u_1 is said to be better compared to a control u_2 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, 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 $\succ_c$.

$$(x, y, u) \sqsupset_c (x', y', u') \Leftrightarrow \begin{cases} x = x' \\ y = y' \\ P(x, y, u) \succ_c P(x, y, u') \end{cases} \qquad (10)$$

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:

$$I_{max} = I - \{(x, y, u) \mid \exists (x, y, u') \in I, \\ (x, y, u') \sqsupset_c (x, y, u)\} \qquad (11)$$

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

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 $\rceil_c$, 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 $I_0 = \{x_0 \ / \ Q_0(x_0) = 0\}$. 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 $Q_0(X_0) = 0$. 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 C, we obtain a polynomial C_0. The solutions of this polynomial are the states that are maximal for the order relation $\rceil_c$.

Theorem 2. *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 5.*

Some other 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* [16].

5.3.2 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 concern 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 a consequence, two new requirements must be added in order to obtain a real controller.

1. The number of opened circuit-breakers 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 one 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
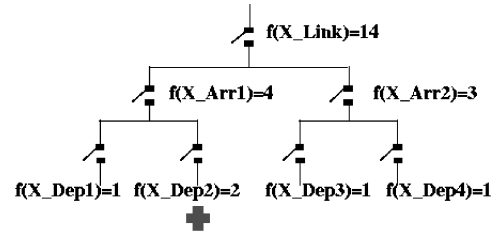

Fig. 8. Cost allocated to the state variables.

encoding the states of the circuit-breakers. To minimize the number of open circuit-breakers and to take into account the importance of the line, we use a cost function f. We simply encode the fact that the more important is the circuit-breaker, the cost allocated is larger compared to the state variable which encodes the circuit-breaker. Fig. 8 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 zero. 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 larger 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-breakers is minimal.

Let us consider the system $S_{C_1} = S + ((C_1, C_0))$. 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 $(x_1 \succeq_c x_2)$ if and only if, for their corresponding subvectors CB_1 and CB_2, we have $CB_1 \succeq_c CB_2$. This order relation is then translated in an algebraic relation $R_{\succ_c}$, following (8) and (9) and, by applying the construction described in Proposition 2 and Theorem 2, 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 $\mathbb{Z}/_3\mathbb{Z}$. Because this model results from the translation of a SIGNAL program [1], 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, the 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.

TABLE 3
Some Useful SIGALI Functions

declare(var,var)$\rightarrow$ Lvar	declaration of variables
union_lvar(Lvar,Lvar)$\rightarrow$ Lvar	performs the union of variable lists
diff_lvar(Lvar,Lvar)$\rightarrow$ Lvar	diff_lvar(L1,L2); returns the sublist of L1 from which the variables of L2 have been removed
comor(P)$\rightarrow$ Poly	comor(E) = $P^*(E)$: gives access to the comorphism of E
implies(Poly,Poly)$\rightarrow$ Poly	implies(P1,P2)=(1-P1^2)P2
complementary(Poly)$\rightarrow$ Poly	complementary(P)=(1-P1^2)
intersection(Poly,Poly,...)$\rightarrow$ Poly	P : intersection(P1,P2)$\Leftrightarrow Sol(P) = Sol(P1) \cap Sol(P2)$
union(Poly,Poly,...)$\rightarrow$ Poly	P : union(P1,P2)$\Leftrightarrow Sol(P) = Sol(P1) \cup Sol(P2)$
exist(LVAR,Poly)$\rightarrow$ Poly	exist(L,P): Existential elimination over the polynomial P w.r.t. the variables of L
forall(LVAR,Poly)$\rightarrow$ Poly	forall(L,P):Universal elimination over the polynomial P w.r.t. L
rename(Poly,Lvar1,Lvar2)$\rightarrow$ Poly	rename(P,L1,L2): renaming of the variables L1 by the variables L2 in P

APPENDIX A

THE SIGALI TOOL BOX

The SIGALI tool box offers algebraic polynomial computation functionalities. It relies on an implementation of polynomials by Ternary Decision Diagram (TDD) (for three valued logics) in the same spirit of BDD [11], but where the paths in the data structures are decorated by values in $\{-1, 0, 1\}$ instead of $\{0, 1\}$.[6]

From a practical point of view, a polynomial dynamical system can be obtained for "free" provided we have specified the system in the high level language SIGNAL [1], [18]. In fact, the equational nature of SIGNAL leads naturally to the use of a method based on polynomial dynamical equation systems over $\mathbb{Z}/_{3}\mathbb{Z}$ as a formal model of program behavior [7]. The model essentially expresses Boolean data and synchronizations. There exists a lot of examples using SIGALI for SIGNAL programs, among them, a production cell [19], a power transformer station controller [7], an experiment with reactive tasking in active robot-vision [20], etc., ...

A.1 Some Elements of the Sigali Syntax

A.1.1 The Basic Syntax

We can write polynomial expressions, lists of polynomials, etc. All the usual polynomial operations are also available (+, -, *, ...). For example, the polynomial $a^2(-b - b^2)$ is written a^2*(-b-b^2). The list of variables, polynomials, equations, etc., is written as follows: [a,b,c,d] is a set of variables, [a+b,c+d] is a list of polynomials, and [a+b=x,a*d^2=b^2] is an equation system. As for the polynomial operations, all the possible operations over lists have been defined (union, intersection, complementary, etc., ...). The function call is classically written f(x,y,z). For example, in order to check the invariance of a set of states defined by a polynomial g w.r.t. an ILTS S, we write:

$$\text{Invariant}(S,g); .$$

The result of such a computation is either *true* or *false*.

The affectation is simply written as follows:

```
symbol : expression;
```

For example, g: gen([a+b=x,a*d^2=b^2]); will attach the name g to the principal generator of this equation system (which is then computed).

Fix point computation can also be performed. For example, given: $p_0 = 0$, $p_{i+1} = p_i^2 + 1$, the corresponding expression in SIGALI is loop x=x^2+1 init 0;. Of course, such sequences do not always converge. This is not checked by the system.

A.1.2 Some Useful Functions.

There exist more than 120 different functions that belong to the kernel of the SIGALI languages. We just provide the one that is used in Sections 1 and 2 (See Table 3).

Moreover, starting from the existing functions, it is also possible to define new functions. The syntax is the following: def f(x,y,z):expression;.

A.2 Ensuring the Invariance.

We here assume that the polynomial dynamical system has already been defined in SIGALI. It is formally given by:

$$S : \begin{cases} Q(X,Y,U) & = & 0 \\ X' & = & P(X,Y,U) \\ Q_0(X_0) & = & 0, \end{cases} \quad (12)$$

where X, X', Y and U are lists of variables, P is a list of polynomial functions, Q is a polynomial, as well as Q_0. The complete PDS S is stored in another list given by

```
S : system(union_lvar(Y,U), X, P, Ini, Q);
```

To compute the controller that ensures the invariance of a set of states modeled by a polynomial Prop, we first need to compute the operator $\widetilde{pre}$. It is done as follows:

```
def Pre_cont(x) :
  forall(Y,implies(exists(U,Q),
              exists(U,intersection(Q,
                           comor(x))))));
```

Using this operator and a fix point computation, according to (7), we are able to compute the greatest U invariant under control of a given set of states, say E (E as to be understood as a polynomial).

```
def S_Invariant(E) :
   loop y = intersection(E, Pre_cont(y)) init E;
```

Finally, according to Section 5.2 a controller is given by :

```
Co : S_Invariant(E); C : comor(Co);
```

and the controlled system by:

```
S_c: system(union_lvar(Y,U), X, P,
       intersection(Co,Ini),
                     intersection(Q,C));
```

ACKNOWLEDGMENTS

REFERENCES

[1] P. Le Guernic, T. Gautier, M. Le Borgne, and C. Le Maire, "Programming Real-Time Applications with Signal," Technical Report 582, IRISA, Apr. 1991.

[2] A. Benveniste and G. Berry, "Real-Time Systems Designs and Programming," *Proc. IEEE,* vol. 79, no. 9, pp. 1270–1282, Sept. 1991.

[3] P.J. Ramadge and W. M. Wonham, "The Control of Discrete Event Systems," *Proc. IEEE; Special Issue on Dynamics of Discrete Event Systems,* vol. 77, no. 1, pp. 81–98, 1989.

[4] 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,* vol. 7, pp. 151–190, 1997.

[5] H. Melcher and K. Winkelmann, "Controller Synthesis for the Production Cell Case Study," *Proc. Second Workshop Formal Methods in Software Practice,* pp. 24–33, Mar. 1998.

[6] H. Marchand, E. Rutten, and M. Samaan, "Synchronous Design of a Transformer Station Controller with Signal," *Proc. Fourth IEEE Conf. Control Applications,* pp. 754–759, Sept. 1995.

[7] M. Le Borgne, H. Marchand, E. Rutten, and M. Samaan, "Formal Verification of Signal Programs: Application to a Power Transformer Station Controller," *Proc. Fifth Int'l Conf. Algebraic Methodology and Software Technology (AMAST'96),* pp. 271–285, July 1996.

[8] P. Bournai and P. Le Guernic, "Un environnement graphique pour le langage signal," Technical Report 741, IRISA, Sept. 1993.

[9] M. Le Borgne, A. Benveniste, and P. Le Guernic, "Polynomial Dynamical Systems over Finite Fields," *Algebraic Computing in Control,* pp. 212–222, Mar. 1991.

[10] H. Marchand and M. Le Borgne "The Supervisory Control Problem of Discrete Event Systems Using Polynomial Methods," Research Report 1271, IRISA, Oct. 1999.

[11] R.E. Bryant, "Graph-Based Algorithms for Boolean Function Manipulations," *Proc. IEEE Trans. Computers,* vol. 45, no. 8, pp. 677–691, Aug. 1986.

[12] H. Marchand, E. Rutten, and M. Samaan, "Specifying and Verifying a Transformer Station in Signal and Signalgti," Research Report 2521, INRIA, Mar. 1995.

[13] B. Dutertre and M. Le Borgne, "Control of Polynomial Dynamic Systems: An Example," Research Report 798, IRISA, Jan. 1994.

[14] H. Marchand, P. Bournai, M. Le Borgne, and P. Le Guernic, "A Design Environment for Discrete-Event Controllers Based on the Signal Language," *Proc. IEEE Int'l Conf. Systems, Man, and Cybernetics,* pp. 770–775, Oct. 1998.

[15] R.E. Bryant and Y. Chen, "Verification of Arithmetic Functions with Binary Diagrams," Research Report, School of Computer Science, Carnegie Mellon Univ., May 1995.

[16] H. Marchand and M. Le Borgne, "On the Optimal Control of Polynomial Dynamical Systems Over z/pz," *Proc. Fourth Int'l Workshop Discrete Event Systems,* pp. 385–390, Aug. 1998.

[17] H. Marchand and M. Le Borgne, "Partial Order Control of Discrete Event Systems Modeled as Polynomial Dynamical Systems," *Proc. IEEE Int'l Conf. Control Applications,* Sept. 1998.

[18] A. Benveniste, T. Gautier, P. Le Guernic, and E. Rutten, "Distributed Code Generation of Dataflow Synchronous Programs: The Sacres Approach," *Proc. 11th Int'l Symp. Languages for Intensional Programming,* Palo Alto, Calif., May 1998.

[19] T. Amagbegnon, P. Le Guernic, H. Marchand, and E. Rutten, "Signal—The Specification of a Generic, Verified Production Cell Controller," *Formal Development of Reactive Systems—Case Study Production Cell,* chapter VII, pp. 115–129, Springer-Verlag, Jan. 1995.

[20] E. Marchand, E. Rutten, H. Marchand, and F. Chaumette, "Specifying and Verifying Active Vision-Based Robotic Systems with the Signal Environment," *Int'l J. Robotics Research,* vol. 17, no. 4, pp. 418–432, Apr. 1998.

Hervé Marchand received the master's degree in mathematics from the Université de Rennes 1 in 1993 and a PhD in computer science from the University of Rennes 1 in 1997. From November 1997 through October 1998, he was a postdoctoral fellow at the University of Michigan, Ann Arbor. Since 1998, he has held an INRIA research position at IRISA, Rennes, on the Real-Time Programming team, where the SIGNAL language is being designed. His research interests include discrete events systems, polynomial dynamical systems, supervisory control problems, and optimal control. He is also interested in high-level languages for reactive and real-time systems programming.

Mazen Samaan graduated from the École Nationale Supérieure de Techniques Avancées in 1983. He performed his doctorial thesis in 1989 at the Institut National Polytechnique de Grenoble. Since 1990, he has been an engineer/researcher at EDF. He worked on CCS-based validation of communication protocol and on the application of synchronous technology to control and monitor electric power transformation posts. Since 1995, he has managed action on using formal description techniques for analyzing control and monitoring systems for electric power generators. In this framework, his research interests concern both advanced regulation and analysis of automatic control systems.