Synthesis of Discrete-Event Controllers based on the Signal Environment

Hervé Marchand **hmarchan@irisa.fr**
IRISA / INRIA - Rennes, Campus Univ. de Beaulieu, F-35042 Rennes, France

Patricia Bournai **bournai@irisa.fr**
IRISA / INRIA - Rennes, Campus Univ. de Beaulieu, F-35042 Rennes, France

Michel Le Borgne **leborgne@irisa.fr**
IRISA / INRIA - Rennes, Campus Univ. de Beaulieu, F-35042 Rennes, France

Paul Le Guernic **leguerni@irisa.fr**
IRISA / INRIA - Rennes, Campus Univ. de Beaulieu, F-35042 Rennes, France

Abstract. In this paper, we present the integration of controller synthesis techniques in the SIGNAL environment through the description of a tool dedicated to the incremental construction of reactive controllers. The plant is specified in SIGNAL and the control synthesis is performed on a logical abstraction of this program, named polynomial dynamical system (PDS) over $\mathbb{Z}/_{3\mathbb{Z}} = \{-1, 0, +1\}$. The control of the plant is performed by restricting the controllable input values with respect to the control objectives. These restrictions are obtained by incorporating new algebraic equations into the initial system. This theory sets the basis for the verification and the controller synthesis tool, SIGALI. Moreover, we present a tool developed around the SIGNAL environment allowing the visualization of the synthesized controller by an interactive simulation of the controlled system. In a first stage, the user specifies in SIGNAL both the physical model and the control objectives to be ensured. A second stage is performed by the SIGNAL compiler which translates the initial SIGNAL program into a PDS, and the control 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). Finally, in a third stage, the obtained controller and some simulation processes are automatically included in the initial SIGNAL program. It is then sufficient for the user to compile the resulting SIGNAL program which generates executable code ready for simulation. Different academic examples are used to illustrate the application of the tool.

Keywords: Control theory, Polynomial methods, Synchronous methodology, SIGNAL environment, SIGALI.

1. Introduction & Motivations

The SIGNAL language [3, 16] is dedicated to reliable specifications of real-time reactive systems [2]. In this area, many applications require high reliability and safety. Traditionally, these requirements are checked *a posteriori* using simulation techniques and/or property verification. Control theory of discrete event systems allows the use of constructive methods that ensure, *a priori*, required properties on the system behavior. In this approach, the validation phase is reduced to properties not guaranteed by the programming process. There exist different theories for control of Discrete Event Systems since the 80's [24, 1, 10, 17]. Usually, the starting point of these theories is: given a model for the system and the control objectives, a controller must be derived by various means such that the resulting behavior of the closed-loop system meets the control objectives.

In our case, the specification of the physical model (or the plant) is realized using the synchronous data-flow language SIGNAL. Control theory is then applied on an equational model of the logical part of SIGNAL programs. For this purpose, the boolean part of the program is translated into a polynomial dynamical system over $\mathbb{Z}/3\mathbb{Z}$ (*i.e.*, integers modulo 3: {0,1,-1}) [15]. Furthermore, using such algebraic methods and polynomial dynamical system as a formal model, we are able to synthesize controllers satisfying various kinds of control objectives (they can be expressed as *invariance*, *reachability* and *attractivity* as well as as order relations over the states of the plant). As SIGNAL is a data-flow language, meaning that the system reacts to inputs sent by the environment and produces outputs resulting from internal transformation, it is natural for us to use an input/output approach (see [1] for another I/O approach); however systems defined as finite state automata, as in Ramadge and Wonham, can also be considered within this framework. In our methodology, the physical model is then represented by a polynomial dynamical system while the control of the system is performed by restricting the controllable input values to values suitable for the control goal. This restriction is obtained by incorporating new algebraic equations into the initial system.

Overview of the tool: We now briefly present how the controller synthesis problem has been integrated in the SIGNAL environment. There are two fundamental aspects: the first one deals with the unification of the formalism and the second one deals with the visualization of the result. In order to simplify the use of the tool, the same language is now used to specify the physical model of the system and the control objectives (as well as the verification objectives). Both can now be written in a new extension of the SIGNAL language, named SIGNAL+. With SIGNAL+, it is not necessary for the user to know (or to understand) the mathematical framework that is necessary to perform the computation of the controller. Irrespective of the model used to represent the system and the synthesized controller, some obstacles prevent the diffusion of formal methods for logical controller synthesis. The most important deals with the intrinsic abstraction of the obtained controllers. For instance, in our framework, the controllers are coded by polynomials over $\mathbb{Z}/_{3\mathbb{Z}}$ and consequently by Ternary Decision Diagrams (TDDs), a slight extension of the Binary Decision Diagrams (BDDs) [5]. In other research works, they are sometimes represented by automata [23, 9]. In both cases, they are too complex to be satisfactorily understood. The number of nodes for the first one and the number of states for the second one is too prohibitive.

Thus, in order to efficiently visualize the result (*i.e.*, the behavior of the controlled system (plant and controller)), a new simulation environment of controlled SIGNAL programs, built around the primitive SIGNAL environment, has been developed. This tool allows the visualization of the controller synthesis result by interactive simulation of the controlled system. Figure 1 sums up the different stages necessary to perform such simulations.

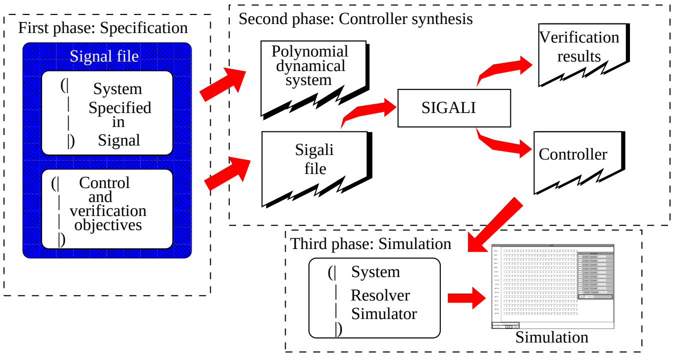


Figure 1. Description of the tool

In the first stage, the user specifies the physical model and the control objectives in SIGNAL+. The second stage is performed by the SIGNAL compiler which translates the initial SIGNAL program into a polynomial dynamical system and the control objectives as well as the verification objectives in terms of polynomial relations and polynomial operations. The controller is then synthesized using the formal calculus tool SIGALI. Finally, in the third stage, the obtained controller is automatically integrated in a new SIGNAL program through an algebraic equation resolver written both in SIGNAL and in C^{++} (some generic processes for simulation can be added at this stage). This new global SIGNAL program can then be compiled. The result is an executable code ready for simulation.

The validation of this prototype has been performed through different academic examples, like the control of a *manufacturing production cell* [13] and the well-known *cat and mouse* example [25], which we now present. This example will be used intensively in the following sections to illustrate the various stages of our controller synthesis techniques and highlight the features of the tool.

The model: a cat and a mouse are placed in a maze shown in Figure 2. The animals can move through doors represented by arrows in this figure.

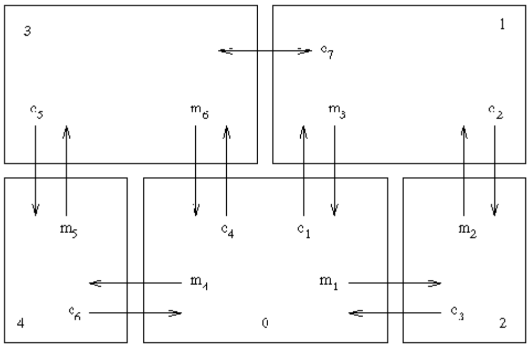


Figure 2. The cat and mouse example.

Doors $C_1, \ldots, C_7$ are exclusively for the cat, whereas doors $M_1, \ldots, M_6$ are exclusively for the mouse. Each doorway can be traversed in only one direction, with the exception of the door C_7. A sensor associated with each door detects the passages of the cat and the mouse through the doors and a control mechanism allows each door to be opened or closed, except for the door C_7 which always stays opened. Initially, the cat and the mouse are in room 2 and 4 respectively.

The requirements: the problem is to control the doors in order to guarantee the two following requirements:

1. The cat and the mouse never occupy the same room simultaneously.

2. It is always possible for the animals to return to their initial positions.

In order to control the system, we assume that the controllable events are door opening and closing requests, whereas the movements of the animals are assumed to be uncontrollable.

The occupation of the rooms and the status of the door, as well as the movements of the animals have been specified in SIGNAL. In order to take into account the requirements (1), (2), with the purpose of obtaining an optimal controller[1], we rely on automatic controller

[1] The controller is optimal in the sense that the controlled system has a smaller behavior than the original one - to fulfill the control objectives - but as large as possible, otherwise, the controlled system "doing nothing" would correspond most of the time (*e.g.*, requirement (1)).

synthesis that is performed on the logical abstraction of the global system.

2. The SIGNAL equational data flow real-time language

In order to specify our model, we use a synchronous approach to reactive real-time systems, and particularly the data flow language SIGNAL [16, 7]. Synchronous languages [8] are derived from theoretical and applied studies on discrete event systems with real time aspects, and on specification methodologies and programming environments for their development [2, 8].

2.1. THE SIGNAL LANGUAGE.

The SIGNAL language [16, 7] manipulates *signals* `X`, which denote unbounded series of typed values, indexed by time. An associated *clock* determines the set of instants at which values are present. 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. Data flow applications are activities executed over a set of instants in time. At each instant, input data are acquired from the execution environment; output values are produced according to the system of equations considered as a network of operations.

The SIGNAL language is defined by a small kernel of operators. Each operator has formally defined semantics and is used to obtain a clock equation and the data dependencies of the participating signals.

- *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.

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 and its semantics, the reader is referred to [16]. The complete programming environment also features a block-diagram oriented graphical user interface [4]) and a proof system for dynamic properties of SIGNAL programs, called SIGALI (see Section 3).

To illustrate the preceding notions, let us consider the specification in SIGNAL of the cat and mouse example.

2.2. Modeling of the cat and mouse system

The complete behavior of the cat and mouse system has been specified in SIGNAL using the graphical interface of the SIGNAL environment. Four processes compose the global system (see Figure 3). The process `State_of_Rooms` describes the occupation of the rooms (*i.e.*, in which room the cat and the mouse are). The process `State_of_Doors` describes the status of the doors (open or closed). The two last processes describe the assumptions that are made on the physical model.

The inputs of the main process `system` are booleans that encode the possible movements of the animals (`Mvt_Mouse_i`, `Mvt_Cat_i`). This inputs will be considered as *uncontrollable* during the synthesis phase. The inputs `DoorState_Mouse_i` and `DoorState_Cat_i` indicate opening and closing requests of the various doors. They will be *controllable*. The outputs of the main process are the booleans (`Mouse_Room_i`, `Cat_Room_i`) and (`Cat_Door_i`, `Mouse_Door_i`) respectively representing the room in which the animals are and the status of the doors.

The `State_of_Rooms` *process*: This process describes the occupation of the rooms for the animals with regard to the mouse and cat movements and the status of the doors (Cf Figure 4 for a part of this process). Consider for example the possible movements of the mouse into or out of room 0. The mouse can come into this room through the door 3

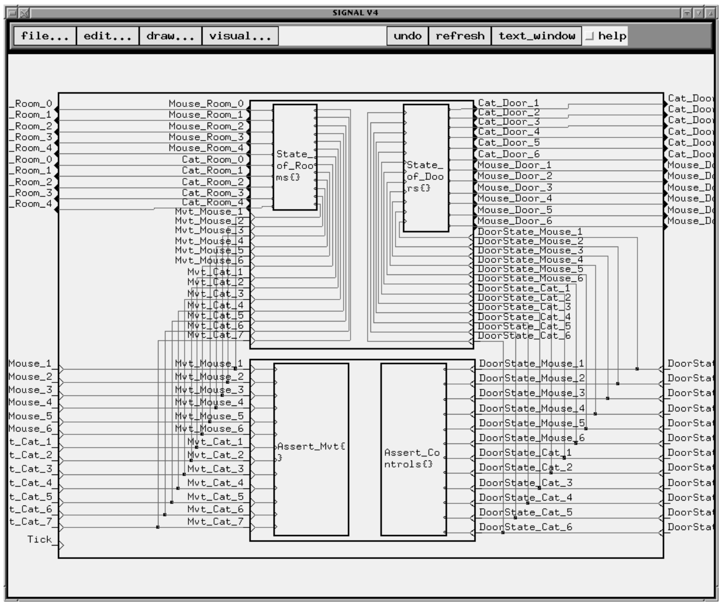


Figure 3. The main process in SIGNAL

(Mvt_Mouse_3) or through the door 6 (Mvt_Mouse_6) or go out through the door 1 (Mvt_Mouse_1) or through the door 4 (Mvt_Mouse_4).

```
(| (| Mouse_Room_0 := (when Mvt_Mouse_3) default (when Mvt_Mouse_6)
                                        default (false when Mvt_Mouse_1)
                                        default (false when Mvt_Mouse_4)
                                        default Z_Mouse_Room_0
    | Z_Mouse_Room_0 :=  Mouse_Room_0 $1
    |)
 | (| Mouse_Room_1 := (when Mvt_Mouse_2) default (false when Mvt_Mouse_3)
                                        default Z_Mouse_Room_1
    | Z_Mouse_Room_1 :=  Mouse_Room_1$1
    |)
 | (| Mouse_Room_2 := .... |)
 | (| Mouse_Room_3 := .... |)
 | (| Mouse_Room_4 := .... |)

 | Mouse_Room_0 ^= Mouse_Room_1 ^= Mouse_Room_2 ^= Mouse_Room_3 ^= Mouse_Room_4
 |)
```

Figure 4. Specification of the states of the rooms (Part of the State_of_Rooms process)

To model the occupation of the room, two supplementary boolean signals are needed: Z_Mouse_Room_0 and Mouse_Room_0. The first one carries the current status of the room 0, where as the second one carries the future status. At a given instant Z_Mouse_Room_0 holds the value *true* if the mouse is in room 0 and is *false* otherwise. Mouse_Room_0 evolves according to the signals Mvt_Mouse_3, Mvt_Mouse_6, Mvt_Mouse_1, and Mvt_Mouse_4, and to the current status of the room: Mouse_Room_0 becomes *true* whenever the mouse enters through the door 3 or 6 (`when Mvt_Mouse_3 default when Mvt_Mouse_6`), *false* when it leaves through the door 1 or 4 (`(false when Mvt_Mouse_3) default (false when Mvt_Mouse_6)`). If none of the sensors signals a passage, the signal `Mouse_Room_0` is equal to the current status `Z_Mouse_Room_0`. The occupation of all the other rooms can be described similarly.

The assumption process: once the physical model specified, describing the global behavior of the system in the language SIGNAL, we have to take into account various assumptions on the physical model. Indeed, when we specify a system, it is not necessary to specify the particular behavior of the inputs. For example, in the cat and mouse problem, we have to take into account the fact that it is not possible for the animals to make two different movements at the same time and that a movement is possible if and only if the animal is in the correct room and if the corresponding door is opened. In order to perform control synthesis as well as verification, a new SIGNAL process has, in general, to be included in the initial SIGNAL program. This process describes the particular behavior of the inputs (i.e., the behavioral restrictions due to the environment).

For example, if the cat is in room 1, two movements are possible: `Mvt_Cat_7` and `Mvt_Cat_2`. Of course, the cat cannot choose both. Moreover the movement `Mvt_Cat_2` is possible if and only if the corresponding door is opened (when the boolean `Z_Cat_Door_2` is *true*). These assumptions are written in SIGNAL as shown in Figure 5.

```
 %  Movement is possible if  the cat is in the room %
(| (| when Mvt_Cat_2 ^= (when Mvt_Cat_2) when Z_Cat_Room_1
    | when Mvt_Cat_7 ^= (when Mvt_Cat_7) when Z_Cat_Room_1
    |)
 %  Movement  is possible if  the door is opened %
 | (| when Mvt_Cat_2 ^= (when Mvt_Cat_2) when Z_Cat_Door_2
    |)
 % the movements of the cat are exclusive %
 | (| (when Mvt_Cat_2) when (when Mvt_Cat_7) ^= not  Tick
    |)
 |)
```

Figure 5. Specification of the mouse movement (Part of the **Assumption** process)

The first line simply says that the signal `Mvt_Cat_2` as to be present only at the instants where the signal `Z_Cat_Door_2` is *present* and *true*, whereas the last line says that the instants where the signals `Mvt_Cat_2` and `Mvt_Cat_7` are *present* at the same time is reduced to the null clock (which means that this never happens).

3. SIGALI: a proof/synthesis environment software

The SIGNAL environment also contains a verification and controller synthesis tool-box, named SIGALI [6]. This tool makes it possible 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: {0,1,-1} = {0,1,2}) as a formal model of program behavior. The theory of PDS uses classical concepts of algebraic geometry, such as ideals, varieties and comorphisms [14]. 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, and not having to enumerate them. The tool SIGALI implements the basic operators: set theoretic operators, fix-point computation, quantifiers [20] (An overview of the SIGALI syntax can be found in [22]). It relies on an implementation of polynomials by Ternary Decision Diagram (TDD) (for three valued logics) in the same spirit of BDD [5], but where the paths in the data structures are labeled by values in $\{-1, 0, 1\}$.

3.1. THE AUTOMATIC CONTROLLER SYNTHESIS METHODOLOGY USING SIGALI

In this section we briefly present the controller synthesis methodology. We first present how the logical part of SIGNAL program can be "extracted" leading to a polynomial dynamical system (PDS). Such a system is the key object on which SIGALI works. From a PDS, we show how it is possible to synthesize a controller according to control objectives. We do not provide the algorithm since the aim of this paper is not to present in details the underlying theory but more to present what we are able to perform using this tool. For a complete review of the theoretical framework of controller synthesis using polynomial methods, the reader is referred to [20].

3.1.1. *The logical abstraction of a* SIGNAL *program*

To model its behavior, a SIGNAL process is translated into a system of polynomial equations over $\mathbb{Z}/_{3\mathbb{Z}}$ [14].

Signals. 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* $\rightarrow 0$). Note that the square of *present* is 1, whatever its value, when it is present. Hence, for a signal `X`, its clock can be coded by x^2. It follows that two synchronous signals `X` and `Y` satisfy the constraint equation: $x^2 = y^2$. This fact is used extensively in the following.

Primitive operators. Each of the primitive processes of SIGNAL can be encoded in a polynomial equation. For example `C := A when B`, which means "*if* $b = 1$ *then* $c = a$ *else* $c = 0$" can be rewritten in $c = a(-b-b^2)$: the solutions of this are the set of behaviors of the primitive process `when`. The delay `$`, which is a dynamic operator deserves some extra explanations. It requires memorizing the past value of the signal into a *state variable*. Translating `B` := `A $1`, requires the introduction of two auxiliary equations: (1) $x' = a + (1 - a^2)x$, where x' denotes the next value of state variable x, expresses the dynamics of the system. (2) $b = a^2x$ delivers the value of the delayed signal according to the memorization in state variable x. Table I shows how all the primitive operators are translated into polynomial equations. For the non boolean expressions, we just translate the synchronization between the signals.

Table I. Translation of the primitive operators.

boolean instructions	
`B := not A`	$b = -a$
`C := A and B`	$c = ab(ab - a - b - 1)$ $a^2 = b^2$
`C := A or B`	$c = ab(1 - a - b - ab)$ $a^2 = b^2$
`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`)`	$x' = a + (1 - a^2)x$ $b = a^2x$ $x_0 = \mathtt{b}_0$
non-boolean instructions	
`B :=` $f(A_1, \ldots, A_n)$	$b^2 = a_1^2 = \cdots = a_n^2$
`C := A default B`	$c^2 = a^2 + b^2 - a^2b^2$
`C := A when B`	$c^2 = a^2(-b - b^2)$
`B := A $1 (init` b_0`)`	$c^2 = a^2$

Remark 1. Note that for a boolean relation/function we have an exact coding of the relation/function as a polynomial function while for a

numerical function/relation, the encoding retains only the synchronisation constraints between the signals involved in this relation/function. Therefore, SIGALI is only able to have reasoning capabilities on the synchronisation and logic properties of SIGNAL programs.

Processes. By composing the equations representing the primitive processes, any SIGNAL specification can be translated into a set of equations called polynomial dynamical system (PDS). Formally, a PDS can be reorganized into three sub-systems of polynomial equations of the form:

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

where X, X' are vectors of variables in $\mathbb{Z}/3\mathbb{Z}$ and $dim(X) = dim(X') = n$. 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 $\mathbb{Z}/3\mathbb{Z}$ and $dim(Y) = m$ called *uncontrollable event variables*, whereas U is a vector of *controllable variables*, with $dim(U) = p$. The first equation called *state transition equation* can be considered as a vector-valued function $[P_1, \ldots, P_n]$ from $(\mathbb{Z}/3\mathbb{Z})^{n+m+p}$ to $(\mathbb{Z}/3\mathbb{Z})^n$. It is composed of all the equations on the state variables, and it captures the dynamical aspect of the system. The second equation is called the *constraint equation* and is a system of equations $[Q_1, \ldots, Q_l]$. It specifies which event may occur in a given state. The last equation gives the initial states.

The trajectories of a controllable system are sequences (x_t, y_t, u_t) in $(\mathbb{Z}/3\mathbb{Z})^n \times (\mathbb{Z}/3\mathbb{Z})^m \times (\mathbb{Z}/3\mathbb{Z})^p$ 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. This particular aspect constitutes one of the main differences with [24]. In our case, the events are partially controllable, whereas in the Ramadge and Wonham formulation, the events are either controllable or uncontrollable. 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 $Q(x_t, y_t, u_t) = 0$.

3.1.2. *Control synthesis Problem.*
Given a PDS S, as defined by (1), a controller is defined by a system of two equations: $C(X,Y,U) = 0$ and $C_0(X) = 0$, where 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 modeled by the system S_c:

$$S_c = \begin{cases} X' = P(X,Y,U) & \\ Q(X,Y,U)=0 & C(X,Y,U)=0 \\ Q_0(X_0)=0 & C_0(X_0)=0 \end{cases} \tag{2}$$

The various control objectives (as well as their definitions) for which we are able to synthesize a controller [20] include:

- **The invariance of a set of states.** A set of states E is *invariant* if every trajectory initialized in E remains in E.
- **The (global) reachability of a set of states.** A set E is *(globally) reachable*, if starting from any possible state, there exists a trajectory that reaches E.
- **The attractivity of a set of states from another set of states.** A set F is *attractive* for a set E if every trajectory initialized in E reaches F.
- 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 (*e.g.*, invariance + reachability).

Finally note that some other control objectives, dealing with *quantitative criteria* can also be considered. In general, these control objectives are expressed as partial order relations. Such relations can, for example, be described by means of numerical cost functions (see [18, 19, 20]).

Remark 2. From a verification point of view, besides the verification of the *invariance*, *reachability* and *attractivity* of a given set of states, it is also possible to symbolically express CTL formulae [6], propositional μcalculus formulae [12, 21] as well as bisimulation equivalences [11, 22].

3.1.3. *Specification of an objective in* SIGNAL+.
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 directly, in the SIGNAL program. With SIGNAL+, it is not necessary for the user to know (or to understand) the mathematical framework which is necessary to perform the computation of the controller. The syntax is given in Table II.

Table II. Basic syntax of SIGNAL+

(|	Sigali(Verif_Objective(Prop))
|	Sigali(Control_Objective(Prop)) |)

The keyword **Sigali** means that the subexpression must be evaluated by SIGALI. In the first case, the function **Verif_Objective** (it could be **invariance**(), **reachability**(), **attractivity**(),etc) means that SIGALI has to check the verification objectives according to the boolean **PROP**, which defines a set of states in the corresponding polynomial dynamical system. In the second case, the function **Control_Objective** means that SIGALI has to compute a controller which will ensure the control objective for the controlled system. It could be one of the control objectives presented in the previous Section, (i.e.(**S_Security()**, **S_Reachability()**, **S_Attractivity()**, etc.). We also proscribe in the SIGNAL program the status of the inputs (controllable or not)) by the function **Controllable()**. The complete SIGNAL program is obtained composing the process specifying the plant and the one specifying both the verification objectives and the control objectives in parallel. The compiler produces a file which contains the polynomial dynamical system resulting from the abstraction of the complete SIGNAL program and the algebraic control (or verification) objectives. This file is then interpreted by SIGALI.

3.2. APPLICATION TO THE CAT AND MOUSE EXAMPLE

To illustrate Section 3.1, let us come back to the cat and mouse example. Using the language SIGNAL+, the control objectives we want to ensure for the cat and mouse problem can be described as follows (cf Figure 6):

We first introduce the signals **cat_mouse_room_i**, (i=0,...,4) which are *true* when the cat and the mouse are both in room i and *absent* otherwise. We then introduce the boolean **error** which is *true* when one of the events **cat_mouse_room_i** is *true* and it is *false* otherwise (in terms of automata, we describe the set of states where objective 1 is violated). To specify the second objective, we introduce the boolean `initial_States` that is *true* whenever both the cat

```
(| (| system() |) % the physical model specified in Signal
 | (| Cat_Mouse_Room_0 := when (Z_Cat_Room_0 and Z_Mouse_Room_0)
    | Cat_Mouse_Room_1 := when (Z_Cat_Room_1 and Z_Mouse_Room_1)
    | Cat_Mouse_Room_2 := when (Z_Cat_Room_2 and Z_Mouse_Room_2)
    | Cat_Mouse_Room_3 := when (Z_Cat_Room_3 and Z_Mouse_Room_3)
    | Cat_Mouse_Room_4 := when (Z_Cat_Room_4 and Z_Mouse_Room_4)
    |)
 | (| Error:= Cat_Mouse_Room_0 default Cat_Mouse_Room_1 default Cat_Mouse_Room_2
             default Cat_Mouse_Room_3 default Cat_Mouse_Room_4 default false
    |   Error ^= Tick
    |)
 | (|   Initial_States:= Z_Cat_Room_2 and Z_Mouse_Room_4 |)
 | (|   Sigali(S_Security(False(Error)))
    |   Sigali(S_Reachability(True(Initial_States)))
    |)
 |)
```

Figure 6. Specification of the control objectives

and the mouse are in their initial room and *false* otherwise. Finally, to ensure the two objectives, we require SIGALI to compute a controller which ensures (i) the invariance of the set of states where the boolean `error` is false (objective 1: `S_Security(False(Error))`) and (ii) the reachability of the cat and mouse initial positions (objective 2: `S_Reachable(True(Initial_states))`).

The corresponding SIGALI file `system_CMD.z3z` (Table III), obtained after the compilation of the global SIGNAL program (such a computation is automatic and performed, in this case, in less than 5Sec), is the following:

Table III. The resulting SIGALI file

read("system.z3z"); ⇒ *loading of the PDS S*
read("Synthesis_Library.z3z");
read("Verification_Library.z3z"); ⇒ *Loading of the necessary libraries*
Set_States_1: False(Error); ⇒ *Compute the states where Error is false*
Set_States_2: True(Initial_States);
⇒ *Compute the states where Initial_States is true*
S_c_1: S_Invariance(S,Set_States_1);
⇒ *Controller ensuring the invariance of Set_States_1*
S_c: S_Reachable(S_c_1,Set_States_2);
⇒ *Controller ensuring the reachability of Set_States_2*

The file "system.z3z" codes the polynomial dynamical system S that represents the initial system. S is represented by 23 state variables, 12 controllable variables and 13 uncontrollable variables. The second and third files ("Verification_library.z3z" and 'Synthesis_library.z3z") are libraries in which we can find the different algorithms concerning verification and controller synthesis problems. The boolean `Error` becomes a polynomial expressed by state variables and events as well as the boolean `Initial_States`. The polynomials `Set_States_1` and

`Set_States_1` respectively represents the set of states where the polynomial `Error` is equal to -1 (i.e. in the SIGNAL world, where the boolean `Error` is *false*) and the set of states where the boolean `Initial_States` is equal to 1 (therefore *true*). The last two lines correspond to the controller computations.

This file is then interpreted by SIGALI (`read(''system_CMD.z3z'')`) which computes the controller with respect to the control objectives. In this case, SIGALI first computes a controller which ensures the the invariance of the set of states where the polynomial `Set_States_1` takes the value 0. The obtained controlled system is called `S_c_1`. From this new system, SIGALI computes the controller that ensures the reachability of the initial states. The result is given by a new controlled system `S_c` from which the global controller ensuring the two control objectives is extracted. The global controller (in fact a TDD) is computed by SIGALI in about 15Sec. This TDD is then saved in a file, which could be used to perform simulation following the principle described in the next section.

4. The simulation of the results

As explain in the introduction, one of the main problems dealing with the controller synthesis methodology is the visualization of the new behavior of the controlled system. What we propose, in our tool, is an easy way to obtain a simulation which allows the visualization of the controller synthesis result by interactive simulation of the controlled system. This way, the user can better understand the controller synthesis effects on the system. We here suppose that a controller has already be synthesized according to given control objectives. The first stage consists in integrating automatically the controller (more precisely, a resolver process) in the initial SIGNAL program and the second stage consists in generating a simulation of this new SIGNAL program, following the architecture of Figure 7.

4.1. INTEGRATION OF THE RESOLVER IN A SIGNAL PROGRAM & SIMULATOR BUILDING

The resolver: In our framework, a controller is a polynomial, and thus is represented by a TDD. In most cases, the result is not deterministic; several values are possible for each command, when the system evolves into a state. Therefore, an algebraic equation resolver has been developed in SIGNAL for the control part of the `resolver` process and in C^{++} for the Algebraic Equation Resolver (AER).

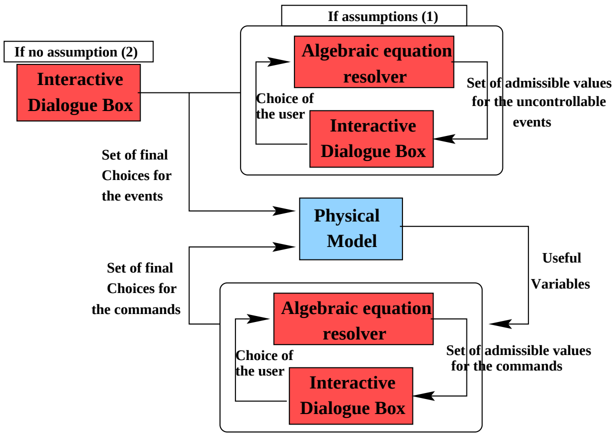


Figure 7. Integration of the resolver and simulation processes : The principle

The AER is able to solve polynomial equations (*i.e.*, controllers in our case) according to the internal state values (the values of the state variables) and the input event values. The constraint part of the controller is given by the polynomial : $C(X,Y,U) = 0$, where X,Y and U are sets of variables. The AER provides, for given values x, y, all the possible values for the commands u; it is important to notice that not only one but all the alternatives of commands are proposed. The AER is automatically integrated in the initial SIGNAL program, following the diagram of Figure 7. The AER, which can be considered as an external function, is encapsulated in a signal process, named `resolver`. The links (*i.e.*, the connections through signals) between the process `resolver`, the AER and the process which specifies the system are automatically added in order to obtain the new SIGNAL program.

Following the same principle, if some assumptions have been expressed on the uncontrollable events, a sub-controller is also integrated in the SIGNAL program. In this case, the sub-controller is a polynomial in variables X and Y given by $Q'(x,y) = 0 \Leftrightarrow \exists u, Q(x,y,u) = 0$, where Q is the constraint equation of the polynomial dynamical system (i.e., the only admissible uncontrollable events are the ones for which there exists at least one command u such that $Q(x,y,u) = 0$). Note that when Q' is not reduced to the null polynomial, another process resolver is included (and that even if no assumption process exists). The fact that Q' be not reduced to the null polynomial simply means that there eventually exist relations (constraints) between the uncontrollable inputs.

Remark 3. The complexity of the algorithm that provides the possible values of the controllable events is very low. Indeed, once the values of the x and y have been sent to the AER, the controller is reduced to an equation $C'(U_1, ..., U_p) = 0$. Hence, in order to know whether U_i could take a value $j \in 0, 1, -1$, the AER just has to test if the TDD $C'(U_1, .U_{i-1}, j, U_{i+1}, .., U_p)$ reduces to the constant polynomial 1. If not, j is a possible value for the controllable event variable U_i.

The simulator: at the same time, the user has the option of adding in this new program some generic processes of simulation. These SIGNAL processes perform, after compilation, the automatic construction of graphical input acquisition buttons (interactive dialogue box in Figure 7) and output display windows for the signals of the interface of the program, in an oscilloscope-like fashion; with regard to the commands and the uncontrollable events, the graphical acquisition button processes are automatically added in the SIGNAL program when the resolver(s) is (are) included. From this point, it is sufficient for the user to compile the resulting SIGNAL program which generates executable code ready for simulation.We are also able to perform real graphical animation in order to simulate the behavior of the system (see Section 4.2).

Simulation principle: once the controller has been computed and integrated in the new SIGNAL program as explained in the previous section, we have to simulate the result of the synthesis. In most cases, the controller is not deterministic, in the sense that for a given state and a given uncontrollable event, more than one command can be admissible. To solve this problem, we choose to perform a step by step simulation.

During the first stage of SIGNAL program specification, the user indicated the controllable and uncontrollable inputs. Thus, the values of the controllable events will be chosen by the user under the control of the resolver through an interactive dialogue box (cf. Figure 8). For example, when the user makes a choice (i.e., the user chooses a particular value for one of the commands), this choice is automatically sent to the AER, which returns the set of possible values for the remaining commands. In fact, each time a new choice is made by the user, a new controller is computed, in the sense that one variable of the polynomial controller has been instantiated. New constraints can then appear on the commands which are not totally specified (there still exist more than one choice); During this exchange between the dialogue box and the resolver, some commands can be totally specified by the resolver in which case their values are then imposed.

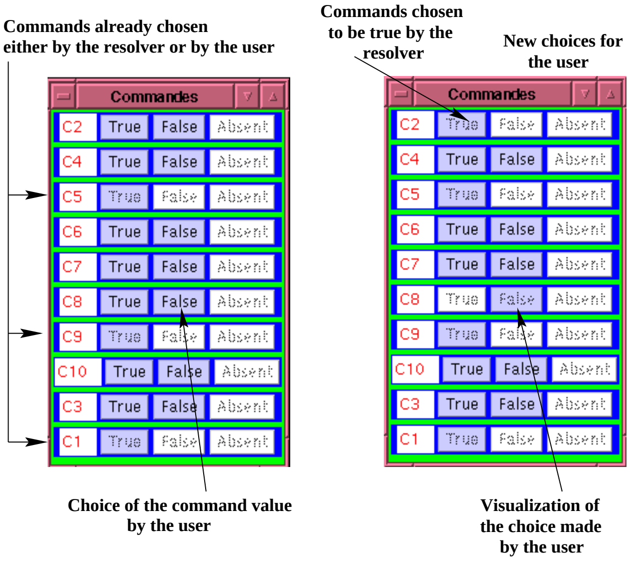


Figure 8. Example of simulation during a step.

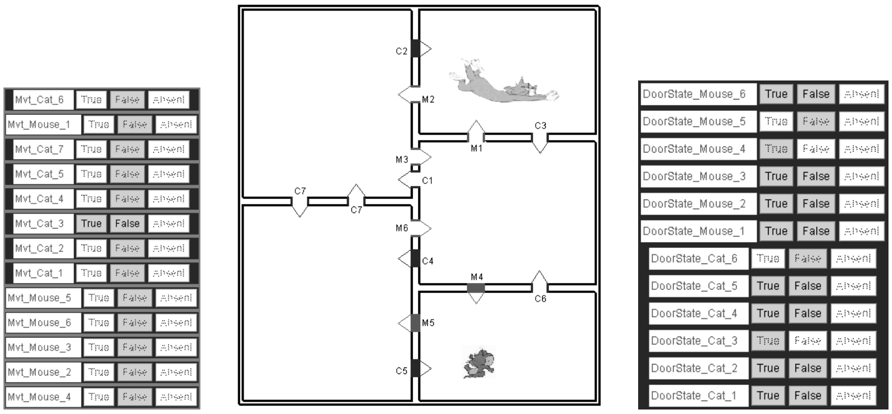
The choice of the command values can be performed step by step by the user, or using a random process for a step of simulation. In the second case, the resolver chooses the command values. The user can also ask for a random simulation during an indeterminate number of simulation steps.

Remark 4. The choice of the uncontrollable event variable values follows the same principle. The value of these inputs will be given by the users (without any restrictions in the case where $Q' \equiv 0$) or using a random process according to the admissibility of these events ((1) in Figure 7). In the case where the uncontrollable events are constrained, then the possible values of these inputs have to be accepted by another resolver, which gives, through a dialogue box, the possible choices for these uncontrollable inputs ((2) in Figure 7).

4.2. Simulation of the Cat and mouse Example

Once the controller corresponding to the two control objectives has been synthesized by Sigali, it is integrated in the Signal environment as explained in Section 4.1. After the compilation of this new Signal program, a graphical simulation is obtained (see Figure 9).

Figure 9(a) represents the uncontrollable events (*i.e.*, the cat and mouse movements). They are constrained by a resolver resulting from the process **assumptions** (never two different movements at the same



(a) The events (b) The simulator interface (c) The commands

Figure 9. Cat and Mouse Problem Simulation

time and a movement is admissible if and only if the door is open and the animal is in the correct room.). Figure 9(c) represents the commands (*i.e.*, the opening and closing requests). The choice of the user is limited by the main resolver in order to ensure the two objectives. Finally, Figure 9(b) represents the graphical interface of simulation, in which the positions of the animals, as well as the states of the door can be observed.

Remark 5. This graphical simulation has not been produced using the generic processes of simulation but has been implemented in Java. To this end, we developed a "generic" interface making the link between the C file produced by the SIGNAL compiler and a Java graphical interface. The control panels are still automatically generated (Figure 9(a) and 9(c) as well as a pace-maker panel (not presented here)). The only part which is non generic is the graphical simulation of Figure 9(b) which concerns the animation of the movements, but this can be done easily.

5. Another example: The control of a manufacturing cell

5.1. Brief description of the problem

In this example, we consider a flexible manufacturing cell, as shown in Figure 10. This manufacturing cell is composed of five workstations (three processing workstations, a part-receiving station (Work Station 1 in Figure 10) and one completed parts station (Work Station 4 in Figure 10)).

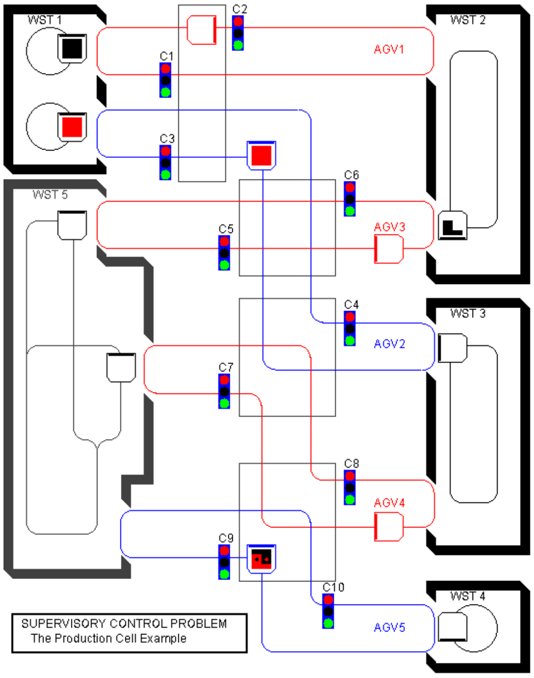


Figure 10. The flexible manufacturing cell

Five Automated Guided Vehicles (AGV's) transport materials between pairs of stations, passing through conflict zones shared with other AGV's. We assume that the controller receives signals from the AGV's indicating their current positions in the manufacturing cell. We also assume that we can stop the AGV's before they enter in some conflict zones (C_i transitions in Figure 10).

The control synthesis problem is to coordinate the movement of the various AGV's in order to avoid collisions in the conflict zones (*i.e.*, it is required that all AGV's be controlled so that each zone be occupied by no more than one AGV).

5.2. Specification in Signal:

The complete behavior of the system is specified in Signal. It is decomposed into 10 subsystems, respectively coding the 5 workstations, and the five AGV's circuits (processes **Work_Station_i** and **Agv_i**). The movement in each subsystem is driven by a clock, possibly different for each subsystem, named **Time_Wst_i** for the workstations, and **Time_Agv_i** for the AGV's. Synchronizations between the different subsystems are performed through exchanged messages, coding the state of each sub-system (in dash in Figure 11).

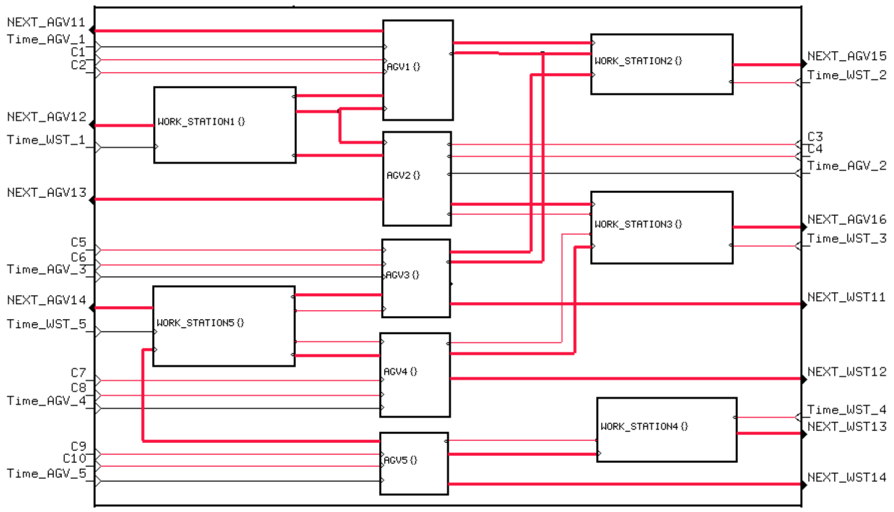


Figure 11. Specification in SIGNAL of the synchronization between the sub-systems

The control objective is specified by another process. Figure 12 describes this specification:

```
(| (| Zone_1:= (true when (when (NEXT_AGV16 or NEXT_AGV13))
                                    when (NEXT_AGV23 or NEXT_AGV212))
                default (true when (NEXT_AGV16 and NEXT_AGV13)) default false
   | Zone_2 := (true when (when (NEXT_AGV25 or NEXT_AGV210))
                                    when (NEXT_AGV33 or NEXT_AGV36))
              default (true when (NEXT_AGV33 and NEXT_AGV36)) default false
   | Zone_3 := .....
   | Zone_4 := .....    |)
 | Tick := Time_AGV_1 default Time_AGV_2 default Time_AGV_3 default Time_AGV_4 default
           Time_AGV_5 default Time_WST_1 default Time_WST_2 default Time_WST_3 default
           Time_WST_4 default Time_WST_5
 | Zone_1 ^= Zone_2 ^= Zone_3 ^= Zone_4 ^= Tick
 | Conflit := Zone_1 or Zone_2 or Zone_3 or Zone_4
 | SIGALI(Controlable(C1,C2,C3,C4,C5,C6,C7,C8,C9,C10)
 | SIGALI(S_Security(False(Conflit)))
 |)
```

Figure 12. The control Objective

In order to realize the control objective, we first have to define the states of the system where two AGV's are at the same time in a common zone. For example, the signal `Zone_1` is a boolean which is *true* when the AGV_1 and the AGV_2 are both in the conflict zone 1. In this expression, the boolean `Next_Agv_i_j` represents the j^{th} position of the i^{th} AGV. `Zone_1` is synchronized with the event `Tick` which is equal to the union of the different clocks for the sub-systems (`Time_...`). Each conflict zone is specified in SIGNAL in this manner. Finally, The boolean `Conflit` is *true* when one of the booleans `Zone_i` is *true*, it is *false* otherwise. It corresponds to the forbidden states (*i.e.*, the states where two AGV's share a conflict zone).

Now that the property has been specified, we add in the SIGNAL program the control objective. We first indicate which event is controllable `SIGALI(Controllable(Ci))`. Finally, to ensure the control objective, we require SIGALI to compute a controller that ensures the invariance of the set of states where the boolean `Conflit` is *false*: `S_Security(False(Conflit))`.

5.3. CONTROLLER SYNTHESIS AND SIMULATION OF THE RESULTS:

The global system (the model process, the control objectives process) is automatically translated by the compiler in a polynomial dynamical system. It is represented by 56 states variables and 10 controllable event variables encoding the `C_i` and 10 uncontrollable events encoding the clock of each sub-system. It can be shown that the corresponding explicit automaton has more than 10^6 reachable states. The controller is then synthesized by SIGALI. Its computation is realized in less than 10Sec by SIGALI. The result is a TDD, which is integrated into the initial SIGNAL program. After the compilation of this new SIGNAL program, a graphical simulation is obtained (see Figure 8 and 10). Figure 8 represents the commands (*i.e.*, the opening and closing requests). The choice of the user is limited by the main resolver in order to ensure the objective. Finally, Figure 10 represents the graphical interface of simulation, in which the positions of the AGV's can be observed.

We also have performed the synthesis of controllers with a more intricate manufacturing cell (in fact with two AGV's in the first, third and fifth transport zone). Note that the corresponding automaton is now composed of more than $2 * 10^8$ different reachable states. For the same control objective, the controller is computed in less than 2 hours (Compared to the previous case, the high computation time is basically due to the state space explosion).

Remark 6. For simulation purposes (simplification), the clocks of the various sub-system have been assumed to be equal. Therefore, during the simulation all the events are controllable. However, the controller synthesis phase has been performed with different clocks for the sub-systems (they were supposed to be uncontrollable).

6. Conclusion

In this paper, we have presented the integration of a controller synthesis methodology in the SIGNAL environment through the description of a tool dedicated to the algebraic computation of a controller and then to the simulation of the controlled system.

The specification of the system is done in a discrete event framework using the language SIGNAL. In order to facilitate this step, the user can use a block-diagram graphical user interface. 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. The formal verification of a SIGNAL program, as well as the automatic controller design are performed using a formal calculus system named SIGALI.

The principle is to translate the logical part of SIGNAL program into a polynomial dynamical system over $\mathbb{Z}/_{3\mathbb{Z}}$, an implicit representation of an automaton. The operations on equation systems belonging to the theory of algebraic geometry enable the treatment of various kinds of properties. The same operations can also be used for the automated synthesis of controllers where algebraic methods are used this time for the derivation, from a model of a system, of a controller satisfying given properties and objectives. Finally, in order to facilitate the use of the controller synthesis methodology, we have added in the SIGNAL language the possibility of directly expressing the control objectives (and the verification objectives) in the initial SIGNAL program. Therefore, it is not necessary for the user to know (or to understand) the mathematical framework which is necessary to perform the computation of the controller. Moreover, as the result is an equation encoded by a TDD, we have developed a simulator in the SIGNAL environment which allows the user to visualize the new behavior of the controlled system.

Acknowledgements

This work was partially supported by lectricit de France (EDF) under contract number M64/7C8321/E5/11 and by the Esprit SYRF project 22703. The authors gratefully acknowledge relevant and insightful comments from the anonymous reviewers of this paper.

References

1. Balemi, S., G. Hoffmann, H. Wong-Toi, and G. Franklin: 1993, 'Supervisory Control of a Rapid Thermal Multiprocessor'. *IEEE Transactions on Automatic Control* **38**(7), 1040–1059.
2. Benveniste, A. and G. Berry: 1991, 'Real-time systems designs and programming'. *Proceedings of the IEEE* **79**(9), 1270–1282.
3. Benveniste, A. and P. Le Guernic: 1990, 'Hybrid Dynamical Systems and the Signal Programming Language'. *IEEE Trans. Automat. Control* **35**, 535–546.

4. Bournai, P. and P. Le Guernic: 1993, 'Un environnement graphique pour le langage Signal'. Technical Report 741, IRISA (In French).
5. Bryant, R.: 1986, 'Graph-Based Algorithms for Boolean Function Manipulations'. *IEEE Transaction on Computers* **C-45**(8), 677–691.
6. Dutertre, B.: 1992, 'Spécification et preuve de systèmes dynamiques'. Ph.D. thesis, Université de Rennes I, IFSIC (In French).
7. Gautier, T. and P. Le Guernic: 1999, 'Code generation in the SACRES project'. In: *Towards System Safety, Proceedings of the Safety-critical Systems Symposium, SSS'99*. Huntingdon, UK.
8. Halbwachs, N.: 1993, *Synchronous programming of reactive systems*. Kluwer.
9. Hoffmann, G. and H. Wong-Toi: 1992, 'Symbolic synthesis of supervisory controllers'. In: *Proc. of 1992 American Control Conference, Chicago,Il,USA*. pp. 2789–2793.
10. Holloway, L., B. Krogh, and A. Giua: 1997, 'A survey of Petri Net Methods for controlled Discrete Event Systems'. *Discrete Event Dynamic Systems: Theory and Application* **7**, 151–190.
11. Kouchnarenko, O. and S. Pinchinat: 1998, 'Intensional approachs for symbolic methods'. *Electronic Notes in TCS* **18**.
12. Kozen, D.: 1983, 'Results on the Propositional μ-calculus'. *Theoretical Computer Science* **27**(3), 333–354.
13. Krogh, B. H.: 1993, 'Supervisory Control of Petri Nets'. In: *Belgian-French-Netherlands' Summer School on Discrete Event Systems*.
14. Le Borgne, M., A. Benveniste, and P. Le Guernic: 1991, 'Polynomial dynamical systems over finite fields'. In: *Algebraic Computing in Control*, Vol. 165. pp. 212–222.
15. Le Borgne, M., H. Marchand, E. Rutten, and M. Samaan: 1996, 'Formal Verification of SIGNAL programs: Application to a Power Transformer Station Controller'. In: *Proceedings of AMAST'96*, Vol. 1101 of *Lecture Notes in Computer Science*. Munich, Germany, pp. 271–285.
16. Le Guernic, P., T. Gautier, M. Le Borgne, and C. Le Maire: 1991, 'Programming Real-Time Applications with Signal'. *Proceedings of the IEEE* **79**(9), 1321–1336.
17. Maler, O., A. Pnueli and J. Sifakis: 1995, 'On the Synthesis of Discrete Controllers for timed Systems'. *Proceedings STACS'95*, , Vol. 900 of *Lecture Notes in Computer Science*, pp. 229–242.
18. Marchand, H. and M. Le Borgne: 1998a, 'On the Optimal Control of Polynomial Dynamical Systems over Z/pZ'. In: *4th International Workshop on Discrete Event Systems*. Cagliari, Italy, pp. 385–390.
19. Marchand, H. and M. Le Borgne: 1998b, 'Partial Order Control of Discrete Event Systems modeled as Polynomial Dynamical Systems'. In: *1998 IEEE International Conference On Control Applications*. Trieste, Italia.
20. Marchand, H. and M. Le Borgne: 1999, 'The Supervisory Control Problem of Discrete Event Systems using polynomial Methods'. Research Report 1271, Irisa.
21. Pinchinat, S.: 1996, 'Sigali *vs* μcalculus'. Personnal communication.
22. Pinchinat, S., H. Marchand, and M. Le Borgne: 1999, 'Symbolic Abstractions of Automata and their application to the Supervisory Control Problem'. Research Report 1279, IRISA.
23. Ramadge, P. J. and W. M. Wonham: 1987, 'Modular Feedback Logic for Discrete Event Systems'. *SIAM J. Control Optim.* **25**(5), 1202–1218.

24. Ramadge, P. J. and W. M. Wonham: 1989, 'The Control of Discrete Event Systems'. *Proceedings of the IEEE; Special issue on Dynamics of Discrete Event Systems* **77**(1), 81–98.
25. Wonham, W. M. and P. J. Ramadge: 1987, 'On the Supremal Controllable Sublanguage of a Given Language'. *SIAM J. Control Optim.* **25**(3), 637–659.