Hybrid Dynamical Systems Theory and the SIGNAL Language

ALBERT BENVENISTE, SENIOR MEMBER, IEEE, AND PAUL LE GUERNIC

Abstract—**We study the logic and synchronization characteristics of general dynamical systems called Hybrid Dynamical Systems. Our theory is related to discrete event dynamical system theory, but handles numerics as well as symbolics. It is supported by the programming language SIGNAL and a mathematical model of general implicit dynamical systems. The core of the theory is the notion of HDS resolution which is based on a coding of any HDS into a "dynamical graph" which consists of a skew product of a polynomial dynamical system on the finite field of integers modulo 3 (to describe the transitions of the underlying automaton) and directed graphs (to describe how data dependencies dynamically evolve). The resolution algorithms are then based on the study of this dynamical system.**

I. Introduction

A. Requirements from Applications: Hybrid Dynamical Systems (HDS)

As recognized in [21] and [15], most modern applications involving dynamical systems are very complex in nature; think of the following:

- real-time complex control or signal processing systems in avionics, aeronautics, and in C^3-military systems;
- automation handling man–machine interfaces of control systems (monitoring, trouble shooting, visual aids in avionics, remote manipulation. . .);
- vision- and sensory-based control in robotics;
- complex pattern recognition applications such as continuous speech recognition

to mention just a few. Some particular features of these applications are the following:

1) the mixed continuous/discontinuous nature of time because of the simultaneous presence of familiar differential/difference dynamical subsystems and discrete event systems relating these subsystems;

2) the presence of dynamics;

3) a large combinatorial complexity as far as logic and synchronization is concerned (there is no concise model to describe precisely such applications), hence the need for modularity.

Discrete event dynamical systems (DEDS) have been introduced as a theoretical framework for the study of flexible manufacturing and related systems by Wonham and Ramadge [28], [29], and have been widely studied since their introduction [25], [24], [17]. Roughly speaking, DEDS are finite state transition systems which are observed and can be controlled by the language generated by the labels that are attached to each transition, regardless of the precise meaning of these labels. However, in most of the above-mentioned applications, the mutual interaction between numerical computations and the enabling or disabling of transitions is very complex and should be considered within a theory of such dynamical systems: this point will be discussed several times throughout this paper.

Manuscript received April 18, 1988; revised April 13, 1989. Paper recommended by Associate Editor, A. Haurie.

The authors are with IRISA-INRIA, Campus de Beaulieu, Rennes Cedex, France.

IEEE Log Number 9034487.

On the other hand, past research in computer science has resulted in the development of a large set of tools to handle such complex dynamical systems in the context of real-time systems and languages. More precisely *communicating systems* theories were developed, with CSP *(communicating sequential processes)* [16], [9] and CCS *(a calculus of communicating systems)* [22], [23] as most famous examples. More recently, the new approach of *synchronous programming* has been introduced and developed around the languages ESTEREL [8], [12], LUSTRE [7], [10], and SIGNAL [20], [5], [4], [19] to specify, program, and analyze the kind of complex dynamical system we described above. This is the direction we want to pursue and further discuss from a control viewpoint in this paper.

In the sequel, *hybrid dynamical systems (HDS)* theory will refer to a theory handling synchronization, logic, and their interconnections with the numerical behavior of dynamical systems. As the reader will understand while reading this paper, the mixed nature of HDS justifies the development of a new theory and paradigm.

B. A New Paradigm, Some Fruitful Remarks

1) Building complex objects requires the use of strongly modular languages. Our first remark was about the highly combinatorial complexity of HDS. Such a complexity faces us with a new problem which was not considered before in the control community, namely the difficulty of simply *describing, specifying,* or *constructing* HDS. Such a formal specification must be based on the use of an unambiguous *syntax* (which can be text-, mathematics-, or even graphics-oriented), that is to say a specification or programming *language*.

2) Why relational dynamical systems? A second claim is that a HDS should be described via a set of *relations* or *constraints,* rather than as a complicated input–output map as it is usually done in control science. This issue has also been brought up by J. C. Willems [30] in the context of the theory of linear dynamical systems. The advantage is that both dynamics and objectives of the system can be stated within a single framework, unlike to classical control theory where they are separated into the dynamics and some performance criterion.

3) The basic problem: HDS resolution. An immediate consequence of this relational framework is that such HDS specifications cannot be effective, i.e., it is not immediately possible to compute the outputs of a so-specified HDS in response to some sequences of inputs. The control scientist will recognize a standard situation when handling descriptor or implicit linear dynamical systems. By HDS *resolution,* we have in mind a procedure to transform any relational HDS specification into a machine which can execute the desired behaviors, and thus represents the desired equivalent input–output map.

4) What is the nature of time for HDS? Complex applications such as mentioned above are inherently distributed in nature. Hence, every subsystem possesses its own time reference, namely the ordered collection of all the communications or actions this subsystem performs: in sensory based control systems, each sensor possesses its own digital processing with proper sampling rate, actuators generally have a slower sampling rate than sensors, and moreover the software devoted to monitoring only reacts to various kinds of alarms that are triggered internally or

0018-9286/90/0500-0535$01.00

externally. Hence, the nature of time in HDS is by no means universal, but rather local to each subsystem, and consequently multiform. A fundamental consequence is that communications between subsystems impose relative constraints on the timing of these subsystems: an alarm can be sent by an actuator or a sensor to the supervisor which in turn is designed to react on actuators: the whole result is a synchronization constraint between these subsystems. Hence, handling these multiform time references and reasoning about them is one of the fundamental tasks we have to perform.

C. Organization of the Paper

Section II is devoted to an introduction to the topic of HDS. For this purpose, we begin with a point of view which we think to be as natural as possible to the control audience, and show step-by-step how the SIGNAL language naturally comes out from this discussion.[1] A mathematical model corresponding to this informal presentation is given in Appendix A. In Section III, HDS resolution is informally presented and discussed. It is shown how the previously mentioned mutual interaction between synchronization and computation is handled via *dynamical graphs,* a notion which combines both advantages of signal flow graphs and automata. An algebraic coding of dynamical graphs using polynomial dynamical systems over the finite field of integers modulo 3 is introduced, and its use is illustrated on the sketchy analysis of properties such as observability and deadlock. Finally, HDS resolution is informally presented on an example. Formal models and methods supporting these examples are found in [5] and [3].

II. The SIGNAL Programming Language; Some Examples

A. Connecting the Topic to a Control Formulation

The purpose of this section is to motivate the notion of hybrid dynamical system we shall consider throughout this paper. A simple mathematical model is presented in Appendix A.I, where the notions and objects we shall informally introduce in the sequel are formally defined.

1) Specifying Hybrid Dynamical Systems: Consider a discrete-time dynamical system described by a set of dynamical plus algebraic equations

$$\xi_{n+1} = f(\xi_n, y_n)$$
$$0 = g(\xi_n, y_n) \qquad (1)$$

where the variables ξ_n and y_n are both vector valued and $n = 1, 2, 3 \cdots$. We can define some of the components in y_n to be input variables, and some as output variables and investigate the resulting input–output behavior of this system. Clearly, depending on the peculiarity of the functions f and g, at a given instant, the output may not exist for a given input and state, or multiple solutions may exist. In this sense, this is a *relational* dynamical system.

Now, assume that all the vectors above have a mixture of components such as real, integer, logic, symbolic,.... Therefore, f and g may look unusual. In this sense, this is a *hybrid* dynamical system, and we shall see later what the consequences of this property are when handling such systems. All this is quite familiar and not really novel in a control formulation.

What is new is a certain kind of restricted asynchronism. This is explained next. Assume that each variable, in addition to the normal values it takes in its range, can also take a special value representing the *absence* of data at that instance. The symbol used for absence is $\perp$. Therefore, an infinite time sequence of an integer variable (we shall refer to as a *signal* in the sequel) may look like

$$1, -4, \perp, \perp, 4, 2, \perp, \cdots \qquad (2)$$

which is interpreted as the signal being absent at the instants $n = 3, 4, 7, \cdots$, etc. The following questions are immediate from this definition.

(1) If a single signal is observed, should we distinguish the following samples from each other?

$$1, -4, \perp, \perp, 4, 2, \perp, \cdots$$
$$\perp, 1, \perp, -4, \perp, 4, \perp, 2, \perp, \cdots$$
$$1, -4, 4, 2, \cdots.$$

Consider an "observer"[2] who monitors this single signal and does nothing else. Since he is assumed to observe only *present* values, there is no reason to distinguish the samples above. In fact, the symbol $\perp$ is simply a tool to specify the *relative* presence or absence of a signal, given an *environment,* i.e., other signals that are also observed (cf. Appendix A.I). Jointly observed signals taking the value $\perp$ simultaneously for any environment will be said to *possess the same clock,* and they will be said to possess different clocks otherwise (cf. Appendix A.I). Hence, clocks can be considered as equivalence classes of signals that are present simultaneously. As a first consequence, we prefer to omit the time index n when referring to signals since clocks are only *relative* rather than absolute notions. In the sequel, signals will be denoted by names such as x, y, zap, $\cdots$, without mentioning the time index.

(2) How to extend usual functions to the special value $\perp$? For instance, what does $2 + \perp$ mean? Several choices are possible. For reasons that will be apparent later, we decided that $2 + \perp$ has *no meaning,* so that $+$ appears as a *partial function* giving, for instance, $2 + 3 = 5$, $\perp + \perp = \perp$, but being undefined when only one of its arguments takes the special value $\perp$. With this convention, usual functions are easily extended to signals possessing the same clock. For instance, we shall write equations such as

```
water := rain − evaporation
help := (water > max_level)
```

The first equation requires that both signals rain and evaporation possess the same clock, and this equation delivers the difference of these inputs when they are present. The second one delivers the boolean help exactly when the level of water is measured, with the value *true* or *false* according to whether the mentioned condition is satisfied or not (max_level is assumed to be a constant threshold and is therefore permanently available). Consequently, while the usual $+$ is certainly a function, our $+$ on signals is *not,* since requiring the clocks of the inputs to be equal is a *constraint* on the input signals. To help for the present discussion, we shall for the moment use the (naive) notation

$$clock(\text{rain}) = clock(\text{evaporation}) = clock(\text{water}) = clock(\text{help})$$

to refer to the equality of clocks.

(3) How to produce signals with new clocks from existing signals? For example, how can we perform undersampling, oversampling, both at data-dependent rates? The simplest idea is to select the instants at which some boolean signal takes the value *true*. For instance, we shall write formulas such as

```
exceptional_funds := (500 flooded_acre) when help
```

to mean that the exceptional_funds must be delivered to the farmers precisely when help occurs and takes the value true (cf. above) and, then, are (in billion $) 500 times the flooded acreage. To simplify, we assume that flooded_acre is measured exactly when help occurs and is true. A new clock was created by this way, which is less frequent than the clock of the boolean signal help.

[1] This discussion is related to social science, but we could have selected an example from control science as well.

[2] In the common sense, no mathematical definition is referred to here.

help. We shall write (cf. Appendix A.I for a formal definition of $\preceq$)

$$clock(\text{exceptional\_funds}) \preceq clock(\text{help})$$

Are other ways needed to transform clocks?

(4) How to combine signals with different clocks? Since usual functions cannot do it, what kind of new operator on signals is needed for this purpose? For instance, we may write

funds := exceptional_funds default exportation_aid

to indicate that regular funds are: 1st/(i.e., with priority) exceptional_funds whenever needed, or 2nd/(i.e., by default) evaporation_aid in other cases. The clock we created in this way is the supremum of the clocks of the signals lying on the right-hand side. In particular, we have

$$clock(\text{exceptional\_funds}) \preceq clock(\text{funds})$$

Is anything else needed?

(5) How to define clock-dependent delay operators? Referring to (2), the two following ways to define the corresponding delayed signal could be considered:

delayed signal (1st idea):	s_0	1	−4	$\perp$	$\perp$	4	2	...
original signal:	1	−4	$\perp$	$\perp$		4	2	$\perp$ ...
delayed signal (2nd idea):	s_0	1	$\perp$	$\perp$	−4	4	$\perp$	...

where s_0 is some initial condition. The first idea is to define the delayed signal δs as $\delta s_n = s_{n-1}$, which seems at a first glance reasonable. Unfortunately, this definition makes explicit use of the time index n which is not desirable for the above discussed reasons. In the second idea, *present* values are shifted while keeping the clock unchanged: the notion of delay is clock-dependent and is *local* to each signal. This second idea is nothing but a mathematical model of the *shift register*: this is the definition we chose. To denote shift registers, we may for instance write

forecasted_funds := funds $ 5000

to mean that the current value of forecasted_funds is, according to the second idea above, the delayed (denoted by $)[3] value of funds with initial condition 5000 (bruteforce extrapolation is performed when the budget is voted). Consequently, we have

$$clock(\text{forecasted\_funds}) = clock(\text{funds})$$

(6) How to interconnect hybrid dynamical systems? This will be extremely easy because we decided to handle the time indexes in an implicit way. Here follows an example:

```
FARMERS_LOVE_RAIN=
| water := rain − evaporation
| help := (water > max_level)
| exceptional_funds := (500 flooded_acre) when help
| funds := exceptional_funds default exportation_aid
| forecasted_funds := funds $ 5000
```

This is just a system of (dynamical) equations, where the symbol "|" is used to denote linking. We provided it with a name for the sake of convenience. Three different "master" clocks are involved in this system: the clocks of the signals water, exceptional_funds, exportation_aids,[4] while other clocks are derived from these. Try to write this with explicit time indexes as in (1)! Let us emphasize that, despite the apparent explicit form of this system, it is an implicit one, due to the various constraints on the clocks we have imposed. This simple example justifies the relational form we started with in (1) for our hybrid dynamical systems (cf. Appendix A.II for a formal definition of HDS).

[3] Time is money.

[4] There is no reason to believe that exportation aids are delivered as frequently as the level of water is measured.

In the Appendixes A.I–A.II, a formal definition is given for the various objects ("signals, $\perp$, |") we introduced here informally, and this model is also used there as a mathematical model for the SIGNAL language.

2) Hybrid Dynamical Systems Resolution: A First Discussion: The constructive thrust of HDS theory is the resolution system. This problem reduces to that of investigating the existence and uniqueness of solutions to a set of equations. Here "solution" means an input–output map producing the same behaviors as the specified system. Here follow a few remarks.

A difference between hybrid- and discrete event dynamical systems theories. Consider again the system FARMERS_LOVE_RAIN. As we said before, it involves three different master clocks. The clocks of the signals water and exportation_aids are not constrained by the considered system of equations: these were specified as inputs, and their clocks as well as their values are free. However, the clock of the signal exceptional_funds is constrained by the value of the boolean signal help, which turns out to depend on the value of water. This possible dependency of clocks on values of other (nonboolean) signals is a special feature of HDS theory; since on the other hand the presence of signals can depend on clocks, *tricky signal/clock interdependences have to be taken into account when HDS resolution is considered.* This is a special feature of HDS compared to DEDS where transformations on integer (or real, etc.) values are considered as "hidden actions" performed when state transitions occur. The skeptical reader is urged to analyze carefully the Gonthier and MUX examples in Sections IIIA3 and IIIB.

HDS resolution and control problems. Up to year 19**, FARMERS_LOVE_RAIN was the law governing the delivery of funds for agriculture. At this time, W. W. LeGuen, a representative of the (French) middlewest, suggested to replace the FARMERS_LOVE_RAIN law by the following slightly different one:

```
FARMERS_LOVE_LEGUEN=
| water := rain − evaporation
| help := (water > max_level)
| exceptional_funds := (500 flooded_acre) when help
| funds := exceptional_funds default forecasted_funds
| forecasted_funds := funds $ 5000
```

This was voted, and the immediate consequence was the following trouble: funds had to be provided at least when exceptional_funds were requested, but could be delivered as frequently as requested! In fact, the last two equations impose only the constraints

$$clock(\text{exceptional\_funds}) \preceq clock(\text{funds}) = clock(\text{forecasted\_funds})$$

This left an unexpected degree of freedom. Farmers immediately proposed to maximize their income (a basic control problem) and proposed to receive funds as frequently as possible:

$$clock(\text{funds}) \equiv \top$$

(the clock which is more frequent than any other one). On the other hand, the national budget proposed to minimize the outcome (another basic control problem) and wanted to never deliver funds:

$$clock(\text{funds}) \equiv \perp$$

(the clock with no presence at all). This latter proposal was recognized as being in contradiction to the FARMERS_LOVE_LEGUEN law and was rejected. Finally, the compromise was to deliver funds as scarcely as possible,

$$clock(\text{funds}) = clock(\text{exceptional\_funds})$$

i.e., only when a flood occurred, so that farmers got angry.

This little story illustrates how the relational nature of HDS could be exploited based on the following philosophy.

- Describe the constraints imposed by the "physics" of the system (both rain and evaporation occur).
- Specify some additional constraints you wish to be satisfied by your system (a kind of model reference).
- Verify whether the whole is consistent, and/or free from ambiguity. In this case, construct the executable machine (it must be some I/O map) that can produce the specified behavior: this is what we called resolution.

On the other hand, questions have been raised such as: did we propose the right tools to model hybrid dynamical systems? Such questions can only be answered by some fundamental investigation involving formal mathematical models of HDS. This work is beyond the scope of the present paper and was, for instance, discussed in [2] and [3]. Incidentally, what we introduced was an outline of the SIGNAL[5] programming language.

B. SIGNAL-Kernel

To be concise, we shall introduce only the primitives of the SIGNAL language, and drop any reference to typing, modular structure, and various declarations; the interested reader is referred to [11]. As we have shown, SIGNAL handles (possibly infinite) sequences of data with time implicit: such sequences will be referred to as *signals*. At a given instant, signals may have the status *absent* (denoted by $\perp$) and *present*. If x is a signal, we shall denote by $\{x_n\}_{n\geq 1}$ the sequence of its values when it is present. Signals that are present simultaneously are said to have the same *clock*, so that clocks are equivalence classes of simultaneously present signals (a formal definition is given in the Appendix A.I). Instructions of SIGNAL are intended to relate clocks as well as values of the various signals involved in a given system. We shall term a system of such relations *program*; programs can be used as modules and further combined as indicated later.

A basic principle in SIGNAL is that a single name is assigned to every signal, so that in the sequel (and unless explicitly stated), identical names refer to identical signals. The kernel-language SIGNAL possesses 5 instructions, the first of them being a generic one.

i) R(x1, $\cdots$, xp)
ii) y := x $ x0
iii) y := x when b
iv) y := u default v
v) P | Q

Their intuitive meaning is as follows (for a formal definition, see Appendix A.III).

i) Direct extension of instantaneous relations into relations acting on signals:

$$\mathsf{R}(\mathsf{x}1,\cdots,\mathsf{xp}) \Leftrightarrow \forall n : R(\mathsf{x}1_n,\cdots,\mathsf{xp}_n) \text{ holds}$$

where $R(\cdots)$ denotes a relation and the index n enumerates the instants at which the signals xi are present. Examples are functions such as z := x + y ($\forall n$:$z_n = x_n + y_n$) or statements such as (a and b) or c = *true* ($\forall n$:(a_n *and* b_n) *or* c_n = *true*). A byproduct of this instruction is that *all referred signals must be present simultaneously, i.e., they must have the same clock*. This is a generic instruction, i.e., we assume a family of relations is available. If one chooses an instantaneous relation accepting any p-uple, the resulting SIGNAL instruction only constrains the involved signals to have the same clock: this is the way we derive the instruction written synchro x, y, $\cdots$ which only forces the listed signals to have the same clock.

ii) Shift register.

$$\mathsf{y} := \mathsf{x}\,\$\,\mathsf{x}0 \Leftrightarrow \forall n > 1 : y_n = x_{n-1},\ y_1 = \mathsf{x}0$$

[5] SIGNAL is a joint trademark of CNET and INRIA.

Recall that the index n refers to the values of the signals when they are present. Again this instruction forces the input and output signals to have the same clock.

iii) Condition (b is boolean): y equals x when the signal x and the boolean b are available and b is true; otherwise, y is not emitted; the result is an event-based undersampling of signals. Here follows a table summarizing this instruction:

x \ b	*true*	*false*	$\perp$
x	x	$\perp$	$\perp$
$\perp$	$\perp$	$\perp$	$\perp$

iv) y merges u and v, with priority to u when both signals are simultaneously present; this instruction is the key to oversampling as we shall see later. Here follows a table summarizing this instruction:

v \ u	u	$\perp$
v	u	v
$\perp$	u	$\perp$

The instructions i)–iv) specify the elementary programs.

v) combination of already defined programs: signals with common names in P and Q are considered as identical. For example

| y := zy + a
| zy := y $ x0

denotes the system of recurrent equations

$$y_n = zy_n + a_n$$

$$zy_n = y_{n-1},\ zy_1 = \mathsf{x}0.$$

We shall say that the smallest set of HDS containing the elementary systems specified by the instructions i)–iv) and closed under the interconnection operation | is the *algebra* of SIGNAL *programs*. A formal definition of this set of instructions is presented in Appendix A.III.

III. HDS Resolution: An Informal Presentation

In the preceding section, we have informally presented the SIGNAL language, and gave a first illustration to specify HDS. A corresponding formal model is found in the Appendix. The next section will be devoted to an informal presentation of HDS resolution, and investigation of examples that are tailored to illustrate the features of our theory. A formal presentation of HDS resolution is beyond the scope of this paper. The interested reader is referred to [5] for such a presentation.

A. Encoding SIGNAL Programs: A Tool for Resolution

HDS resolution aims at transforming implicit dynamical systems of the form (1) into an equivalent explicit I/O form. If no restriction is imposed on the nature of f, g, this will be generally impossible. To overcome this difficulty, we shall first define a map (or *coding*) from the algebra of SIGNAL programs onto a smaller algebra where resolution is possible. The idea behind this coding is the following. There are two basic types of tools for transforming and analyzing how computations are organized in general dynamical systems. The first one is the *signal flow graph* showing data dependencies; this may be sufficient for very regular algorithms such as those encountered in basic digital signal processing. The second one is the class of *finite state automata*

describing the scheduling, the branching, and related features of the dynamical system. We show next how to handle signal flow graphs which evolve dynamically under the control of an automation: the algebra of *dynamical graphs* we shall obtain in this way will be the subalgebra where we shall be able to perform resolution.

1) Dynamical Graphs: What we must handle jointly are the special value $\perp$, booleans, and nonboolean data dependencies. We shall first provide an algebra with a convenient calculus where the pairs $\{\perp, \text{booleans}\}$ can be represented. What we want to encode are the following status: *absent*, *present*, *true*, *false*, the last two for boolean signals only. These are encoded onto the finite field $\mathfrak{F}_3 = Z/3Z$ of integers modulo 3 as follows

$$true \leftrightarrow +1$$

$$false \leftrightarrow -1$$

$$absent \leftrightarrow 0$$

$$present \leftrightarrow \pm 1$$

where ± 1 denotes a nondeterminate choice of $+1$ or -1; i.e., we handle in the same way nonboolean signals and boolean ones that possess a nondeterminate value. HDS involving only boolean data types will be encoded via *dynamical systems* over $\mathfrak{F}_3^p$ for some integer p. Such dynamical systems will be generally denoted by the letter Δ, and are of the following form:

$$\xi_{n+1} = P(\xi_n, y_n)$$

$$0 = Q(\xi_n, y_n) \tag{3}$$

This form deserves some comment. In (3), $(\xi, y) \in \mathfrak{F}_3^p$, ξ is the state vector, y is the vector of the observed signals, and P, Q are polynomial vectors. Hence, the observation equation of usual dynamical systems is here a relation instead of a function: this is just a particular case of the general form (1) of HDS we introduced. In (3), we shall denote by $\{y(i)\}_{1 \le i \le I}$ the components of the vector y (so that we must have $I \le p$).

Let us now introduce dynamical graphs. A *dynamical graph* is a triple $\{\Delta, \Gamma, \gamma\}$ where:

- Δ is a dynamical system of the form (3). The purpose of Δ is to encode the logic and synchronization of the considered HDS.
- Γ is a directed graph with the symbols $\{y(i)\}_{1 \le i \le I}$ as its set of vertices. The graph Γ summarizes potential (nonboolean) data dependencies, as in signal flow graphs.
- γ is a function mapping $\mathfrak{F}_3^p$ into the set of the subgraphs of Γ. This map γ plays a key role in specifying the *actual* data dependencies at each instant (for this purpose, instants may be characterized by points in $\mathfrak{F}_3^p$).

Notice that the map γ is equally well-defined as follows: for each branch $y(i) \to y(j) \in \Gamma$, specify the subset of $\mathfrak{F}_3^p$ composed of the points x such that $\gamma(x)$ contains the considered branch. This is denoted by

$$\mathcal{V} : y(i) \to y(j) \text{ or } y(i) \overset{\mathcal{V}}{\to} y(j) \tag{4}$$

where $\mathcal{V}$ is the considered subset of $\mathfrak{F}_3^p$.

Hence, dynamical graphs are skew products of dynamical systems and graphs. The dynamical system is intended to encode the underlying automaton within a program, while the directed graphs will encode the way dependencies evolve during an execution: the dependencies at a given instant will only depend on the set of signals that are present in this instant.

2) Encoding SIGNAL Programs: Here follows the intuitive description of our method. Recall that SIGNAL is obtained by extending to multiple clocked dynamical systems a given "algebra" of instantaneous relations. This remark is the keystone of the method we present next to encode SIGNAL programs.

- *Step 1:* Among the relations of this algebra, select the subfamily of relations and corresponding data types for which you accept to solve systems of equations (and are supposed to be able to!).
- *Step 2:* Other instantaneous relations must be *functions*, and are encoded into their dependency graphs; hence corresponding data types are handled as sets of labels for which dependency graphs summarize all the possible rewritings or substitutions

$$y = f(x_1, \cdots, x_p) \text{ is encoded as } \{x_1 \to y, \cdots, x_p \to y\}.$$

This defines our map; its ability to reason about dynamical systems as well as its complexity relies on the choice we have done in Step 1. Here we shall select the boolean variables together with the boolean relations generated by {:=, and, or, not} and the constants *true*, *false*; this choice is motivated by the particular role played by the booleans in the instruction when.

Based on these remarks, the algebraic coding of SIGNAL programs into dynamical graphs is derived next. For this purpose, we shall use the following notations. For a signal with SIGNAL name zap, we shall write *zap* to denote its value at the considered instant (here, "value" means $\perp$ or the actual value if the signal is present), and we shall also write *zap* to denote the image of zap in the dynamical system over $\mathfrak{F}_3^p$.[6] Using these notations, we first show how presence/absence of signals of any type is encoded

$$x = \perp \leftrightarrow x^2 = 0$$

$$x \neq \perp \leftrightarrow x^2 = 1 \tag{5}$$

Only squares appear since the value of booleans plays no role here. The following formulas concern (present) boolean values:

$$b = true \leftrightarrow b = 1$$

$$b = false \leftrightarrow b = -1$$

$$b = not\, a \leftrightarrow b = -a$$

$$c = a \,and\, b \leftrightarrow c = 1 - (ab + a + b) \tag{6}$$

Let us illustrate how the coding works on the instruction y := x when b in the nonboolean case. Four cases occur according to the presence/absence of the various involved signals

$$i)\colon b^2 = 0, x^2 = 0, y^2 = 0$$

$$ii)\colon b^2 = 0, x^2 = 1, y^2 = 0$$

$$iii)\colon b^2 = 1, x^2 = 0, y^2 = 0$$

$$iv)\colon b = -1, x^2 = 1, y^2 = 0$$

$$v)\colon b = +1, x^2 = 1, y^2 = 1, x \to y.$$

Case v) is the only one where the output y is present. In this case, its value depends on the value of x: this is indicated with the $\to$ in v). These four cases can be summarized as the double coding

$$\mathsf{y := x\ when\ b} \leftrightarrow \left[\frac{y^2 = x^2(-b - b^2)}{y^2 : x \to y}\right] \tag{7}$$

where the first field of this coding is the equation of the dynamical system over $\mathfrak{F}_3$ (it is static here). In the second field of this coding, "y^2:" means that the dependency holds exactly when $y^2 = 1$. A systematic application of this method yields the algebraic coding of programs that we present next.

[6] No confusion should result in the sequel from this common notation.

The following notation will be used for this coding into dynamical graphs:

$$\text{program} \leftrightarrow \left[\frac{\text{clock calculus}}{\text{conditional dependency graph}}\right]$$

where

- program denotes the program to be encoded;
- "clock calculus" denotes the set of algebraic equations encoding the constraints on synchronization or logic as we discussed above; these equations define dynamical systems over $\mathfrak{F}_3^p$;
- "conditional dependency graph" denotes the set of possibly occurring dependencies together with the instants where these dependencies are in force.

Instruction i): Relation or Function:

Boolean Relation: The coding of all boolean relations is easily derived from the coding of the following instructions and the coding of the composition we shall see below:

$$\text{a} := \textit{true} \leftrightarrow \left[\frac{a^2 - a = 0}{\emptyset}\right]$$

$$\text{b} := \text{not a} \leftrightarrow \left[\frac{b = -a}{\emptyset}\right]$$

$$\text{c} := \text{a and b} \leftrightarrow \left[\begin{array}{c} a^2 = b^2 \\ \hline c = a^2 - (ab + a + b) \\ \hline \emptyset \end{array}\right]. \tag{8}$$

The algebraic equation of the first formula possesses $a = 1$, $a = 0$ as only solutions, which means that a is either absent or true. The second equation is obvious. To derive the last one, note that its first component encodes the fact that both signals a and b must have the same clock (they are either both present or absent, which is encoded as $a^2 = 1$ and $a^2 = 0$, respectively). Then it is straightforward to verify that the last equation maps the pairs (0, 0), (1, 1), (−1, 1), (1, −1), (−1, −1) onto 0, 1, −1, −1, −1, respectively. Since only booleans are involved, no coding of dependencies is required, hence the symbol $\emptyset$ in the second field.

Nonboolean Function:

$$\text{y} := \text{f}(\text{x}_1, \cdots, \text{x}_p) \leftrightarrow \left[\frac{y^2 = x_1^2 = \ldots = x_p^2}{y^2 : x_1 \rightarrow y, \cdots, x_p \rightarrow y}\right]. \tag{9}$$

The first field encodes the constraints on clocks (equality), while the second one encodes the data dependencies. The second field means "the listed dependencies hold when $y^2 = 1$." Notice that a := (u < v) produces a boolean, but it is a nonboolean function.

Instruction ii): The Register:

Boolean Register: This is the key case where dynamical systems in $\mathfrak{F}_3$ are used.

$$\text{b} := \text{a}\,\$\,u \leftrightarrow \left[\begin{array}{c} \xi' = (1 - a^2)\xi + a,\ \xi_1 = u \\ \hline b = a^2\xi \\ \hline \emptyset \end{array}\right] \tag{10}$$

where ξ' is the current state of the dynamical system, ξ its previous state, and u its initial condition (± 1-valued). The corresponding explicit form of this dynamical system is

$$\xi_{n+1} = (1 - a_n^2)\xi_n + a_n,\ \xi_1 = u$$

$$b_n = a_n^2 \xi_n$$

where n is any time index fast enough to capture every presence of signal. Notice that the state takes +1 or −1 as only values, i.e., states are persistent. The state is modified when a new input is received, and at the same instant the old state is delivered at the output. Again no dependency graph is necessary.

Nonboolean Register:

$$\text{y} := \text{x}\,\$\,u \leftrightarrow \left[\frac{y^2 = x^2}{\emptyset}\right]. \tag{11}$$

The first field expresses that clocks must be identical; the second field is empty even if we consider nonboolean types, since the current value of y does not depend on the current value of x, but on the content of the memory (which has been lost in the coding).

Instruction iii): The when:

when *with boolean output.*

$$\text{c} := \text{a when b} \leftrightarrow \left[\frac{c = a(-b - b^2)}{\emptyset}\right]. \tag{12}$$

when *with nonboolean output.*

$$\text{y} := \text{x when b} \leftrightarrow \left[\frac{y^2 = x^2(-b - b^2)}{y^2 : x \rightarrow y}\right]. \tag{13}$$

The second field expresses that x influences y when y is produced.

Instruction iv): The default.

default *with boolean output.*

$$\text{c} := \text{a default b} \leftrightarrow \left[\frac{c = a + b(1 - a^2)}{\emptyset}\right]. \tag{14}$$

default *with nonboolean output.*

$$\text{y} := \text{u default v} \leftrightarrow \left[\begin{array}{c} y^2 = u^2 + v^2(1 - u^2) \\ \hline u^2 : u \rightarrow y \\ \hline v^2(1 - u^2) : v \rightarrow y \end{array}\right]. \tag{15}$$

The second field expresses the fact that u influences y when it is present, while v influences y when it is present and u is absent.

Instruction v): The Composition: The fields of P and Q have to be merged to produce a single clock field and a single conditional dependency graph field.

The General Form of Coding for a SIGNAL Program: By combining the elementary codings (8)–(15), we derive the following form of dynamical graph to encode an arbitrary SIGNAL program (cf. (3) and (4) for the undefined notations):

$$\xi_{n+1} = P(\xi_n, y_n)$$

$$0 = Q(\xi_n, y_n)$$

$$y(i) \overset{h(i,j)}{\rightarrow} y(j) \tag{16}$$

where $h(i, j)$ is a polynomial expression involving the variables $y(k)$ in $\mathfrak{F}_3$ which specifies when the considered dependency holds.

C. Examples

1) A Synchronized Memory: The output y returns either the present value of x (when the latter is present), or the last received value of x when b is present and true. We call the corresponding program CELL:

```
CELL (input: x,b output: y) =
| zy := y $ y0
| y := x default zy
| synchro y, (x default (true when b))
```

When encoding this program, we shall make a distinction between two cases: x boolean, and x nonboolean.

Encoding the Boolean CELL *into its Clock Calculus.*

$$\xi' = (1 - y^2)\xi + y,\ \xi_1 = y_0$$

$$zy = y^2\xi$$

$$y = x + zy(1 - x^2)$$

$$y^2 = x^2 + (-b - b^2)(1 - x^2).$$

Eliminating zy and rewriting ξ' as a function of the input x yields

$$\xi' = (1 - x^2)\xi + x,\ \xi_1 = y_0$$

$$y = x + (-b - b^2)(1 - x^2)\xi$$

which reflects exactly the meaning of the program CELL: the memory is refreshed when x is received (first equation), and y delivers the current value of x when x is received, or its last value when b is received and true.

Encoding the Nonboolean CELL *into its Clock Calculus and Conditional Dependency Graph.*

Clock calculus:

$$zy^2 = y^2 = x^2 + zy^2(1 - x^2)$$

$$y^2 = x^2 + (-b - b^2)(1 - x^2)$$

which yields

$$zy^2 = y^2 = x^2 + (-b - b^2)(1 - x^2).$$

Conditional dependency graph:

$$zy \overset{y^2(1-x^2)}{\rightarrow} y \overset{x^2}{\leftarrow} x.$$

Here the dynamics has been lost; the clock calculus expresses only how the clocks of the signals are related.

2) An Example of Deadlock: The following example is due to G. Gonthier (private communication). An if...then...else statement is provided as a macro in the full SIGNAL language. Using this macro, we can write the following SIGNAL instruction:

```
if z > 0 then z := a else z := b
```

The expansion of this program in terms of the primitive instructions we have given in the following:

```
| synchro a,b
| beta = (z > 0)
| x := a when beta
| y := b when not beta
| z := x default y
```

Denoting by β the image of beta, the clock calculus is

$$a^2 = b^2$$

$$\beta^2 = z^2$$

$$x^2 = a^2(-\beta - \beta^2)$$

$$y^2 = b^2(\beta - \beta^2)$$

$$z^2 = x^2 + y^2(1 - x^2).$$

Then, we set $h = a^2 = b^2$, and, combining the above equations, we get

$$a^2 = b^2 = h$$

$$x^2 = h(-\beta - \beta^2)$$

$$y^2 = h(\beta - \beta^2)$$

$$\beta^2 = z^2 = x^2 + y^2(1 - x^2) = h\beta^2.$$

Now, consider the last equation $\beta^2 = h\beta^2$. It is equivalent to

$$\beta^2 = \Phi^2 h$$

where Φ is a *free* variable of $\mathfrak{T}_3$, which we shall call a *phantom*. Hence, the clock calculus can be rewritten as follows (the deleted equation was redundant):

$$a^2 = b^2 = h$$

$$x^2 = h(-\beta - \beta^2)$$

$$y^2 = h(\beta - \beta^2)$$

$$\beta^2 = z^2 = \Phi^2 h. \tag{17}$$

Notice that the presence of the phantom reflects the fact that the clock of the signals z and β is not completely determined by the inputs a,b. From this form of the clock calculus, it appears that h is the fastest clock, so that we can assume

$$h \equiv 1. \tag{18}$$

Finally, using (18), (17) is rewritten as

$$a^2 = b^2 \equiv 1$$

$$x^2 = -\beta - \beta^2$$

$$y^2 = +\beta - \beta^2$$

$$\beta^2 \quad \text{free.} \tag{19}$$

Using again (18), the conditional dependency graph is

$$a \overset{-\beta-\beta^2}{\rightarrow} x \overset{-\beta-\beta^2}{\rightarrow} z \overset{\beta^2}{\rightarrow} \beta$$

$$b \overset{+\beta-\beta^2}{\rightarrow} y \overset{+\beta-\beta^2}{\rightarrow} z \overset{\beta^2}{\rightarrow} \beta. \tag{20}$$

In both cases, β appears both at the end of the last branch of the graph, and as a label of the first two branches. This is a sort of a cycle where both *control and data dependencies* are involved. This is handled with the following procedure we outline now (the reader is referred to [5] for a detailed presentation of this procedure and the formal model to support it).

From the clock calculus (19), it is seen that the value of β does influence both clocks x^2 and y^2: let us encode this via the following graph:

$$\beta \rightarrow x$$

$$\beta \rightarrow y. \tag{21}$$

Combining (20) and (21), we get the following labeled graph:

$$a \overset{-\beta-\beta^2}{\rightarrow} x \overset{-\beta-\beta^2}{\rightarrow} z \overset{\beta^2}{\rightarrow} \beta \rightarrow x$$

$$b \overset{+\beta-\beta^2}{\rightarrow} y \overset{+\beta-\beta^2}{\rightarrow} z \overset{\beta^2}{\rightarrow} \beta \rightarrow y. \tag{22}$$

Two cycles are exhibited. The first cycle is effective when $\beta = +1$, while the second one is effective when $\beta = -1$. Hence, to prevent both cycles we must enforce $\beta^2 = 0$. Consequently, this program accepts the inputs (a,b), but refuses to produce any other signal.

This illustrates informally how deadlocks can be detected and isolated by taking into account clocks and data dependencies. This example also reveals why *taking into account only the dynamical system over* $\mathcal{F}_3$ *in the coding of* SIGNAL *programs may result in dramatic errors in deadlock detection:* no deadlock is detected in this example from the inspection of the clock calculus only. This is perhaps the most illustrative example of the additional difficulty encountered in handling HDS as compared to DEDS.

3) A Time-Multiplexer: Here follows a desired behavior of the program we shall write next:

$$\textit{input}\ \mathsf{r}: 2\ \perp\ \perp\ 4\ \perp\ \perp\ \perp\ \perp\ 0\ 5\ \perp\ \ldots$$

$$\textit{output}\ \mathsf{n}: 2\ 1\ 0\ 4\ 3\ 2\ 1\ 0\ 0\ 5\ 4\ \ldots$$

The input r is a nonnegative integer. When r is read and takes the value r, r additional samples are produced for the output n before the next value of r is read: this is a (variable rate) time-multiplexer which is a basic tool to construct oversampling. Here follows the program:

```
MUX (input: r, output: n) =
| n := r default (zn − 1)
| zn := n $ 0
| b := (n = 0)
| past_b := b $ true
| true_past_b := true when past_b
| synchro true_past_b, r
```

The first two instructions define a decreasing counter n with reset signal r. The boolean signal b tests for n = 0, past_b is the delayed signal b, while true_past_b extracts the instants where past_b is true. Therefore, when n reaches 0, the next time n is produced, a new sample of the input signal r must be read, due to the last instruction synchro which constrains the clocks of the listed signals to be equal. Notice that, in this way, we implemented an *oversampling*, so that both requests about clock changes that were expressed when introducing the FARMERS_LOVE_RAIN story are now satisfied: SIGNAL *provides tools for both (variable rate) under- and over-sampling*.

We shall encode this program. Clock calculus and dependency graph will be written with the short signal names pb, tpb instead of the full names past_b, true_past_b. This coding is written instruction-by-instruction:

$$n^2 = r^2 + (1 - r^2)zn^2,\ r \xrightarrow{r^2} n,\ zn \xrightarrow{(1-r^2)zn^2} n$$

$$zn^2 = n^2$$

$$b^2 = n^2,\ n \xrightarrow{n^2} b$$

$$\begin{cases} \xi' = (1 - b^2)\xi + b,\ \xi_1 = 0 \\ pb = b^2\xi \end{cases}$$

$$tpb = -pb - pb^2$$

$$tpb^2 = r^2. \tag{23}$$

The first two equations of the clock calculus are equivalent to

$$n^2 = zn^2 = r^2 + \Phi^2(1 - r^2)$$

where Φ is a phantom. This phantom reflects the fact that, in the first two instructions

```
| n := r default (zn − 1)
| zn := n $ 0
```

of the program, the clock of the output n is not completely determined by the clock of the input r alone: the only constraint on synchronization we get is that n must be more frequent than r.

Using the last instruction synchro allows us to remove this phantom: the whole clock calculus can be rewritten as follows:

$$\begin{cases} \xi' = (1 - b^2)\xi + b,\ \xi_1 = 0 \\ pb = b^2\xi \end{cases}$$

$$zn^2 = n^2 = pb^2 = b^2$$

$$r^2 = tpb^2 = tpb = -pb - pb^2 \tag{24}$$

while the corresponding conditional dependency graph is now

$$r \xrightarrow{r^2} n,\ zn \xrightarrow{(1-r^2)b^2} n,\ n \xrightarrow{b^2} b. \tag{25}$$

Considering only the clock calculus (24), it looks like if b was the input (all other clocks are determined from b). However, this boolean signal is *not* an input of the program: this paradox is due to the fact that, in considering the clock calculus alone, we just disregarded the dependencies encoded in the conditional dependency graph (25). This reveals again the intricate mutual interaction between graphs and automata in HDS theory. To further investigate this point, we shall now discuss in details on this example how HDS resolution is performed.

B. HDS Resolution: The Example of the Time-Multiplexer

To show that the form (24), (25) corresponds to an already *solved* HDS, we shall simply show how the finite state machine works, which produces the desired behavior. This is described next. First, note that in (24), we are free to take $b^2 \equiv 1$. Here is a description of this finite state machine.

FOR EACH INSTANT DO:

Initial stage: a new instant begins.

$$\begin{cases} \xi' = b,\ \xi_1 = 0 \\ pb = \xi \end{cases}$$

$$\begin{cases} 1 = zn^2 = n^2 = pb^2 = b^2 \\ r \xrightarrow{r^2} n,\ zn \xrightarrow{(1-r^2)} n,\ n \to b \end{cases}$$

$$r^2 = tpb^2 = tpb = -pb - 1.$$

The initial stage is equivalent to (24), (25) if we take into account $b^2 \equiv 1$. Two cases may occur, depending on the value of the current state ξ.

Case 1: $\xi = +1$.

Step 1: r^2 is evaluated to 1. The branches labeled with a 0 are removed (the corresponding dependency is not in force at the considered instant). Some branches remain with a label 1 (the corresponding dependency is in force), we remove this label 1 for the sake of simplicity. Finally, the signals and clocks that have been evaluated and will not be used any more are removed. This yields

$$\xi' = b$$

$$\begin{cases} 1 = n^2 = b^2 \\ r \to n,\ n \to b \end{cases}$$

$$r^2 = -1 - 1 = 1.$$

Step 2: Since $r^2 = n^2 = 1$ and $r \rightarrow n$, n can be evaluated

$$\xi' = b$$

$$\begin{cases} 1 = n^2 = b^2 \\ n \rightarrow b. \end{cases}$$

Step 3: b is ready to be evaluated

$$\xi' = b.$$

Step 4: The next value of the state can be computed

$$\emptyset.$$

This ends the considered instant.
Case 2: $\xi = -1$. Here follow the corresponding steps.
Step 1:

$$\xi' = b$$

$$\begin{cases} 1 = zn^2 = n^2 = b^2 \\ zn \rightarrow n, n \rightarrow b \end{cases}$$

$$r^2 = 0.$$

Step 2:

$$\xi' = b$$

$$\begin{cases} 1 = n^2 = b^2 \\ n \rightarrow b \end{cases}.$$

Step 3, Step 4, as before.
This execution scheme revealed what really are the inputs of this HDS. Knowing that $b^2 \equiv 1$ is a first input (it is here a trivial information). Knowing this, it is possible to read the value of the current state ξ: this indicates whether the input signal r should be read or not, and allows us to evaluate n, then b, and finally to write the latter value in the next state. *This shows that the input of this system is split into 1) its "internal clock"; 2) the values of the input reset signal* r *when the latter is read.*

C. Discussion

1) On the "Graph Peeling" Mechanism: The MUX example did not really involve numerics, except for the $zn - 1$ instruction. But the mechanism of *clock dependent graph peeling* we presented as a key step of the execution machine is obviously valid for numerical computations as well. The MUX example and the example of deadlock due to G. Gonthier illustrate best why the mixed nature of HDS makes their study more difficult than the study of DEDS.

2) Solving HDS: A formal model supporting HDS resolution is presented in [5] using the technique of "conditional rewriting rules" borrowed from formal semantics of languages in computer science.

3) Studying Dynamical Systems in $\mathfrak{F}_3$: A systematic study of the dynamical systems of the form (3) is presented in [18]. This study is control oriented, and covers some of the classical topics of DEDS. It is entirely based on the tools of polynomial ideal theory.

IV. Conclusion

We have presented a brief account of HDS theory. Notice the following points.

- HDS involve both numerics and symbolics (logic and synchronization), which makes them of wide applicability, but also difficult to analyze. The reader interested in a formal presentation of the theory is referred to [3] for a control oriented presentation, and to [5] for a computer science oriented one.
- Our theory is supported by the SIGNAL programming language. A version of the SIGNAL compiler is currently experimented with in some universities, and a block-diagram oriented interface is available to specify systems in a control engineering style. The compiler produces as an intermediate level code the pair clock calculus, conditional dependence graph; from this intermediate level code, executable Fortran code, but also distributed OCCAM code (programming language of the "Transputer" by INMOS), is generated. Current applications to test the language are signal processing systems, radar systems, and a whole continuous speech recognition application. The existence of this programming language makes the treatment of real applications really feasible, cf. [15]. A thorough discussion of SIGNAL from the user's viewpoint is presented in [6].
- Our approach is relational, which allows us to specify HDS via constraints involving logic and synchronization, to analyze them, and to execute them. This relational approach provides a "proof system" for these properties, similar to (but weaker than) temporal logic [26]. This point is further investigated in [18] using a control point of view and polynomial ideal theory. The main problem is then HDS resolution, i.e., transforming the implicit specification into an explicit input/output map.
- Our theory captures the notion of deadlock; a detailed discussion of this and other properties is presented in [18].

Finally, let us mention related work on synchronous languages: the declarative (but functional) language LUSTRE [7] [10], the imperative language ESTEREL [8] [12] the syntax of which possesses some flavor of ADA. Loosely related to the same field are the STATECHARTS and STATEMATES developed by Harel and Pnueli [13], [14], which provide a graphical interface to specify in a hierarchical way states and transitions in automata.

Appendix A

A Mathematical Model for HDS and the SIGNAL Language

In this Appendix, a mathematical model for HDS is presented, and used to formally define SIGNAL. The reader is referred to Section II for the motivation of the following definitions.

1. Signals, Clocks

Consider an alphabet (finite set) A of typed variables called *ports*. For each $a \in A$, $\mathfrak{D}_a$ is the domain of values (integers, reals, booleans...) that may be carried by a at every instant. Introduce

$$\mathfrak{D}_A = \bigcup_{a \in A} \mathfrak{D}_a, \mathfrak{D}_A^{\perp} = \mathfrak{D}_A \cup \{ \perp \}$$

where the additional symbol $\perp$ denotes the absence of the value associated with a port at a given instant. For two sets A and B, the notation $A \rightarrow B$ will denote the set of all maps defined from A into B. Using this notation, we introduce the following objects.

Events: Events specify the values carried by a set of ports at a considered instant. The set of the A-events (or "events" for short when no confusion is likely to occur) is defined as

$$\mathcal{E}_A = A \rightarrow \mathfrak{D}_A^{\perp}.$$

Events will be generally denoted by ϵ. We shall denote by $\perp$ the event ϵ such that $\epsilon(a) = \perp \forall a \in A$.

2) Traces: Traces are infinite sequences of events. Let N denote the set of integers, then the set of *A-traces* (or simply "traces") is defined as

$$\mathfrak{I}_A = N \rightarrow \mathcal{E}_A.$$

3) Compressions: The *compression* of an A-trace T (deleting

the $\perp$'s) is defined as the (unique) A-trace S such that

$$S(n) = T(k_n)$$

where

$$k_0 = \min\{m \geq 0 : T(m) \neq \perp\},$$
$$k_n = \min\{m > k_{n-1} : T(m) \neq \perp\}.$$

The compression of a trace T will be denoted by $T\downarrow$.

4) Signals: The condition

$$T\downarrow = T'\downarrow$$

defines an equivalence relation on traces we shall denote by $T \sim T'$. The corresponding equivalence classes are called *multiple signals*, and simply *signals* when the set A is reduced to a singleton. The set of multiple signals on A will be denoted by $\mathcal{S}_A$, so that we have ($\cdot_{/\sim}$ denotes here the quotient space by the relation $\sim$)

$$\mathcal{S}_A = (\mathcal{T}_A)_{/\sim}.$$

The notion of "signal" has been informally discussed in Section IIA1(1), where we motivated the formal definition of signals as equivalence classes with respect to the relation $\sim$. Clearly, the symbol $\perp$ is useful when *components of multiple signals* are considered, such components are frequently simply referred to as "signals" in the informal discussion.

5) Clocks: Extend the domain $\mathcal{D}_A^{\perp}$ with another distinguished value $\top$, intended to encode the status "present" regardless of any particular value, and write $\mathcal{D}_A^{clock} = \{\perp\} \cup \{\top\}$. We define on $\mathcal{D}_A^{clock}$ the order relation $\prec$ by setting $\perp \prec \top$. Consider the map $clock_{\mathcal{D}} \in \mathcal{D}_A^{\perp} \to \mathcal{D}_A^{clock}$ defined by

$$clock_{\mathcal{D}}(\perp) = \perp,\ clock_{\mathcal{D}}(x) = \top \qquad \text{for } x \neq \perp.$$

For each $\epsilon \in \mathcal{E}_A$, there is a unique map in $\mathcal{E}_A \to \mathcal{E}_A$ making the following diagram commutative, denote it by $clock_{\mathcal{E}}$:

$$\begin{array}{ccc} & A & \\ \epsilon & \swarrow \quad \searrow & \mathrm{clock}_{\mathcal{E}}(\epsilon) \\ \mathcal{D}_A^{\perp} & \xrightarrow{\mathrm{clock}_{\mathcal{D}}} & \mathcal{D}_A^{\mathrm{clock}} \end{array}$$

Similarly, there is a unique map in $\mathcal{T}_A \to \mathcal{T}_A$, we denote by $clock_{\mathcal{T}}$, making the following diagram commutative:

$$\begin{array}{ccc} & N & \\ T & \swarrow \quad \searrow & \mathrm{clock}_{\mathcal{T}}(T) \\ \mathcal{E}_A & \xrightarrow{\mathrm{clock}_{\mathcal{E}}} & \mathcal{E}_A \end{array}$$

This map satisfies the condition $T_1 \sim T_2 \Rightarrow clock_{\mathcal{T}}(T_1) \sim clock_{\mathcal{T}}(T_2)$, so that it induces a map in $\mathcal{S}_A \to \mathcal{S}_A$ we shall now denote by *clock*: the clock of a (multiple) signal is another (multiple) signal which summarizes the status {present/absent} of each of its components.

In Section IIA2, we discussed informally some relations on clocks of signals involved in a given SIGNAL program. By this, we had in mind the following. A SIGNAL program defines a subset of "admissible" multiple signals (see below). The components of these multiple signals are denoted by the different names used in the program. Hence, by *clock*(zap), we mean the image by the map *clock* of the component zap of the considered multiple signal.

Finally, the order $\prec$ we introduced above on the pair $\{\perp, \top\}$ can be carried out via the above construction to obtain the order on clocks denoted by $\preceq$ in Section IIA2.

II. HDS

1) Definition of HDS: An HDS is simply a subset

$$\mathcal{HC} \subset \mathcal{S}_A$$

of the set of all multiple signals on A.

2) Restricting HDS: Consider a subset of A' of the alphabet A. The inclusion $A' \subset A$ induces a projection $\epsilon \to \epsilon_{/A'}$ from $\mathcal{E}_A$ onto $\mathcal{E}_{A'}$. Following the same argument as for the definition of clocks, we derive the following family of *restrictions* we generically denote by $\cdot_{/A'}$. First, the following commutative diagram:

$$\begin{array}{ccc} & N & \\ T & \swarrow \quad \searrow & T_{/A'} \\ \mathcal{E}_A & \xrightarrow{\cdot_{/A'}} & \mathcal{E}_{A'} \end{array}$$

uniquely defines the restriction $T \to T_{/A'}$ on traces. Since $T_1 \sim T_2 \Rightarrow (T_1)_{/A'} \sim (T_2)_{/A'}$ holds, a restriction on signals $S \to S_{/A'}$ can be defined, which finally yields a restriction on HDS we denote by

$$\mathcal{HC} \to \mathcal{HC}_{/A'}.$$

This restriction maps the set of HDS defined over the alphabet A onto the set of HDS defined over the alphabet A'. The HDS $\mathcal{HC}_{/A'}$ is called the *restriction* of $\mathcal{HC}$ to (the subalphabet) A'.

3) Combining HDS: Consider two HDS $\mathcal{HC}_1$, $\mathcal{HC}_2$, respectively, defined over the alphabets A_1 and A_2. Set $A = A_1 \cup A_2$. Then $\mathcal{HC}_1|\mathcal{HC}_2$ will denote the maximal[7] HDS defined over the alphabet A satisfying the following conditions:

$$(\mathcal{HC}_1|\mathcal{HC}_2)_{/A_1} \subseteq \mathcal{HC}_1$$
$$(\mathcal{HC}_1|\mathcal{HC}_2)_{/A_2} \subseteq \mathcal{HC}_2$$

The meaning of these conditions is that both constraints induced by $\mathcal{HC}_1$ and $\mathcal{HC}_2$ have to be satisfied by $\mathcal{HC}_1|\mathcal{HC}_2$: this is exactly what "combining two systems of equations" usually means (see [19] for a similar construction).

III. The Definition of SIGNAL

According to the preceding section, in order to specify an HDS over a given alphabet, we have to describe a subset of all multiple signals that can be built upon this alphabet. Since signals are defined as equivalence classes of traces with respect to the relation $\sim$, this can be done by *listing a family of constraints on the set of all traces* that can be built on this alphabet. This is what we shall do next.

Instruction i): R(x1,$\cdots$xp)

$$\forall n \in \mathbf{N} : \mathrm{xi}(n) \neq \perp\ \forall \mathrm{i}$$
$$\forall n \in \mathbf{N} : R(\mathrm{xi}(n), \cdots, \mathrm{xp}(n)) \text{ holds.}$$

Here, the notation xi(n) denotes the value carried by the port with name xi at the nth instant of the considered trace. This notation will be further used in the sequel.

Instruction ii): y := x $ x0

$$\forall n \in \mathbf{N} : \mathrm{x}(n) \neq \perp$$
$$\forall n > 1 : \mathrm{y}(n) = \mathrm{x}(n-1)$$
$$\mathrm{y}(1) = \mathrm{x}0.$$

Instruction iii): y := x when b

$$\forall n \in \mathbf{N},\ \mathrm{y}(n) = \begin{cases} \mathrm{x}(n) & \text{if } \mathrm{x}(n) \neq \perp \text{ and } \mathrm{b}(n) = \textit{true} \\ \perp & \text{otherwise.} \end{cases}$$

[7] With respect to the order by inclusion $\mathcal{HC}' \subseteq \mathcal{HC}$ defined on HDS.

Instruction iv): y := u default v

$$\forall n \in \boldsymbol{N},\ y(n) = \begin{cases} u(n) & \text{if } u(n) \neq \perp \\ v(n) & \text{if } u(n) = \perp \text{ and } v(n) \neq \perp \\ \perp & \text{otherwise.} \end{cases}$$

Instruction v): P | Q
We already defined the operator | on HDS.

Appendix B

A Sample Work of the SIGNAL Compiler: The Program MUX

Here is the syntactically correct form of the program MUX:

```
process MUX=
      { ? integer R
        ! integer N }
      (| N := R default ( ZN - 1 )
       | ZN := N $1
       | B := N = 0
       | PAST_B := B $1
       | synchro { when PAST_B, R }
       |)!! N
      where
            logical B, PAST_B init true;
            integer ZN init 0
end
```

Some differences with the syntax we used in the paper are mentioned now. The symbol $1 denotes a unit delay ($n is also available to denote an n-delay), the initial condition of ZN is given in the where field. The field { ?..., !... } specifies the interface (? stands for "input", and ! for "output"). At the end of the body, !!N means that N is the only output visible from outside (this is redundant with the interface specification).

The translation of MUX results in the following SIGNAL program which specifies now a "solved" HDS. This means that synchronization, boolean, and nonboolean data are determined by functions instead of relations: this HDS is ready for execution. The special work "event" denotes:

- a type "pure clock signal" (a boolean which always carries the value *true*) when encountered in the specification of the interface;
- the clock of the mentioned signal when encountered in the body of the program (event N_2 is the first example).

The symbols H_**_H refer to clocks that are synthesized by the clock calculus, or to modules that are fired according to the clock with the mentioned name (cf. for instance process H_10_H).

```
process MUX_TRA=
      { ? integer R_1
        ! integer N_2 }
      (| (| H_10_H := event N_2
         | synchro { H_10_H, N_2 }
         | H_10_H( )
         |)
       |)/ H_11_H, H_10_H
      where
            event H_11_H, H_10_H
      process H_10_H=
            { ? event H_10_H;
                integer R_1
              ! event H_11_H;
                integer N_2 }
            (| synchro { H_10_H, B_6, PAST_B_7, ZN_8 }
             |(| H_11_H := when PAST_B_7
              | synchro { H_11_H, R_1 }
              | H_12_H := when( not PAST_B_7 )
              |)
             |(| ZN_8 := N_2 $1
              | PAST_B_7 := B_6 $1
              | N_2 := ( R_1 when H_11_H )
                default( ( ZN_8 - 1 )when H_ 12_H )
              | B_6 := ( N_2 = 0 )when H_10_H
              |)
            |)/ H_12_H, B_6, PAST_B_7, ZN_8
           where
                event H_12_H;
                logical B_6, PAST_B_7 init true;
                integer ZN_8 init 0
      end
end
```

The above program is ready for sequential (i.e., centralized) as well as parallel (i.e., distributed) implementation.

The compiler also produces the following Fortran Subroutine (an instance of sequential implementation). This subroutine calls user-defined input/output functions for each of the input/output signals.

```
C   SIGNAL P1.5-(Fevrier 1989-release September 1989)-
        IRISA

      SUBROUTINE SMUX

      INTEGER R1

      INTEGER N2

      INTEGER ZN8
      LOGICAL PASTB7,B6
C List of clocks

      LOGICAL h10h,h12h,h11h

C Body of the program
      ENTRY CMUX(h10h)
      h10h = .TRUE.
      h11h = PASTB7
      h12h = .NOT. h11h
      IF (h12h)THEN
      N2 = ZN8 - 1
      ENDIF
      IF (h11h)THEN
C Reading input R
      CALL RR (R1,h10h)
      IF ( .NOT. (h10h))RETURN
      N2 = R1
      ENDIF
C Producing output N
      CALL WN (N2)
      B6 = N2 .EQ. 0
C Handling delays
      PASTB7 = B6
      ZN8 = N2

      RETURN

C Initialization
      ENTRY IMUX

      ZN8 = 0
      PASTB7 = .TRUE.

      RETURN
      END
```


Acknowledgment

The authors wish to thank B. Le Goff and C. Ozveren for fruitful discussions and suggestions, and an anonymous reviewer who performed an outstanding work in rephrasing for a control audience the motivations for our theory.[8]

[8] However, the authors are entirely responsible for the illustration example related to social science, and for all kinds of excommunication that could result.

REFERENCES

[1] K. J. Astrom, J. J. Anton, and K. E. Arzen, "Expert control," *Automatica,* vol. 22, no. 3, pp. 277–286.

[2] A. Benveniste and P. Le Guernic, "A denotational theory of synchronous communicating systems," INRIA Research Rep. 685.

[3] A. Benveniste, B. Le Goff, and P. Le Guernic, "Hybrid dynamical systems theory and the language SIGNAL," INRIA Research Rep. 838, 1988.

[4] A. Benveniste and P. Le Guernic, "Hybrid dynamical systems theory and nonlinear dynamical systems over finite fields," in *Proc. 1988 IEEE CDC,* Austin, TX, Dec. 7–9, 1988.

[5] A. Benveniste, P. Le Guernic, and C. Jacquemot, "Synchronous programming with events and relations: The SIGNAL language and its semantic," IRISA Research Rep., 1989.

[6] ——, "The SIGNAL software environment for real-time system specification, design, and implementation," in *Proc. 1989 IEEE Workshop on Comput.-Aided Contr. Syst. Design,* Tampa, FL, Dec. 16, 1989.

[7] J. L. Bergerand, P. Caspi, N. Halbwachs, D. Pilaud, and E. Pilaud, "Outline of a real-time data-flow language," in *Proc. Real Time Systems Symp.,* San Diego, CA, Dec. 1985.

[8] G. Berry and G. Gonthier, "The ESTEREL synchronous programming language: design, semantics, implementations," CMA Research Rep.; also in *Sci. Comput. Programming,* to be published.

[9] S. D. Brookes, C. A. R. Hoare, and A. W. Roscoe, "A theory of communicating sequential processes," *J. ACM,* vol. 31, no. 3, pp. 560–599.

[10] P. Caspi, D. Pilaud, N. Halbwachs, and J. A. Plaice, "LUSTRE: A declarative language for programming synchronous systems," in *Proc. 14th ACM Symp. Principles of Programming Languages,* 1987.

[11] T. Gautier, P. Le Guernic, and L. Besnard, "SIGNAL, A declarative language for synchronous programming of real-time systems," in *Proc. 3rd Conf. Functional Programming Languages and Computer Architecture,* G. Kahn Ed. (Lect Notes in Computer Science, Vol. 274). New York: Springer-Verlag, 1987.

[12] G. Gonthier, thesis, Univ. de Nice and Ecole des Mines, 1988.

[13] D. Harel, "Statecharts: A visual approach to complex systems," *Sci. Comput. Programming,* vol. 8, no. 3, pp. 231–275, 1987.

[14] D. Harel and A. Pnueli, "On the development of reactive systems: Logic and models of concurrent systems," in *Proc. NATO Advanced Study Institute on Logics and Models for Verification and Specification of Concurrent Systems* (NATO ASI Series F., Vol. 13). New York: Springer-Verlag, 1985, pp. 477–498.

[15] Y. C. Ho, "Basic research, manufacturing automation, and putting the cart before the horse," *IEEE Trans. Automat. Contr.,* vol. AC-32, pp. 1042–1043, Dec. 1987.

[16] C. A. R. Hoare, "Communicating sequential processes," *Commun. ACM,* vol. 21, no. 8, pp. 666–678.

[17] K. Inan and P. Varaiya, "Finitely recursive processes," in *Proc. CDC,* 1987, pp. 252–256.

[18] M. Le Borgne, A. Benveniste, and P. Le Guernic, "Polynomial ideal theoretic methods in discrete event, and hybrid dynamical systems," in *Proc. 1989 CDC,* Tampa, FL, Dec. 13–15, 1989.

[19] B. Le Goff, thesis, IRISA, Univ. Rennes I, 1989.

[20] P. Le Guernic, A. Benveniste, P. Bournai, and T. Gautier, "SIGNAL: A data-flow oriented language for signal processing," *IEEE Trans. Acoust., Speech, Signal Processing,* vol. ASSP-34, no. 2, pp. 362–374.

[21] A. Levis *et al.*, "Challenges to control, A collective view," *IEEE Trans. Automat. Contr.,* vol. AC-32, pp. 274–285, 1987.

[22] R. Milner, *A Calculus of Communicating Systems* (Lect. Notes in Computer Science Vol. 92). New York: Springer-Verlag.

[23] ——, "Calculi for synchrony and asynchrony," *Theoret. Comput. Sci.,* vol. 25, no. 3, pp. 267–310.

[24] C. Ozveren and A. S. Willsky, "Aggregation and multi-level control in discrete event dynamic systems," Mass. Inst. Technol., Reps. LIDS-P-1902, LIDS-MIT; also in *Automatica,* 1989.

[25] C. Ozveren and A. S. Willsky, "Observability of discrete event dynamic systems," Mass. Inst. Technol., Rep. LIDS-P-1861; also in *IEEE Trans. Automat. Contr.,* to be published.

[26] A. Pnueli, "The temporal logic of programs," in *Proc. IEEE Symp. Foundations of Comput. Sci.,* Providence, RI.

[27] P. J. Ramadge, "Observability of discrete event systems," in *Proc. CDC,* Athens, Greece, 1986, pp. 1108–1112.

[28] P. J. Ramadge and W. M. Wonham, "Supervisory control of a class of discrete event processes," *SIAM J. Contr. Optimiz.,* vol. 25, no. 1, pp. 206–230, 1987.

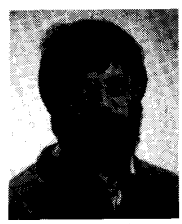
[29] ——, "On the supremal controllable sublanguage of a given language," *SIAM J. Contr. Optimiz.,* vol. 25, no. 3, pp. 637–659, 1987.

[30] J. C. Willems, "From time series to linear systems," *Automatica,* vol. 22, pp. 561–580, 1986.

Albert Benveniste (M'81–SM'89) was born in Paris, France, in 1949. He graduated from Ecole des Mines de Paris in 1971.

From 1971 to 1973 he has been with the Centre d'Automatique de l'Ecole des Mines, Fontainebleau. From 1974 to 1976, he has been with INRIA, Rocquencourt. In 1976, he joined IRISA, Rennes, where he holds an INRIA research position. After some work in probability theory Markov processes and Ergodic theory, for the "Thèse d'Etat" in 1975, his interest moved towards the area of applied mathematics (signal processing, identification and adaptive algorithms, speech and image coding, data communication systems, change detection). Since 1983, he has also been involved in the SIGNAL project under the head of Paul le Guernic. He is the author of numerous papers on signal processing, automatic control, and computer science. He is the coauthor of a book on adaptive algorithms with M. Métivier and P. Priouret.

Dr. Benveniste is Associate Editor for the IEEE TRANSACTIONS ON AUTOMATIC CONTROL, the *International Journal on Adaptive Control and Signal Processing,* and *Discrete Event Systems.* He is Chairman of the IFAC Committee on Theory for the triennium 90–93.



Paul Le Guernic was born in Bonneval, France, in 1950. He graduated from the Institut National des Sciences Appliquées, Rennes, France, in 1974.

Since 1978, he has been a member of IRISA, Rennes, with an INRIA research position. From 1974 to 1978, he worked in language theory and compiling techniques, which was the area of his "Thèse de troisième cycle" in 1976. Since 1978, he has mainly been concerned with parallel processing, and, since 1981, more precisely with real time systems. The Programming Environment for Real Time Applications group that he manages has defined the language SIGNAL. As a designer of the SIGNAL environment, he is also interested in symbolic manipulation tools.