

# A denotational theory of synchronous reactive systems

Albert Benveniste, Paul Le Guernic  
IRISA-INRIA, Campus de Beaulieu  
35042 RENNES CEDEX, FRANCE

Yves Sorel, Michel Sorine  
INRIA, Rocquencourt, BP 105  
78153 LE CHESNAY CEDEX, FRANCE

May 30, 1995

## Abstract

In this paper, systems which interact permanently with their environment are considered. Such systems are encountered, for instance, in real-time control or signal processing systems,  $C^3$ -systems, man-machine interfaces, to mention just a few. The design and implementation of such systems require a concurrent programming language which can be used to verify and synthesize the synchronization mechanisms, and to perform transformations of the concurrent source code to match a particular target architecture. Synchronous languages are convenient tools for such a purpose: they rely on the assumption that: 1/ internal actions of synchronous systems are instantaneous, and, 2/ communication with the environment is performed via instantaneous flashes involving some external stimuli. In this paper, we present a mathematical model of synchronous languages and illustrate its use on the SIGNAL language. This model is denotational, and encompasses both relational and functional styles of specification. It allows us to answer fundamental questions related to synchronous languages, such as “what are the basic constructions which should be provided by such languages?”

# 1 Introduction

*Reactive* systems<sup>1</sup> are considered in this paper. These are systems which interact permanently with their environment *at rates which may be driven by this environment*. Such systems are encountered, for instance, in real-time control or signal processing systems,  $C^3$ -systems, man-machine interfaces, to mention just a few. It is usually recognized that a reliable design of such systems should be supported by a *concurrent* programming style. On the other hand, the highly demanding nature of these applications forces to consider as well the requirement of highly efficient and reliable implementation, in both cases of sequential or distributed implementation. This requires powerful formal tools to prove equivalence of such different implementations. Hence fundamental studies on mathematical models of reactive systems are requested to provide the basis for the above mentioned tools.

We shall not discuss here the drawbacks and merits of current tools in programming reactive systems (finite state machines, Petri Nets, concurrent programming languages such as ADA or OCCAM); the interested reader is referred to the excellent discussion in [8] [9] on this subject. We shall merely concentrate on the discussion of the *synchronous approach* we follow in this paper.

## 1.1 The basic synchronicity hypotheses

While classical (i.e. asynchronous) concurrent languages do implicitly or explicitly refer to some external and universal time reference, the notion of “time” is completely different in synchronous reactive systems. To be more explicit, synchronous reactive systems differ from asynchronous ones in the following aspects:

1. *The internal mechanisms of the system:* every action (computation or internal communication) is instantaneous, i.e. has a zero duration;
2. *The communications with the external world:* the set of the possible input channels is fixed and known in advance, and the flows carried by these channels are specified through both
  - the values they carry
  - a total ordering of the “instants” at which these values are available at the external ports.

---

<sup>1</sup>this name was introduced in [19] [20], and extensively used in [8]

Of course, this last requirement is the fundamental feature which characterizes the way synchronous reactive systems communicate with the external world, compared to asynchronous ones. Let us illustrate this point using a simple example. Consider a reactive system with two inputs:

1. a data input carrying an ordered file of data named  $\mathbf{x}$ ,
2. an interrupt input port named  $\mathbf{s}$ .

Then, the specification of an input history according to the synchronous point of view must be of the form

$$\begin{array}{cccc} \mathbf{x}_1 & \mathbf{x}_2 & \mathbf{x}_3 & \perp \\ \perp & \mathbf{s}_2 & \perp & \mathbf{s}_4 \text{ etc...} \end{array}$$

(as usually,  $\perp$  denotes the absence of data) i.e. both the values and their global interleaving must be specified; the integer index  $n = 1, 2, \dots$  is used for this purpose. *The progression of this index  $n$  has to be considered as the proper notion of time flow in synchronous systems.* In other words, the essentially nondeterministic character of the communications with the external world in reactive systems is concentrated inside some (ignored) external mechanism which *decides* this global ordering. Hence, the advantage of the synchronous point of view is that *the nondeterminism of external communications is strictly concentrated on this mechanism, and it is not propagated inside the body of the system itself.*

A first consequence is that any function of, or any constraint on these input stimuli may be specified by mathematical recurrent equations. Another fundamental consequence is that the notion of time is *local to a given subsystem*: there is no universal time reference, as we shall see later when communications will be studied.

Theses are the fundamental reasons of the power of the synchronous approach. Among languages relying on this synchronicity assumption are the imperative language ESTEREL ([8] [18]), the declarative and functional language LUSTRE ([12] [11] [27]), and the declarative and relational language SIGNAL we discuss in this paper; related to the same formalism is also the approach of STATECHARTS in [19] [20].

## 1.2 On the semantics of SIGNAL

As argued before, SIGNAL must rely on a mathematical model; such a model and the language were developed simultaneously. In fact, two models of

different styles were introduced.

A didactic overview of the fundamental issues raised by SIGNAL may be found in [7]. Then, an *operational semantics* was given in [6] in terms of conditional rewriting rules à la Plotkin [28], and, in [5], it was shown how this operational semantics may be used to develop the SIGNAL compiler. On the other hand, a data-flow oriented introduction to SIGNAL is presented in [17], where the *Trace* model for SIGNAL, we further develop in the present paper, was introduced first.

The purpose of the present paper is different. We want to concentrate on fundamental aspects of the synchronous approach for reactive systems. This will be done by developing a new *denotational semantics*. This denotational model is related to the mathematical notion of “discrete-time dynamical system”, or “system of recurrent equations”, a discrete-time counterpart of differential systems. Such ideas are found in LUCID [3] [2] which is proposed as a model for data-flow languages. But the allowance of uncausality and particular ways to handle timing make LUCID not suitable as a model of reactive systems, as discussed in [11]. To our knowledge, the pioneering work relevant to the denotational style of semantics is the *Dynamic Network Processes* model introduced in [22] [21]. DNP’s are functions mapping input histories into output histories; their denotational semantics has been studied in detail in [15]. Kahn’s model has been used with suitable extensions and modifications in [27] to cope with the synchronicity assumption as a model for the LUSTRE language. Let us emphasize that these are non trivial modifications of the original DNP model, which was essentially free from any notion of synchronicity. Studies on synchronisation mechanisms within data-flow languages are presented in the excellent article [13], see also [23] for a data-flow model loosely related to SIGNAL.

Here, we shall introduce our approach via the very simple mathematical notion of *Multiple Clocked Recurrent Systems (MCRS)*, an immediate extension of systems of recurrent equations to properly reason about synchronization and timing. From this easily accepted starting point, the relevance of a “*Trace*” model of relational style will clearly follow. By “relational” we mean a model where behaviours are specified via constraints or relations rather than functions. This model may be considered as a suitable generalization of  $\omega$ -languages (or Büchi automata in the regular case [10][25][24]) in order to handle synchronisation and data types which are not finite alphabets; they also have a flavour of the theory of traces [1]. This *Trace*

model is first used to give the semantics of SIGNAL.

Then our purpose is to study fundamental questions related to synchronous reactive systems such as: what are the *basic constructions* a synchronous language should provide? To study this, we must refine our purely relational *Trace* model in order to be able to introduce *axioms* that our basic synchronisation tools (signals and clocks) should satisfy. This yields the  $\Omega$  model. The ideas behind the  $\Omega$  model are borrowed from the *probability and ergodic theories* ([26][14]). Its advantages are that

1. it allows us to axiomatize the notions of signal and clock,
2. it encompasses both relational and functional styles of specification,
3. parallelism is a built-in notion.

Using this  $\Omega$  model, we are able to prove that SIGNAL *provides the right primitives to achieve the maximum expressive power for synchronisation mechanisms in reactive systems.*

### 1.3 Organization of the paper

Section 2 is devoted to an informal introduction to MCRS and the SIGNAL language. The *Trace* model is presented in Section 3, and is used to formally define the SIGNAL language. Section 4 is the core of the paper: the  $\Omega$  model is introduced and studied, and the *filtering* and *multiplexing* are exhibited as fundamental primitives to build any synchronisation mechanism. Finally the  $\Omega$  model is used in Section 5 to study some properties of SIGNAL and show by the way that it possesses a maximum expressive power for synchronisation mechanisms.

## 2 Multiple Clocked Recurrent Systems and the SIGNAL language

### 2.1 An informal introduction to Multiple Clocked Recurrent Systems

Consider a discrete-time dynamical system described by a set of recurrent equations:

$$\begin{aligned} x_{n+1} &= f(x_n, y_n) \\ 0 &= g(x_n, y_n) \end{aligned} \tag{1}$$

where the variables  $x_n$  and  $y_n$  are both vector valued and  $n = 1, 2, 3, \dots$ . The  $x_n$ 's are internal variables, or states, and we may define some of the components in  $y_n$  to be input variables, and some as output variables and investigate the resulting input-output behaviour 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.

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 instant. The symbol used for absence is  $\perp$ . Therefore, an infinite time sequence of a variable (we shall refer to informally as a *signal* in this discussion) may look like

$$1, -4, \perp, \perp, 4, 2, \perp, \dots \quad (2)$$

which is interpreted as the signal being absent at the instants  $n = 3, 4, 7, \dots$  etc. Systems such as (1) where the signals are of the form (2) will be termed *Multiple Clocked Recurrent Systems (MCRS)*. The following questions are immediate from this definition:

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

$$\begin{aligned} &1, -4, \perp, \perp, 4, 2, \perp, \dots \\ &\perp, 1, \perp, -4, \perp, 4, \perp, 2, \perp, \dots \\ &1, -4, 4, 2, \dots \end{aligned}$$

Consider an “observer”<sup>2</sup> 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. 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. Hence clocks may 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.

---

<sup>2</sup>in the common sense, no mathematical definition is referred to here

**(2) How to interconnect two MCRS of the form (1)?** Consider the following two MCRS:

$$y_n = \text{if } x_n > 0 \text{ then } x_n \text{ else } \perp \quad (3)$$

and the usual addition on sequences, namely

$$z_n = y_n + u_n \quad (4)$$

In combining these MCRS, it is certainly preferable to match the successive occurrences  $y_1, y_2, \dots$  in (4) with the corresponding *present* occurrences in (3). But this is in contradiction with the immediate mathematical interpretation of the system of equations

$$\begin{aligned} y_n &= \text{if } x_n > 0 \text{ then } x_n \text{ else } \perp \\ z_n &= y_n + u_n \end{aligned}$$

which yields  $z_n = \perp + u_n$  whenever  $x_n \leq 0$ , and certainly does not match the usual interpretation of the addition of sequences. This kind of subtlety should convince the reader that the naive writing (1) for MCRS is inconvenient as either a specification technique or as a mathematical model. In the following section, we shall introduce informally the kernel of the SIGNAL language to specify MCRS. A more extensive discussion of such and related issues may be found in [7].

## 2.2 SIGNAL-kernel

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 [16]. 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  $\mathbf{x}$  is a signal, we denote by  $\{x_n\}_{n \geq 1}$  the sequence of its values when it is present. Signals that are always present simultaneously are said to have the same *clock*, so that clocks are equivalence classes of simultaneously present signals. Instructions of SIGNAL are intended to relate clocks as well as values of the various signals involved in a given system. We term a system of such relations *program*; programs may 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, identical names refer to identical signals. The

kernel-language SIGNAL possesses 6 instructions, the first of them being a generic one.

- (i)  $R(x_1, \dots, x_p)$
- (ii)  $y := x \$ x_0$
- (iii)  $y := x \text{ when } b$
- (iv)  $y := u \text{ default } v$
- (v)  $P \mid Q$
- (vi)  $P !! x_1, \dots, x_p$

Their intuitive meaning is as follows (for a formal definition, see the section 3):

- (i) direct extension of instantaneous relations into relations acting on signals:

$$R(x_1, \dots, x_p) \iff \forall n : R(x_{1_n}, \dots, x_{p_n}) \text{ holds}$$

where  $R(\dots)$  denotes a relation and the index  $n$  enumerates the instants at which the signals  $x_i$  are present. Examples are functions such as  $z := x+y$  ( $\forall n : z_n = x_n + y_n$ ) or statements such as  $(a \text{ and } b) \text{ or } c = \text{true}$  ( $\forall n : (a_n \text{ and } b_n) \text{ or } c_n = \text{true}$ ). 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**  $x, y, \dots$  which only forces the listed signals to have the same clock.

- (ii) shift register.

$$y := x \$ x_0 \iff \forall n > 1 : y_n = x_{n-1}, y_1 = x_0$$

Here 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 absent; the result is an event-based undersampling of signals. Here follows a table summarizing this instruction:

<b>b</b>	<i>true</i>	<i>false</i>	$\perp$
<b>x</b>			
<i>x</i>	<i>x</i>	$\perp$	$\perp$
$\perp$	$\perp$	$\perp$	$\perp$

(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:

<b>u</b>	<i>u</i>	$\perp$
<b>v</b>		
<i>v</i>	<i>u</i>	<i>v</i>
$\perp$	<i>u</i>	$\perp$

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:

$$\begin{aligned} y_n &= zy_n + a_n \\ zy_n &= y_{n-1}, \quad zy_1 = x_0 \end{aligned}$$

On the other hand, the program

```
(| y := x when x>0
 | z := y+u
 |)
```

yields

$$\text{if } x_n > 0 \text{ then } \begin{cases} y_n = x_n \\ z_n = y_n + u_n \end{cases} \\ \text{else } y_n = u_n = z_n = \perp$$

where  $(x_n)$  denotes the sequence of present values of  $\mathbf{x}$ . Hence the communication  $|$  causes  $\perp$  to be inserted whenever needed in the second system  $\mathbf{z}:=\mathbf{y}+\mathbf{u}$ . This is what we wanted for the example (3,4).

(vi) restriction to the listed set of signals: other signals are local to the considered program and therefore play no role in program communication.

A formal semantics of SIGNAL is presented in the section 3 using the *Trace* model.

### 3 The *Trace* model for MCRS and a semantics of SIGNAL

In this section, a mathematical model for MCRS is presented, and used to formally define SIGNAL. The reader is referred to Section 2 for the motivation of the following definitions.

#### 3.1 Histories, signals, clocks

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

$$\mathcal{D}_A = \bigcup_{a \in A} (\mathcal{D}_a \cup \{\perp\})$$

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

$$\mathcal{E}_A = A \rightarrow \mathcal{D}_A$$

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

**Traces.** Traces are infinite sequences of events. Let  $\mathbf{N}_+ = \{1, 2, \dots\}$  denote the set of integers, then the set of  $A$ -traces (or simply “traces”) is defined as

$$\Theta_A = \mathbf{N}_+ \rightarrow \mathcal{E}_A$$

**Compressions.** The *compression* of an  $A$ -trace  $T$  (deleting the silent events) 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  $T \downarrow$ .

**Histories and 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 *histories*. The set of all possible histories on  $A$  will be denoted by  $\Omega_A$ , so that we have<sup>3</sup>

$$\Omega_A = (\Theta_A)_{/\sim}$$

Elements of  $\Omega_A$  will be generically denoted by  $\omega_A$  or simply  $\omega$  when no confusion can occur. While the notion of trace refers to a particular environment (since the  $\perp$ 's are explicitly listed), the notion of history does not. Since

$$\Omega_A = [\mathbf{N}_+ \rightarrow (A \rightarrow \mathcal{D}_A)]_{/\sim}$$

any  $\omega_A \in \Omega_A$  may be written as

$$\omega_A = (\omega_a)_{a \in A} \tag{5}$$

and the  $\omega_a$ 's are termed *signals*. Hence a signal is a component of a history specified by selecting a particular port in the alphabet  $A$ . The notion of “signal” has been informally discussed in section 2.1-(1), where we motivated the definition of signals and histories as equivalence classes with respect to the relation  $\sim$ .

---

<sup>3.</sup> $_{/\sim}$  denotes here the quotient space by the relation  $\sim$

**Clocks.** Extend the domains  $\mathcal{D}_a$  with another distinguished value  $\top$ , intended to encode the status “present” regardless of any particular value. Consider the map  $chronos_{\mathcal{D}} \in \mathcal{D}_A \rightarrow \{\perp, \top\}$  defined by

$$chronos_{\mathcal{D}}(\perp) = \perp, \quad chronos_{\mathcal{D}}(x) = \top \text{ for } x \neq \perp$$

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

$$\begin{array}{ccc} & A & \\ \epsilon \swarrow & \xrightarrow{chronos_{\mathcal{D}}} & \searrow \\ \mathcal{D}_A & & \mathcal{D}_A \end{array} \quad \begin{array}{ccc} & & \\ & & chronos_{\mathcal{E}}(\epsilon) \\ & & \end{array}$$

Similarly, there is a unique map in  $\Theta_A \rightarrow \Theta_A$ , we denote by  $chronos_{\Theta}$ , making the following diagram commutative

$$\begin{array}{ccc} & \mathbf{N}_+ & \\ T \swarrow & \xrightarrow{chronos_{\mathcal{E}}} & \searrow \\ \mathcal{E}_A & & \mathcal{E}_A \end{array} \quad \begin{array}{ccc} & & \\ & & chronos_{\Theta}(T) \\ & & \end{array}$$

This map satisfies the condition  $T_1 \sim T_2 \Rightarrow chronos_{\Theta}(T_1) \sim chronos_{\Theta}(T_2)$ , so that it induces a map in  $\Omega_A \rightarrow \Omega_A$  we shall now denote by  $chronos$ : the *chronos* of a history is another history which summarizes the status {present/absent} of each of its signals (i.e. components).

Now, given  $\omega \in \Omega_A$  and  $a \in A$ , consider the signal of port  $a$  of the history  $chronos(\omega)$ : this signal summarizes the *relative* status present/absent of the signal  $\omega_a$  given the other signals involved in the history  $\omega$ . We shall call this signal the *clock* of  $\omega_a$ , or the clock of  $a$  for short when no confusion is likely to occur, and denote it by  $clock(\omega_a)$  or  $clock(a)$ .

### 3.2 MCRS

**Definition of MCRS.** A MCRS is simply a subset

$$\Omega \subset \Omega_A$$

of the set of all histories on  $A$ . In other words, we consider the dynamical system (1), or, better, a SIGNAL program, as a way to specify “legal” histories.

**Restricting MCRS.** Consider a subset  $A'$  of the alphabet  $A$ . The inclusion  $A' \subset A$  induces a projection from  $\mathcal{E}_A$  onto  $\mathcal{E}_{A'}$  we denote by  $\epsilon \longrightarrow \epsilon_{!!A'}$ . 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

$$\begin{array}{ccc} & \mathbf{N}_+ & \\ T \swarrow & & \searrow T_{!!A'} \\ \mathcal{E}_A & \xrightarrow{!!A'} & \mathcal{E}_{A'} \end{array}$$

uniquely defines the restriction  $T \longrightarrow T_{!!A'}$  on traces. Since  $T_1 \sim T_2 \Rightarrow (T_1)_{!!A'} \sim (T_2)_{!!A'}$  holds, a restriction on histories  $\omega \longrightarrow \omega_{!!A'}$  may be defined, which finally yields a restriction on MCRS we denote by

$$\Omega \longrightarrow \Omega_{!!A'}$$

This restriction maps the set of MCRS defined over the alphabet  $A$  onto the set of MCRS defined over the alphabet  $A'$ . The MCRS  $\Omega_{!!A'}$  is called the *restriction* of  $\Omega$  to (the subalphabet)  $A'$ : only the signals with ports in  $A'$  are visible from outside and may be used for MCRS communication we shall define next.

**MCRS communication.** Consider two MCRS  $\Omega_1, \Omega_2$  respectively defined over the alphabets  $A_1$  and  $A_2$ . Set  $A = A_1 \cup A_2$ . Then  $\Omega_1|_{\Omega_2}$  will denote the maximal<sup>4</sup> MCRS  $\Omega$  defined over the alphabet  $A$  satisfying the following conditions:

$$\begin{array}{l} \Omega_{!!A_1} \subseteq \Omega_1 \\ \Omega_{!!A_2} \subseteq \Omega_2 \end{array}$$

In other words, the communication constrains the signals in  $\Omega_1$  and  $\Omega_2$  of shared port to be identical (i.e. to be present simultaneously and then carry the same value). This is exactly what we wanted while discussing the example of eqns (3,4).

### 3.3 The definition of SIGNAL

According to the preceding section, in order to specify an MCRS over a given alphabet, we have to describe a subset of all histories that can be

---

<sup>4</sup>with respect to the order by inclusion  $\Omega' \subseteq \Omega$  defined on MCRS

built upon this alphabet. Since histories are defined as equivalence classes of traces with respect to the relation  $\sim$ , this may be done by *listing a family of constraints on the set of all traces* that can be built on this alphabet. The equivalence classes of the so specified traces are the specified histories. This is what we shall do next.

**Instruction (i):**  $R(x_1, \dots, x_p)$

$$\begin{aligned} \forall n \in \mathbf{N}_+, \forall i & : \mathbf{x}i_n \neq \perp \\ \forall n \in \mathbf{N}_+ & : R(\mathbf{x}1_n, \dots, \mathbf{x}p_n) \text{ holds} \end{aligned}$$

Here, the notation  $\mathbf{x}i_n$  denotes the value carried by the port with name  $\mathbf{x}i$  at the  $n$ -th instant of the considered trace. This notation will be further used in the sequel of this subsection.

**Instruction (ii):**  $y := x \$ x_0$

$$\begin{aligned} \forall n \in \mathbf{N}_+ & : \mathbf{x}_n \neq \perp \\ \forall n > 1 & : \mathbf{y}_n = \mathbf{x}_{n-1} \\ & \mathbf{y}_1 = \mathbf{x}_0 \end{aligned}$$

**Instruction (iii):**  $y := x \text{ when } b$

$$\forall n \in \mathbf{N}_+, \mathbf{y}_n = \begin{cases} \text{if } \mathbf{x}_n \neq \perp \text{ and } \mathbf{b}_n = \text{true} \text{ then } \mathbf{x}_n \\ \text{else } \perp \end{cases}$$

**Instruction (iv):**  $y := u \text{ default } v$

$$\forall n \in \mathbf{N}_+, \mathbf{y}_n = \begin{cases} \text{if } \mathbf{u}_n \neq \perp \text{ then } \mathbf{u}_n \\ \text{else if } \mathbf{u}_n = \perp \text{ and } \mathbf{v}_n \neq \perp \text{ then } \mathbf{v}_n \\ \text{else } \perp \end{cases}$$

**Instruction (v):**  $P \mid Q$

We already defined the operator  $\mid$  on MCRS.

**Instruction (vi):**  $P !! x_1, \dots, x_p$

We already defined the restriction of MCRS to a subset of ports.

### 3.4 Discussion

At this point the following question should be investigated: *did we propose in the SIGNAL language the right primitives to specify MCRS?* More specifically, we would like to prove that *no loss occurs in using SIGNAL instead of the general and abstract mathematical model of section 3.1 to specify constraints and relations concerning timing in MCRS.* This is the subject of the rest of the paper.

## 4 The $\Omega$ model for MCRS

### 4.1 Criticizing the *Trace* model

In the preceding section, we have introduced the *Trace* model for MCRS. Although simple, this model is not powerful enough to analyse the fundamentals of timing. To illustrate this claim, let us consider the following MCRS, that are specified using SIGNAL:

1.  $y := u+v$
2.  $y := x$  when  $b$
3.  $y := u$  default  $v$

Referring to Section 3, the corresponding MCRS are defined via constraints on the set of all possible joint behaviours of the signals involved in these instructions. This is a *relational* style of specification. Its advantage is to allow a very simple definition of the MCRS communication. However taking systematically a relational point of view is certainly restrictive as shown by the above examples: the signals on the righthand side may be certainly considered as inputs and  $y$  as output. Therefore we shall extend our *Trace* model to allow mixed relational/functional styles of specification.

Moreover our intuition is that the example 2. possesses the clocks of  $x$  and  $b$  as master clocks whereas the clock of  $y$  is entirely determined by the two master clocks and the values of the boolean signal  $b$ . A similar argument holds for the example 3.

Therefore what we would like to have is a mathematical model of MCRS with the following features:

- both relational as well as functional point of views should be encompassed.

- the notion of communication should be easily described.
- more fundamentally, we should be able to define clocks via a set of self-explanatory *axioms* and to derive from these axioms how new clocks may be created from given ones.

Using such a model, we would be able to answer the question we raised in the preceding section, namely *does SIGNAL provide the right primitives to specify general synchronisation mechanisms for MCRS?* The  $\Omega$  model we shall introduce next as a refinement of the *Trace* model has this as its objective.

## 4.2 The fundamentals of timing

### 4.2.1 Processes and information flows

Consider a MCRS  $\Omega$ . For  $\omega \in \Omega$ , denote by

$$T_1(\omega) \tag{6}$$

the unique trace such that (cf Section 3.1)  $\forall T \in \omega, T \downarrow = T_1(\omega)$ , i.e.  $T_1(\omega)$  is the unique trace representing  $\omega$  with no “silent” events. For two histories  $\omega, \omega' \in \Omega$ , we define the equivalence relation

$$\omega \approx_n \omega' \Leftrightarrow \forall m \leq n : [T_1(\omega)]_m = [T_1(\omega')]_m \tag{7}$$

When (7) holds, we say that  $\omega$  and  $\omega'$  possess *identical initial segments up to  $n$* . The equivalence relation  $\approx_n$  on  $\Omega$  defines a partition of  $\Omega$  we denote by  $\Pi_n$ . The family of partitions  $(\Pi_n)$  is ordered as follows: for  $m < n$ ,  $\Pi_n$  is *finer* than  $\Pi_m$ , written  $\Pi_m \leq \Pi_n$ , which means that every element of  $\Pi_m$  is a union of elements of  $\Pi_n$ . This yields the following definition where  $\mathbf{N}$  denotes the set of nonnegative integers:

**Definition 1** A **process** is a pair  $\{\Omega, (\Pi_n)_{n \in \mathbf{N}}\}$  or  $\{\Omega, \Pi\}$  for short, where

- $\Omega$  is a MCRS,
- for every  $n$ ,  $\Pi_n$  is a partition of  $\Omega$  into sets of histories of identical initial segments up to  $n$  (cf. (7)), and  $\Pi_0$  is the trivial partition  $\{\Omega, \emptyset\}$ .

The ordered family of partitions  $(\Pi_n)$  is called the **information flow** of the process.

$\Pi_n$  is to be interpreted as “the information available at time  $n$ .” Hence the refinement of the notion of process with respect to that of MCRS lies in the attention we pay to initial segments.

### 4.2.2 Clocks

The purpose of this subsection is to generalize the notion of clock we introduced in the *Trace* model. Recall that in this model, signals are *components* of histories and clocks summarize the relative status present/absent of these signals. But taking components is just a particular function defined on histories, hence we shall extend this notion by allowing more general functions to be considered. Such functions will have to satisfy special “causality” conditions as discussed in the next example.

**Example.** In this example we use the same conventions as in Section 3.3 to specify histories via constraints on traces. Consider the MCRS consisting of all possible single signals  $\omega$  of integer type. Select a threshold  $\lambda$  and consider the successive instants  $n = n_1, n_2, \dots$  such that  $\omega_n > \lambda$  holds. We define a new signal by setting “ $y_n = \text{if } n \in \{n_k\} \text{ then } \omega_n \text{ else } \perp$ ”, and the clock of this signal  $y$  is just defined by the sequence of instants  $\{n_k\}$ . To know whether a given instant  $n$  is a tick of this clock, it suffices to know the initial segment  $[\omega_1, \dots, \omega_n]$  of  $\omega$  up to the instant  $n$ . This is a sort of a causality property that will serve as a basis for our axiomatic model we shall present now.

In this example we illustrated how to create new clocks via (history-dependent) undersampling, but history-dependent *oversampling* is useful as well in specifying synchronisation in MCRS (cf. [7]). For instance, the set  $\mathbf{N}^k$  endowed with the lexicographic order is useful to represent  $k - 1$  nested loops that are fired at each instant. To allow for oversampling, we need to consider with some care what are the time index sets we want to handle.

Given two totally ordered sets  $\mathcal{T}$  and  $\mathcal{T}'$ , we shall write

$$\mathcal{T} \subseteq_{o.p.} \mathcal{T}' \tag{8}$$

to mean that  $\mathcal{T}$  is a subset of  $\mathcal{T}'$  and that the natural injection from  $\mathcal{T}$  into  $\mathcal{T}'$  is *order preserving*. Similarly

$$\mathcal{T}' \wedge_{o.p.} \mathcal{T}''$$

denotes the supremum of all  $\mathcal{T}$ 's satisfying  $\mathcal{T} \subseteq_{o.p.} \mathcal{T}'$  and  $\mathcal{T} \subseteq_{o.p.} \mathcal{T}''$ . If  $\mathcal{T}'$  and  $\mathcal{T}''$  possess a common upperbound for the relation  $\subseteq_{o.p.}$ , their supremum is well defined and is denoted by

$$\mathcal{T}' \vee_{o.p.} \mathcal{T}''$$

**Definition 2** A **time index set** is a denumerable, totally ordered set  $\mathcal{T}$  such that  $\mathbf{N} \subseteq_{o.p.} \mathcal{T}$ . Time index sets are generically denoted by  $\mathcal{T}$  and their elements by the letters  $s, t, u, v$ . The elements  $0$  and  $\infty$  of  $\mathbf{N}$  are assumed to be respectively the infimum and the supremum of  $\mathcal{T}$ .

Using the natural embedding of  $\mathbf{N}$  into  $\mathcal{T}$  allows us to write expressions such as  $s \leq n$  where  $s \in \mathcal{T}$  and  $n \in \mathbf{N}$ .

**Comment:** we consider that there exists some master time index set  $\mathbf{N}$ . The introduction of oversets of  $\mathbf{N}$  will allow to consider clocks that are more frequent than  $\mathbf{N}$ . To face the same need, Gonthier [18] uses the real line as a universal time index set, but only discrete subsets are effectively used. We prefer our approach since referring to a universal notion of time might be misleading in our context.

Clocks may be defined using three equivalent points of view: a set of instants, an increasing sequence of dates, an increasing counter. In any case, clocks are history-dependent, so that they will be defined as *functions of histories*. This is consistent with the discussion of Section 4.1. In the following definition, we are given a time index set  $\mathcal{T}$ .

**Definition 3** A **clock** is defined via the following equivalent points of view:

1. **Using sets.** A clock is specified by a subset  $\overline{H} \subseteq \Omega \times \mathcal{T}$  satisfying the property

$$\{(\omega, s) \in \overline{H}; s < n + 1, \omega' \approx_n \omega\} \Rightarrow \{(\omega', s) \in \overline{H}\} \quad (9)$$

We shall use the following notations

$$\overline{H}(\omega, \cdot) = \{s \in \mathcal{T} : (\omega, s) \in \overline{H}\} \quad (10)$$

$$\overline{H}(\cdot, s) = \{\omega \in \Omega : (\omega, s) \in \overline{H}\} \quad (11)$$

2. **Using dates.** A clock is a function

$$H : \Omega \times \mathbf{N} \rightarrow \mathcal{T}$$

satisfying the following properties

$$H(\omega, 0) = 0 \quad , \quad H(\omega, \infty) = \infty \quad (12)$$

$$m < n \text{ and } H(\omega, n) < \infty \Rightarrow H(\omega, m) < H(\omega, n) \quad (13)$$

$$H(\omega, n) < m + 1 \text{ and } \omega' \approx_m \omega \Rightarrow H(\omega', n) = H(\omega, n) \quad (14)$$

We shall often write  $H_n(\omega)$  instead of  $H(\omega, n)$ .

These two definitions are related to each other via the formulae

$$\begin{aligned} (\omega, s) \in \overline{H} &\Leftrightarrow \exists n : H_n(\omega) = s \\ H_n(\omega) &= \min \left\{ s : \#\{\overline{H}(\omega, \cdot) \cap [0, s]\} = n \right\} \end{aligned} \quad (15)$$

where  $\#\{\dots\}$  denotes cardinal. Counters were introduced for the same purpose in [12]. They may also be introduced here via the formula

$$\mu_s^H(\omega) = \#\{m \in \mathbf{N} : H_m(\omega) \leq s\}$$

### Comments

1. The notion of clock we introduced in Section 3 may be viewed as a particular case of the first point of view. Given  $\omega$  and  $a \in A$ , the clock of  $\omega_a$  may be represented by the set of indices  $n$  such that  $[T_\perp(\omega)_n](a) \neq \perp$  (cf. the notation (6) and the definition of traces in Section 3). The causality condition (9) is immediate in this case. Only undersampling was encountered in the *Trace* model.
2. The second point of view (using dates) will be more convenient than the first one in the sequel.
3. The conditions (9) or (14) axiomatize the causality property we discussed in the example of Section 4.1. In particular, (14) expresses that, to decide whether  $H_n \in [m, m+1)$ , it suffices to know the initial segments up to and including  $m$ . This reflects the fact that, while the “system” may live between  $m$  and  $m+1$ , it does not receive fresh information during this period.

**Time changes.** Suppose you have a clock, and you are interested in an infinite sequence of data which are present at each occurrence of this clock. Then, you would probably like to forget the original time reference, and prefer to work with the above mentioned clock *as if it were the time reference*. For this purpose, it is needed to define how processes are carried out through such time changes.

**Definition 4** *Let  $\{\Omega, (\Pi_n)\}$  be a process, and  $H$  a clock. Define*

$$\omega \approx_{H_n} \omega' \Leftrightarrow m \leq H_n(\omega) < m+1 \text{ and } \omega \approx_m \omega' \quad (16)$$

*This is an equivalence relation. Formula (16) defines  $(\Pi_{H_n})$  as the **information flow associated with the clock  $H$** .*

The fact that (16) defines an equivalence relation is due to the property (14). The information flow associated with a clock is the good way to encode the notion of initial segment in the case of time changes.

### 4.2.3 Signals

To encompass both relational and functional points of view, we shall generalize the notion of signal as introduced in Section 3. In this section, signals were introduced as being components of histories. But selecting a component is just a particular function. Hence we shall more generally define signals as being functions of histories that satisfy suitable causality conditions.

**Definition 5** *Let  $\{\Omega, \Pi\}$  be a process, and  $H$  a clock. A  $\{\Omega, \Pi, H\}$ -**signal** (or **signal of clock  $H$**  for short when no confusion is likely to occur) taking its values in a set  $\Xi$  is a function*

$$X : \Omega \times \mathbf{N}_+ \rightarrow \Xi, \quad \text{written } (\omega, n) \rightarrow X_n(\omega)$$

*satisfying the following property:*

$$\omega' \approx_{H_n} \omega \Rightarrow X_n(\omega') = X_n(\omega) \quad (17)$$

Definition 5 expresses the fact that  $X_n$  has to be considered as present and known at time  $H_n$ ;  $H_n(\omega) = \infty$  means that  $X_n(\omega)$  is never delivered.

**Comment:** The notion of signal introduced in the section 3 may be viewed as a particular case of the definition 5. Namely, for  $a \in A$ ,  $\omega_a$  as defined in (5) is equally well specified by the pair

$$\{\text{clock}(\omega_a), ((\omega_a)_{n_1}, (\omega_a)_{n_2}, \dots)\}$$

where  $n_k$  is the  $k$ -th tick of  $\text{clock}(\omega_a)$  and  $(\omega_a)_{n_k}$  is the value of the  $k$ -th present occurrence of  $\omega_a$ . Considering next  $\omega$  as a variable yields exactly a particular case of definition 5 since the condition (17) is immediate.

## 4.3 The algebra of clocks

Throughout this section, we are given a fixed process  $\{\Omega, \Pi\}$ , and all clocks we shall consider are defined on this process. The aim of this section will be to introduce a “clock algebra”: writing relations within this algebra will be the convenient way to specify constraints on clocks. For this purpose we

shall introduce a partial order on the set of the clocks, and we shall introduce two useful primitives on this set, namely the *filtering* and the *multiplexing*. And we shall finally prove that *these primitives allow us to build any clock in finitely many steps*. This will be the first major result of our paper. Throughout this section, we shall use the notation

$$\mathcal{T}(H)$$

to refer to the time index set where the clock  $H$  takes its values.

### 4.3.1 A partial order on the set of the clocks

Given two clocks  $H$  and  $K$  such that  $\mathcal{T}(K) \subseteq_{o.p.} \mathcal{T}(H)$  holds, we define (cf. (10))

$$K \subseteq H \Leftrightarrow \forall \omega : \overline{K}(\omega, \cdot) \subseteq \overline{H}(\omega, \cdot) \quad (18)$$

In other words,  $K \subseteq H$  means that the set of occurrences of  $K$  is included in the set of the occurrences of  $H$  *whatever the history  $\omega$  is*.

**Warning:** since the intersection of two time index sets is a time index set, any two clocks  $H$  and  $K$  possess an infimum. Unfortunately, the supremum of two time index sets is generally *not* a time index set, so that the supremum of two arbitrary clocks is not defined in general. The operations of infimum and supremum (when the latter is properly defined) will be respectively denoted by

$$K \wedge H, K \vee H \quad (19)$$

### 4.3.2 The filtering

**Lemma 1** *Let  $H$  be a clock and  $\mathcal{T}$  its time index set, and let  $B$  be a boolean signal with clock  $H$ . The formula*

$$\overline{K} = \{(\omega, s) \in \Omega \times \mathcal{T} : \exists n \text{ such that } H_n(\omega) = s \text{ and } B_n(\omega) = \text{true}\}$$

*defines a new  $\mathcal{T}$ -valued clock we shall denote by*

$$K = H \downarrow B \quad (20)$$

*and is referred to as the clock obtained by **filtering**  $H$  by  $B$ .*

**Proof** that  $K$  is a clock: easy, left to the reader.

The meaning of the filtering is the following:  $H \downarrow B$  extracts from  $H$  the instants where the boolean  $B$  is true. Conversely, we have the following result:

**Lemma 2** *If  $K \subseteq H$ , then we have  $K = H \downarrow B$  where  $B$  is given by*

$$B_n(\omega) = \begin{cases} \text{true} & \text{if } H_n(\omega) \in \overline{K}(\omega, \cdot) \\ \text{false} & \text{otherwise} \end{cases}$$

The proof is elementary, and is left to the reader. The filtering is a primitive instruction of the languages LUSTRE and SIGNAL (the **when**), and may be built in ESTEREL.

### 4.3.3 The multiplexing

No tool is usually provided in real time oriented languages to allow over-sampling at data dependent rates. Our purpose is now to investigate theoretically the difficulties behind this notion within our  $\Omega$  model.

We are given a clock  $H$  taking values in a time index set  $\mathcal{T}$ . Let  $C$  be an integer valued (nonnegative) signal with clock  $H$ , such that  $0 \leq C_n < \infty$  for  $n$  finite, and  $C_\infty = 0$ . Set

$$\mathcal{T}' = \mathcal{T} \times \mathbf{N}$$

endowed with the *lexicographic order* defined by

$$[s, k] < [s', k'] \Leftrightarrow s < s' \text{ or } \{s = s' \text{ and } k < k'\}$$

Note that  $\mathcal{T}$  is naturally identified with the subset  $\mathcal{T} \times \{0\}$  of  $\mathcal{T}'$ , we shall often use this embedding in the sequel.

**Definition 6** *The multiplexing of the clock  $H$  by the signal  $C$  is denoted by*

$$K = H \uparrow C$$

*and is the  $\mathcal{T}'$ -valued clock  $K$  defined as follows:*

$$(\omega, [s, k]) \in \overline{K} \Leftrightarrow \{\exists m : H_m(\omega) = s \text{ and } 0 \leq k \leq C_m(\omega)\} \quad (21)$$

*where  $C_0(\omega) = 0$  by convention.*

Figure 1: The multiplexing

The figure 1 depicts this procedure. The  $\uparrow$ 's denote an increase by 1 of the second component of  $H \uparrow C$ , whereas the  $\downarrow$ 's replace at the same time  $H_m$  by  $H_{m+1}$  and  $k = C_m$  by  $k = 0$ .  $C_n$  specifies how many additional instants have to be inserted between the  $n$ -th and the  $(n + 1)$ -st instants of  $H$ . To justify the above definition, we have to prove the following result:

**Theorem 1**  $K = H \uparrow C$  is a  $\{\Omega, \Pi\}$ -clock.

**Proof:** Using (21) we have

$$(\omega, [s, k]) \in \overline{K} \Rightarrow (\omega, s) \in \overline{H}$$

so that

$$\left. \begin{array}{l} (\omega, [s, k]) \in \overline{K} \\ [s, k] < n + 1 \\ \omega' \approx_n \omega \end{array} \right\} \Rightarrow (\omega', s) \in \overline{H} \quad (22)$$

since  $[s, k] < n + 1 \Rightarrow s < n + 1$  by definition of the lexicographic order. Then take  $m$  as in (18), since  $C$  is a signal of clock  $H$ , we have

$$\omega' \approx_n \omega \Rightarrow C_m(\omega') = C_m(\omega) \quad (23)$$

Finally, (22) and (23) together prove the theorem.

The next theorem is the fundamental result of this paper. It expresses the fact that *the filtering and multiplexing are the right primitives to construct any clock*.

**Theorem 2** Let  $H$  be a  $\mathcal{T}$ -valued clock, where

$$\mathcal{T} = \mathbf{N}^L, 0 < L < \infty$$

is endowed with the lexicographic order. Then  $H$  may be decomposed as follows:

$$\begin{aligned}
H^0 &= Id \\
H^1 &= H^0 \downarrow B^1 \\
\forall l > 0 : H^{l+1} &= (H^l \uparrow C^l) \downarrow B^l \\
H &= H^L
\end{aligned} \tag{24}$$

where  $B^l$  and  $C^l$  respectively are boolean and nonnegative integer signals<sup>5</sup>, and  $Id$  denotes the clock “Identity” defined by  $Id_n(\omega) = n$ . Furthermore, among all possible decompositions, there is a **minimal** one, we denote by  $H_*^0, \dots, H_*^L$ , such that

$$\forall l \leq L : H_*^l \subseteq H^l \tag{25}$$

for any decomposition (24).

**Proof:** Denote by  $proj_l$  the projection of  $\mathcal{T}$  onto  $\mathbf{N}^l$  obtained by discarding the  $L - l$  last coordinates of  $t$  to get  $proj_l(t)$ . Using the notation (9) we define the clock  $H^l$  by

$$\overline{H^l} = proj_l(\overline{H}) \tag{26}$$

To prove the theorem, it suffices

1. to verify that  $\overline{H^l}$  satisfies the condition (9),
2. to prove that  $H^{l+1}$  and  $H^l$  are related via (24).

The first assertion is proved by induction over  $l$ . Write for short

$$t(l) = proj_l(t)$$

and decompose  $t(l+1) = [t(l), k]$ . Consider  $n$  such that  $n \leq t(l) < n+1$ . The definition of the lexicographic order implies that  $n \leq t(l+1) < n+1$  also holds. The formula

$$\overline{H^l} = proj_l(\overline{H^{l+1}})$$

implies in this case that<sup>6</sup>

$$\begin{aligned}
\overline{H^l}(\cdot, t(l)) &= proj_l(\overline{H^{l+1}}(\cdot, t(l))) \\
&= \bigcup_k \overline{H^l}(\cdot, [t(l), k])
\end{aligned}$$

<sup>5</sup>of suitable clocks so that the corresponding formulae make sense

<sup>6</sup>we use the notation (11)

so that

$$\{(\omega, t(l)) \in \overline{H^l} \text{ and } \omega' \approx_n \omega\} \Rightarrow \{(\omega', t(l)) \in \overline{H^l}\}$$

which proves the first assertion by induction.

To prove the second assertion, select  $m$  as in (21) and set

$$C_m^l(\omega) = \max \{k \geq 0 : [t(l), k] \in \overline{H^{l+1}}(\omega, \cdot)\}$$

Then apply Lemma 1 to the clocks  $H^{l+1} \subseteq H^l \uparrow C^l$  to get  $B^l$ .

To prove (25) we remark that the formula (26) yields the desired decomposition. This finishes the proof of Theorem 1.

### Comments.

1. The most general time index set we may expect is any denumerable ordinal  $\mathcal{T}$ . But, if  $t$  denotes a limit ordinal in  $\mathcal{T}$ , there exists an increasing sequence in  $\mathcal{T}$  converging to  $t$ . Hence only finitely many such limit ordinals may exist in  $\mathcal{T}$ , otherwise  $\mathcal{T}$  would contain  $\mathbf{N}^{\mathbf{N}}$  with lexicographic order, but this latter ordinal is isomorphic to  $\aleph_o$  which is not denumerable. Finally  $\mathcal{T}$  must be a subset of  $\mathbf{N}^L$  for some finite  $L$ , so that Theorem 2 is the most general one may expect.
2. This theorem states that assuming that every clock is defined in terms of the *most frequent* one is incorrect from the mathematical point of view: both multiplexing and filtering should be used in general.
3. The combined use of the operations  $\uparrow$  and  $\vee$  on clocks may cause difficulties, as the following example shows. Consider two different nonnegative integer signals  $C$  and  $C'$  of clock  $Id$ . Should we consider that the two clocks  $Id \uparrow C$  and  $Id \uparrow C'$  take their values

(a) in the *same*, or

(b) in *different*

copies of the set  $\mathbf{N} \times \mathbf{N}$ ? In the first case, the supremum  $(Id \uparrow C) \vee (Id \uparrow C')$  does exist, while it does not in the second case since no total order is defined on the union of these two different copies.

Although simpler, the first choice is not very convenient, for it would result in very strange situations. Take for instance  $C \equiv 1$  and  $C' \equiv 3$ : one may expect that this should correspond to increasing the sampling by a rate 2 and 4 respectively. Unfortunately this is not what we get by applying the definition of multiplexing, which yields instead

ticks of $Id$ :	•				•				•			
ticks of $Id \uparrow C$ :	•	•			•	•			•	•		
ticks of $Id \uparrow C'$ :	•	•	•	•	•	•	•	•	•	•	•	•

Finally, the most reasonable choice is the second one, namely to always assume that *different signals  $C$  create clocks with values in different time index sets*. This should be kept in mind in the sequel.

4. The main result of this section is that **the algebra of the clocks of a given process is equipped with the operations  $\wedge$  and  $\vee$  in a natural way, and that the filtering and the multiplexing are the convenient constructions to build any clock**. Obviously (as it has been pointed out in [27]), these two constructions may be combined into a single one provided that in Definition 6,  $C_n$  be interpreted as the amount of instants inside the semi-closed interval  $[H_n, H_{n+1})$  instead of the open one  $(H_n, H_{n+1})$  as we have done. But we preferred to keep this distinction since only the multiplexing may create problems.

#### 4.4 Isomorphisms and process algebra

Notice that, given two processes  $P = \{\Omega, \Pi\}$  and  $P' = \{\Omega', \Pi'\}$ , their *communication*  $P|P'$  is well defined: just take the MCRS  $\Omega|\Omega'$  and consider its information flow according to Definition 1. The same holds for the restriction  $P!!A$  which is built over  $\Omega!!A$ . Hence the set of processes equipped with the communication and restriction will be called the *process algebra*. We state now a notion of *isomorphism* within the process algebra.

**Definition 7** *Two processes  $P = \{\Omega, \Pi\}$  and  $P' = \{\Omega', \Pi'\}$  are said to be isomorphic, written*

$$P \cong P'$$

*if there exists a bijection  $\Phi : \Omega \rightarrow \Omega'$  such that  $\Phi(\Pi_n) = \Pi'_n$*

Clearly, this notion of isomorphism is a congruence, namely

$$P \cong P' \text{ and } Q \cong Q' \Rightarrow P|Q \cong P'|Q'$$

A natural notion of morphism may be defined as well: the map  $\Phi$  introduced in the above definition is a *morphism* from  $P$  into  $P'$  if  $\Phi^{-1}(\Pi'_n) \leq \Pi_n$ , where  $\Phi^{-1}$  denotes the inverse map of  $\Phi$  (which acts on subsets of  $\Omega'$ ), and  $\leq$  is the ordering on partitions we introduced just before the definition 1. But we shall not discuss the notion of morphism any further.

## 4.5 Processes revisited: a more abstract definition

In Definition 4 we introduced time changes. Referring to this definition we may wish to set  $\tilde{\Pi}_n = \Pi_{H_n}$  and consider objects such as  $\{\Omega, \tilde{\Pi}\}$ . Unfortunately, such objects are *not* covered by Definition 1 since the elements of the partition  $\tilde{\Pi}_n$  are not initial segments of  $\Omega$  of length  $n$ .

However it is not until the beginning of subsection 4.4 that we used the fact that the  $\Omega$ 's are MCERS and the  $\Pi$ 's associated information flow of initial segments. In fact this property has only been explicitly used in defining the process communication  $P|P'$ . In all other statements and proofs the only properties we really needed on the objects  $\Omega, \Pi_n, \approx_n$  were the following:

- $\Omega$  is a set.
- $(\Pi_n)$  is an ordered family of partitions of  $\Omega$  and  $\omega \approx_n \omega'$  if by definition  $\omega$  and  $\omega'$  belong to the same element of the partition  $\Pi_n$ .

Let us state this more precisely.

**Definition 1 revisited** *A process is a pair  $\{\Omega, (\Pi_n)_{n \in \mathbb{N}}\}$  or  $\{\Omega, \Pi\}$  for short, where*

- $\Omega$  is a set
- for every  $n$ ,  $\Pi_n$  is a partition of  $\Omega$  and we write  $\omega \approx_n \omega'$  to mean that  $\omega$  and  $\omega'$  belong to the same element of the partition  $\Pi_n$ ,
- the family of partitions  $(\Pi_n)$  is ordered by refinement, i.e. for  $m \leq n$ , each element of  $\Pi_m$  is a union of elements of  $\Pi_n$ ; moreover,  $\Pi_0 = \{\Omega, \emptyset\}$ .

*The ordered family of partitions  $(\Pi_n)$  is called the **information flow** of the process.*

Everything in this section 4 carries out to this more abstract notion of process, *except the definition of process communication* for which the definition we gave explicitly used the fact that  $\Omega$  is a MCERS. In particular, we may use the results on the algebra of clocks for the (time changed) process  $\{\Omega, \tilde{\Pi}\}$  we introduced at the beginning of this subsection, and we may also use the notion of process isomorphism. *This generalization will be required in Section 5 where some properties of SIGNAL are studied.*

In fact, the whole  $\Omega$  model might have been developed entirely based on this abstract definition, including the notion of process communication. This makes the whole theory harder to follow so that we preferred the presentation of this paper. The reader interested in the abstract version of the  $\Omega$  model is referred to [4] [5].

#### 4.6 Discussion

We have introduced the  $\Omega$  model as a refinement of the *Trace* model. We first equipped the notion of MCRS with the structure of initial segments to derive the notion of *process*. Then we built on this new notion a denotational theory which encompasses both relational and functional styles of specification. Finally we have shown how this theory may perfectly fit a more abstract notion of process that covers in particular the use of time changes.

We used this model to study the algebra of clocks and shown that the *filtering* and *multiplexing* are convenient primitives to build any synchronisation mechanism. Unfortunately it appeared that the multiplexing as such causes difficulties to occur since the supremum of two clocks obtained via multiplexing is generally not properly defined. The  $\Omega$  model will be used in the next section to investigate the properties of SIGNAL.

### 5 Properties of SIGNAL

In this section, we show that SIGNAL satisfies the following properties:

1. The semantics of any program may be stated using a process which possesses *Id* as the most frequent<sup>7</sup> clock. Consequently, no multiplexing is involved in such semantics and we *do not encounter the problems that may be caused by the simultaneous use of  $\vee$  and  $\uparrow$  operators on clocks* (cf. the warning of subsection 4.3.1 and the comment 3. following the proof of theorem 2).
2. However SIGNAL allows to *simulate the multiplexing* in a sense we shall formalize. Hence this ensures that SIGNAL has the *maximum descriptive power to specify synchronisation mechanisms* in synchronous reactive systems.
3. The  $\Omega$  model may be used to define different semantics of a SIGNAL program, from purely relational to purely functional ones, and these

---

<sup>7</sup>or maximal in the sense of the ordering on the clock algebra

different semantics are shown to be *bisimulation equivalent* in a sense we shall make precise.

Because of point 3, we shall indicate explicitly whether we consider the *Trace*-semantics of a SIGNAL program, or one of its  $\Omega$ -semantics. In handling SIGNAL programs and their semantics, we shall use the following notations: for each signal  $\mathbf{X}$  involved in a SIGNAL program, we denote by  $X$  and  $H(X)$  the corresponding signal and its clock in the considered semantics.

### 5.1 SIGNAL does not use the multiplexing as such

**Theorem 3** *The Trace-semantics of any SIGNAL program may be stated without the use of multiplexing.*

**Proof:** this is an immediate consequence of the two following facts:

1. the *Trace*-semantics of SIGNAL has been given in terms of the *Trace* model of Section 3,
2. no oversampling of clocks is possible within the *Trace* model, cf. the comment 1. following definition 3.

### 5.2 SIGNAL allows to “simulate” the multiplexing

The undefined notations may be found in Section 4.3.3. Consider the clock

$$H = Id \upharpoonright C$$

We may write

$$H_k = [N_k, M_k] \tag{27}$$

where<sup>8</sup>

$$\begin{aligned} N_k &= \text{if } (M_{k-1} = C_{N_{k-1}}) \text{ then } N_{k-1} + 1 \text{ else } N_{k-1} \\ M_k &= \text{if } (M_{k-1} = C_{N_{k-1}}) \text{ then } 0 \text{ else } M_{k-1} + 1 \end{aligned} \tag{28}$$

We shall translate these formulae into a SIGNAL program. The current instant is  $k$ : it will be handled implicitly.

However we also need to handle the signal  $C_{N_k}$ : this signal may be produced by the following program:

---

<sup>8</sup>these formulae are an immediate writing of the figure 1

```
(| CURRENT_C := C default LAST_C
 | LAST_C := CURRENT_C $ 0
 | synchro CURRENT_C, M, N
 |)
```

The last instruction specifies that the three mentioned signals must have the same clock. Then, `CURRENT_C` carries the most recent value of `C`, and  $C_{n_k}$  is represented by the signal `CURRENT_C`.

The boolean signal ( $m_{k-1} = C_{n_{k-1}}$ ) is also needed. The input signal `C` is received the instant following a *true* occurrence of this boolean (this is expressed by the last `synchro` instruction). The corresponding program is

```
(| DOWN_NEXT_TIME := (M = CURRENT_C)
 | DOWN := DOWN_NEXT_TIME $ true
 | synchro C, true when DOWN
 |)
```

Then it remains to encode the two equations (28):

```
((| N := (ZN+1 when DOWN) default ZN
 | ZN := N $ 0
 |)
(| M := (0 when DOWN) default ZM+1
 | ZM := M $ 0
 |)
|)
```

This gives finally the program<sup>9</sup>

```
MUX { ? C ! N,M } % ? list of inputs, ! list of outputs %
=
(|(| CURRENT_C := C default LAST_C
 | LAST_C := CURRENT_C $ 0
 | synchro CURRENT_C, M, N
 |)
(| DOWN_NEXT_TIME := (M = CURRENT_C)
 | DOWN := DOWN_NEXT_TIME $ true
 | synchro C, true when DOWN
```

---

<sup>9</sup>a much more concise program exhibiting a multiplexing mechanism has been presented in [7], it is based on a decreasing counter; the present form is useful for our theoretical purpose

```

|)
|(|(| N := (ZN+1 when DOWN) default ZN
  | ZN := N $ 0
  |)
|(| M := (0 when DOWN) default ZM+1
  | ZM := M $ 0
  |)
|)
|) !! C,M,N          % visible ports %

```

For each of the three modules we have introduced, it is straightforward although tedious to verify using the *Trace*-semantics of Section 3.3 that it implements the desired formulae, namely (27) and (28).

Next, consider the following two processes:<sup>10</sup>

$$\begin{aligned}
 \text{PRIMITIVE\_MUX} &= \{\Omega_{\mathbf{C}}, \Pi_{\mathbf{C}}\}, \text{ where} & (29) \\
 \mathbf{C} &\text{ is the alphabet } \{\mathbf{C}\} \\
 \Omega_{\mathbf{C}} &\text{ is the set of all possible histories of } \mathbf{C}
 \end{aligned}$$

and  $\Pi_{\mathbf{C}}$  is the associated information flow, and

$$\begin{aligned}
 \text{MUX} &= \{\Omega, \Pi\}, \text{ where} & (30) \\
 \Omega &\text{ is the } \textit{Trace}\text{-semantics of program MUX, cf Section 3.3}
 \end{aligned}$$

and  $\Pi$  is the associated information flow. According to (5), each  $\omega \in \Omega$  is of the form

$$\omega = (\omega_{\mathbf{C}}, \omega_{\mathbf{N}}, \omega_{\mathbf{M}})$$

and we denote by  $\Phi$  the first projection:

$$\Phi : \omega \in \Omega \longrightarrow \omega_{\mathbf{C}} \in \Omega_{\mathbf{C}}$$

On the other hand, denote by  $H$  the clock of  $C$  in  $\text{MUX}$ , and consider the time-changed information flow (cf. Definition 4)

$$\tilde{\Pi}_k \triangleq \Pi_{H_k}$$

on  $\text{MUX}$ . Then we have the following theorem where the abstract notion of process as in Definition 1 *revisited* is used:

---

<sup>10</sup>cf. Section 3.1 and 4.2 for undefined notations

**Theorem 4** 1. We have

$$\left. \begin{array}{l} \omega \approx_{H_n} \omega' \\ H_n(\omega) \leq k < H_{n+1}(\omega) \end{array} \right\} \Rightarrow \omega \approx_k \omega'$$

2. The map  $\Phi$  is an isomorphism from the process  $\{\Omega, \tilde{\Pi}_n\}$  onto the process  $\{\Omega_{\mathbf{C}}, (\Pi_{\mathbf{C}})_n\}$ , and the image by  $\Phi$  of the  $\mathbf{N}^2$ -valued signal  $[N, M]$  is the clock  $Id \uparrow C$ .

**Comment:** the first statement expresses that no fresh information is received by  $MUX$  between two successive occurrences of  $\mathbf{C}$ , so that no loss occurs by replacing the original information flow  $(\Pi_n)$  by the time changed one  $(\tilde{\Pi}_n)$ . And the second statement gives a precise meaning to what we mean by “simulating the multiplexing”.

**Proof:** For

$$H_n(\omega) \leq k < H_{n+1}(\omega) \quad (31)$$

it is easily checked on the *Trace*-semantics of  $MUX$  that this process evolves as follows:

$$N_{k+1}(\omega) = N_k(\omega), \quad M_{k+1}(\omega) = M_k(\omega) + 1$$

which proves statement 1 since the condition (31) only depends on the initial segment of length  $H_n$ .

That  $\Phi$  is a bijection is a consequence of the fact that the behaviour of the process  $MUX$  is entirely determined by its input  $C$ . Finally, that  $[N, M]$  is mapped onto  $Id \uparrow C$  is an immediate consequence of the fact that the program  $MUX$  implements the formulae (27)(28).

**Discussion:** The reader may have found what is the deep reason for  $SIGNAL$  to be able to simulate the multiplexing. The key tool is in fact process communication. The multiplexing is rebuilt within the program  $MUX$  according the following two pieces:

1. The first piece is the two following instructions

```
(| CURRENT_C := C default LAST_C
 | LAST_C := CURRENT_C $ 0
 |)
```

The communication causes the two signals `LAST_C` and `CURRENT_C` to have the same clock, and the first instruction asserts that `C` is less frequent than the two other signals. This program causes a (non determinate) amount of  $\perp$ 's to be inserted between successive occurrences of `C`. More generally, the effect of the communication as defined in Section 3 may also be expressed on the “compressed” traces (i.e. traces with no silent events): in  $P|Q$  the traces of  $P$  and  $Q$  are “expanded” (i.e. silent events are inserted) to allow for signals of shared ports to be identical. This expansion mechanism is a sort of “weak” multiplexing, i.e. a multiplexing which is not determined entirely by  $P$ , but needs a communication with another process in order to occur.

2. The second piece is the instruction

```
(| synchro C, true when DOWN
  |)
```

Since `DOWN` is a function of `C`, this instruction specifies the amount of inserted silent events between successive occurrences of `C` as a function of `C` itself. This fixes the “weak” multiplexing created by the communication and makes it a deterministic operator.

### 5.3 From Trace-*semantics* to $\Omega$ -*semantics* of SIGNAL programs

Theorem 4 states that at least two different semantics may be of interest for the program `MUX {?C !N,M}`, and that they are related via a time change followed by the isomorphism  $\Phi$ . We shall show that this situation may be generalized.

**Example:** a semantics of the instruction `Y := X when B` may be given in the two following ways:

1. Its *Trace-*semantics**, namely

$$P_1 = \{\Omega, \Pi\}$$

where  $\Omega$  is the *Trace-*semantics** built according to the rules of Section 3.3 and  $\Pi$  is the associated flow of initial segments. This is a purely relational style of semantics.

2. A new semantics, namely

$$P_2 = \left\{ \Omega_{\{\mathbf{X}, \mathbf{B}\}}, \Pi_{\{\mathbf{X}, \mathbf{B}\}} \right\}$$

where  $\Omega_{\{\mathbf{X}, \mathbf{B}\}}$  is the set of all histories on the alphabet  $\{\mathbf{X}, \mathbf{B}\}$ , as defined in Section 3.3 and  $\Pi_{\{\mathbf{X}, \mathbf{B}\}}$  is the associated flow of initial segments, whereas  $Y$  and its clock  $H(Y)$  are defined by

$$\begin{aligned} H(Y) &= H(X) \wedge (H(B) \downarrow B) \\ H(Y)_n(\omega) &= H(X)_m(\omega) \Rightarrow Y_n(\omega) = X_m(\omega) \end{aligned} \quad (32)$$

This is a purely functional style of semantics as wished at the beginning of Section 4.

In either case 1 or 2, however, a triple  $\{Y, X, B\}$  of signals were defined on a process  $\{\Omega, \Pi\}$ , which satisfied the relations (32). This is a situation similar to that of Section 5.2, and we show next that both examples are particular cases of a general result.

### 5.3.1 Determinism

Consider a SIGNAL program  $P$  and partition the set of its signals as  $\{U_1, \dots, U_p; Y_1, \dots, Y_q\}$ . Denote by  $\Omega$  its *Trace*-semantics. We consider also the associated process  $P = \{\Omega, \Pi\}$  where  $\Pi$  is the information flow of initial segments of  $\Omega$ .

The information on  $P$  an observer may learn by having access to  $U_1, \dots, U_p$  only is represented by an information flow we denote by  $\Pi^U$  and call the *information flow generated by the  $U_i$ 's*. This information flow may be constructed as follows. Consider the family of all information flows on  $\Omega$  making each of the  $U_i$  ( $i = 1, \dots, p$ ) to be a signal of clock  $H(U_i)$ . This set is not empty since it contains  $\Pi$ . Referring to the order on information flows defined by  $\Pi' \leq \Pi''$  if by definition  $\forall n, \Pi'_n \leq \Pi''_n$ , this set is stable by finite infimum. Hence a minimal information flow does exist within this set: this is  $\Pi^U$ . The information flow  $\Pi^U$  is characterized by the following property: for  $(\omega, k)$  define the index  $n_i$  via

$$H(U_i)_{n_i}(\omega) \leq k < H(U_i)_{n_i+1}(\omega)$$

then, we have

$$\omega \approx_{\Pi_k^U} \omega' \quad (33)$$

$$\begin{aligned} & \Leftrightarrow \\ \forall i = 1, \dots, p; n \leq n_i : & \left\{ \begin{array}{l} H(U_i)_n(\omega') = H(U_i)_n(\omega) \\ (U_i)_n(\omega') = (U_i)_n(\omega) \end{array} \right. \end{aligned}$$

This formula expresses that  $\Pi^U$  is entirely known when the  $U_i$ 's are observed.

**Definition 8** *The program  $P$  is said to be **deterministic w.r.t.**  $U_1, \dots, U_p$  if*

$$\Pi_n^U = \Pi_n \quad (34)$$

*holds (no information is lost by observing only the  $U_i$ 's).*

It turns out that the reasoning of Section 5.2 may be borrowed here. Set

$$H = H(U_1) \vee \dots \vee H(U_p)$$

and write  $\tilde{\Pi}_n = \Pi_{H_n}$ . Then the following theorem holds:

**Theorem 5** *If  $P$  is deterministic w.r.t.  $U_1, \dots, U_p$  then*

1. *no fresh information is received between  $H_n$  and  $H_{n+1}$ :*

$$\left. \begin{array}{l} \omega \approx_{H_n} \omega' \\ H_n(\omega) \leq k < H_{n+1}(\omega) \end{array} \right\} \Rightarrow \omega \approx_k \omega' \quad (35)$$

2. *writing  $\omega = (\omega_{U_1}, \dots, \omega_{U_p}; \omega_{Y_1}, \dots, \omega_{Y_q})$ , the map*

$$\Phi : \omega \longrightarrow (\omega_{U_1}, \dots, \omega_{U_p})$$

*is an isomorphism from  $\{\Omega, \tilde{\Pi}\}$  onto  $\{\Omega_{!!U}, \Pi_{!!U}\}$  where  $\Omega_{!!U}$  is the restriction of  $\Omega$  to the subalphabet  $U = \{U_1, \dots, U_p\}$  and  $\Pi_{!!U}$  is the associated information flow of initial segments.*

**Proof:** statement 1 is just a rewriting of (33), and statement 2 is an immediate consequence of 1. Notice that it is not assumed here that the  $U_i$ 's are *free* inputs: they may be constrained, hence the use of the restriction  $\Omega_{!!U}$  which is in general different from the set of all possible histories on the alphabet  $\{U_1, \dots, U_p\}$ .

Sufficient conditions to guarantee for a SIGNAL program to be deterministic are checked by the SIGNAL compiler as shown in [6] [5].

### 5.3.2 Bisimulations

Consider a process  $\{\Omega, \Pi\}$  where  $\Omega$  is a MCRS and  $\Pi$  the associated information flow of initial segments. Then assume we are given another process  $\{\Omega', \Pi'\}$  in the generalized sense of Definition 1 *revisited*.

**Definition 9** A **bisimulation** from  $\{\Omega, \Pi\}$  onto  $\{\Omega', \Pi'\}$  is a pair  $(H, \Phi)$  where

1.  $H$  is a clock on  $\{\Omega, \Pi\}$  satisfying the condition (35)
2.  $\Phi$  is an isomorphism from  $\{\Omega, \tilde{\Pi}\}$  onto  $\{\Omega', \Pi'\}$  where  $\tilde{\Pi}_n = \Pi_{H_n}$ .

We write

$$\{\Omega, \Pi\} \xrightarrow{H, \Phi} \{\Omega', \Pi'\}$$

to refer to the above property and we define the **bisimulation equivalence** as its transitive and reflexive closure.

**Comment:** Hence, observing two processes that are bisimulation equivalent provides the *same information*, however at rates that may be *different* (cf. the theorem 6 below for a precise statement of this). The communication of a given process with two processes that are bisimulation equivalent also yields two processes that are bisimulation equivalent. These remarks justify the use of the name “bisimulation” which is classical in process calculi. Finally, note that theorem 5 relates bisimulation with determinism.

**Theorem 6** We are given a bisimulation

$$\{\Omega, \Pi\} \xrightarrow{H, \Phi} \{\Omega', \Pi'\}$$

and we set  $C_n(\omega) = H_{n+1}(\omega) - H_n(\omega)$ . We define the map

$$(H, \Phi)^* : K \longrightarrow K'$$

where  $K$  is a clock on the process  $\{\Omega, \Pi\}$  which we decompose according to theorem 2<sup>11</sup>

$$K = \left( \dots (Id \downarrow B^1) \dots \uparrow C^L \right) \downarrow B^L$$

<sup>11</sup>we use in fact the “minimal” decomposition labelled with  $\cdot, *$ 's in theorem 2

and  $K'$  is then given by<sup>12</sup>

$$K' = \left[ \left( \dots ([Id \uparrow C] \downarrow B^1) \dots \uparrow C^L \right) \downarrow B^L \right] \circ \Phi^{-1}$$

Then the map  $(H, \Phi)^*$  is an isomorphism between the clock algebras of  $\{\Omega, \Pi\}$  and  $\{\Omega', \Pi'\}$ .

**Proof:**

1.  $C \circ \Phi^{-1}$  is a signal of clock  $Id$  on  $\{\Omega', \Pi'\}$ . Since  $H$  is a clock, we have

$$\left. \begin{array}{l} H_{n+1}(\omega_1) > k \\ \omega_2 \approx_k \omega_1 \end{array} \right\} \Rightarrow H_{n+1}(\omega_2) > k$$

From this and condition (35) we derive

$$\left. \begin{array}{l} C_n(\omega_1) > k - H_n(\omega_1) \\ \omega_2 \approx_{H_n} \omega_1 \end{array} \right\} \Rightarrow C_n(\omega_2) > k - H_n(\omega_2)$$

so that

$$\omega_2 \approx_{H_n} \omega_1 \Rightarrow C_n(\omega_2) \geq C_n(\omega_1)$$

whence equality follows by symmetry: this proves step 1. Consequently  $Id \uparrow (C \circ \Phi^{-1})$  is a clock on  $\{\Omega', \Pi'\}$ .

2.  $K'$  is a clock on  $\{\Omega', \Pi'\}$  and the image by  $\Phi$  of the partition  $\Pi_{K_n}$  is the partition  $\Pi'_{K'_n}$ . We prove this by induction on the length  $L$  of the decomposition of  $K$ . The result for  $L = 1$  was proved in step 1. Hence we assume the result to hold for  $K$  as in the theorem, and prove it for the clock  $(K \uparrow C^{L+1}) \downarrow B^{L+1}$ . By definition of the multiplexing

$$\omega_2 \approx_{K_n} \omega_1 \Rightarrow C_n^{L+1}(\omega_2) = C_n^{L+1}(\omega_1)$$

But by assumption

$$\omega_2 \approx_{K_n} \omega_1 \Leftrightarrow \Phi(\omega_2) \approx_{K'_n} \Phi(\omega_1)$$

so that  $C_n^{L+1} \circ \Phi^{-1}$  is constant on the elements of the partition  $\Pi'_{K'_n}$ , whence  $(K \uparrow C^{L+1}) \circ \Phi^{-1}$  is a clock. Similarly we prove that  $B^{L+1} \circ \Phi^{-1}$  is constant on the elements of the partition  $\Pi'_{K''_n}$  where  $K'' = (K \uparrow C^{L+1}) \circ \Phi^{-1}$ . This proves step 2.

---

<sup>12</sup> $f \circ g$  denotes the composition of the maps  $f$  and  $g$

program	clocks	signals
$R(x_1, \dots, x_p)$	$H = H(x_1) = \dots = H(x_p)$	$H : R(x_1, \dots, x_p)$
$y := x \ \$ \ x_0$	$H = H(y) = H(x)$	$H : y_n = x_{n-1}$
$y := x \ \text{when} \ b$	$H = H(y) = H(x) \wedge (H(b) \downarrow b)$	$H : y = x$
$Y := u \ \text{default} \ v$	$H(y) = H(u) \vee H(v)$	$H(u) : y = u$ $H(v) - H(u) : y = v$
$P   Q$	$H(P) \cup H(Q)$	$sig(P) \cup sig(Q)$

Table 1:

3. The map  $(H, \Phi)^*$  is invertible and its inverse is  $K' \rightarrow K$  where

$$K' = \left( \dots (Id \downarrow B'^1) \dots \uparrow C'^L \right) \downarrow B'^L$$

and

$$K = \left( \dots (H \downarrow B^1) \dots \uparrow C^L \right) \downarrow B^L$$

where  $B^l = B'^l \circ \Phi$ ,  $C^l = C'^l \circ \Phi$ . This is easy although tedious to verify.

4. That  $(H, \Phi)^*$  is an isomorphism of clock algebras follows immediately from the preceding steps. For instance  $K_2 = K_1 \downarrow B$  yields  $K'_2 = K'_1 \downarrow B'$  where  $B' = B \circ \Phi^{-1}$ . Similarly  $K = K_1 \vee K_2$  rewrites to  $K_1 = K \downarrow B_1$  and  $K_2 = K \downarrow B_2$  where  $B_1 \text{ or } B_2 = \text{true}$  which carries out through the map  $(H, \Phi)^*$ . This finishes the proof of the theorem.

### 5.3.3 The $\Omega$ -semantics of a SIGNAL program

Consider the table 1 where  $H : y = x$  is a short-hand to mean

$$[H_k(\omega) = H(y)_n(\omega) = H(x)_m(\omega)] \Rightarrow [y_n(\omega) = x_m(\omega)],$$

$H(v) - H(u)$  denotes the unique clock  $K$  such that  $H(v) = K \vee (H(u) \wedge H(v))$ . This table shows how to derive the system of clock equations  $H(P)$  and signal equations  $sig(P)$  associated with the program  $P$ .

**Definition 10** We are given a SIGNAL program  $P$  with involved signals  $\{u_1, \dots, u_p; y_1, \dots, y_q\}$  and we assume  $P$  to be deterministic w.r.t.  $u_1, \dots, u_p$ . Then we term a  $\Omega$ -semantics of  $P$  a triple  $\{\Omega_U, \Pi_U; [u_1, \dots, u_p, y_1, \dots, y_q]\}$  where

- $\Omega_U$  is the Trace-*semantics* of the program  $P!!u_1, \dots, u_p$ <sup>13</sup> and  $\Pi_U$  is the associated information flow of initial segments
- for  $i = 1, \dots, p$ ,  $u_i$  is the  $i$ -th signal of the history  $\omega_U \in \Omega_U$  and  $H(u_i)$  its clock
- for  $j = 1, \dots, q$ ,  $y_j$  is a signal of clock  $H(y_j)$  on the process  $\{\Omega_U, \Pi_U\}$

and the family of signals  $[u_1, \dots, u_p, y_1, \dots, y_q]$  satisfy the constraints specified by the program  $P$  according to the table 1.

**Summary.** The  $\Omega$ -*semantics* of a program is generally not unique as shown by the examples of the **MUX** and of the **when**. Theorems 5 and 6 provide a way to build different  $\Omega$ -*semantics* of a program by starting from its *Trace-*semantics** and selecting any  $t$ -uple making this program deterministic. The so obtained  $\Omega$ -*semantics* are *bisimulation equivalent*.

## 6 Conclusion

Starting from elementary discussions related to systems of dynamical equations we motivated the introduction of **SIGNAL** as a language to specify and program reactive systems. As a first attempt to provide a denotational semantics of **SIGNAL** we introduced the *Trace* model which is purely relational and exhibits built-in parallelism: objects within this model are defined as restrictions on the set of all possible joint behaviours of “signals”. To further investigate fundamental issues related to synchronous languages and reactive systems we introduced a drastically new  $\Omega$  model which encompasses both relational and functional styles of specification, and allowed us to introduce the notions of clock and signal via *axioms*. Then two basic constructions where proved able to build any new clock from a master one, namely the *filtering* (or event based undersampling) and *multiplexing* (or event based oversampling). Finally we proved that **SIGNAL** possesses the first construction as a built-in primitive while the second one may be “simulated” in some precise sense. This shows in particular that **SIGNAL** possesses maximum descriptive power for synchronisation mechanisms. Finally, we have shown how the  $\Omega$  model may be used as an alternative semantic domain of **SIGNAL** to obtain different semantics (from relational to purely functional ones) that are bisimulation equivalent.

---

<sup>13</sup>the restriction of  $P$  to the listed signals

We believe that, although probably unfamiliar to the computer science community, our  $\Omega$  model is a significant contribution to fundamental studies on synchronous reactive systems. In particular a variation of this model provided us recently with a multiple clocked generalisation of Leiserson and Saxe’s theory of *retiming* that may be applied to various proofs of equivalence of synchronous reactive systems. This will be presented in a forthcoming paper.

*ACKNOWLEDGEMENT: the authors wish to thank two reviewers who helped in improving a previous version of this paper, and Paul Caspi and Thierry Gautier for fruitful discussions.*

### Appendix: notations

$T \downarrow$	compression of a trace	sect. 3.1
$\Omega_A, \omega_A$	histories on $A$	sect. 3.1
$\omega_a$	signal of port $a$	sect. 3.1
$clock(a), clock(\omega_a)$	clock of $a$	sect. 3.1
$\Omega$	MCRS	sect. 3.2 and 4
$\Omega$	a set	sect. 5
!!	restriction	sect. 3.2
$\Omega_1   \Omega_2$	MCRS communication	sect. 3.2
	(Trace)-semantics	sect. 3.3
$\approx_n$	initial segments	sect. 4.2, eqn. (7)
$\Pi, \Pi_n$	information flows	sect. 4.2 def. 1, 5
$\{\Omega, \Pi\}$	process	sect 4.2 def. 1, 5
$\mathcal{T} \ni s, t$	time index set	sect. 4.2 def. 2
$\overline{H}, H, H(X)$	clocks	sect. 4.2 def. 3
$\approx_{H_n}$	time change	sect. 4.2 def. 4
$H \downarrow B$	filtering	sect. 4.3
$H \uparrow C$	multiplexing	sect. 4.3
$\forall \wedge$	sup, inf of clocks	sect. 4 eqn. (19)

### References

- [1] I.J. AALBERSBERG , G. ROZENBERG, “Theory of traces”, *Theoretical Comp. Sc.* 60, 1-82, 1988.
- [2] E.A. ASHCROFT, W.W. WADGE, “LUCID - a formal system for writing and proving programs”, *SIAM J. Comp.*, vol 5 No 3, 336-354, 1976.

- [3] E.A. ASHCROFT, W.W. WADGE, *LUCID, the Data-Flow programming language*, Academic Press 1985.
- [4] A. BENVENISTE, P. LE GUERNIC, “A denotational theory of synchronous communicating systems”, INRIA research report No 685.
- [5] A. BENVENISTE, B. LE GOFF, P. LE GUERNIC, “Hybrid Dynamical Systems theory and the language SIGNAL”, INRIA research report No 838, 1988.
- [6] A. BENVENISTE, P. LE GUERNIC, C. JACQUEMOT, “Synchronous programming with events and relations: the SIGNAL language and its semantic”, IRISA res. rep. 459, 1989.
- [7] A. BENVENISTE, P. LE GUERNIC, “Hybrid Dynamical Systems Theory and the SIGNAL Language”, *IEEE Trans. on AC*, 35 No 5, 535-546, May 1990.
- [8] G. BERRY, G. GONTHIER “The ESTEREL synchronous programming language: design, semantics, implementation”, CMA res. rep., to appear in *Science of Computer Programming*.
- [9] G. BERRY, “Real Time Programming: Special Purpose Languages or General Purpose Languages”, IFIP 1989, San Francisco.
- [10] J.R. BÜCHI, “On a Decision Method in Restricted Second Order Arithmetic”, Int. Congress on Logic Methodology and Philosophy of Science, Stanford, 1960.
- [11] P. CASPI, D. PILAUD, N. HALBWACHS, J.A. PLAICE, “LUSTRE: a declarative language for programming synchronous systems”, Proc of the 14th ACM symp. on Principles of Programming Languages, 1987.
- [12] P. CASPI, N. HALBWACHS, “A functional model for describing and reasoning about time behaviour of computing systems”, *Acta Informatica* 22, 595-627, 1986.
- [13] P. CASPI, “Clocks in Data-flow languages”, to appear in *Theoretical Comp. Sc.*, 1990.
- [14] C. DELLACHERIE, P.A. MEYER, *Probabilités et Potentiels*, 2nd edition, Hermann, Paris, 1976.

- [15] A. DE BRUIN, W. BOEHM, “The Denotational Semantics of Dynamic Network of Processes”, *ACM Trans. on Prog. Lang. and Syst.*, vol 7 No 4, 656-679, 1985.
- [16] T. GAUTIER, P. LE GUERNIC, L. BESNARD, “SIGNAL, a Declarative Language for Synchronous Programming of Real-time Systems”, proc. of the third Conference on Functional Programming Languages and Computer Architecture, G. Kahn Ed, Lect Notes in Computer Science, vol 274, Springer V., 1987.
- [17] P. LE GUERNIC, T. GAUTIER, “Data-flow to Von Neumann: the SIGNAL approach”, Proc. of the Workshop *Data-flow: a Status Report*, Eilat (Israël), 1989, to appear in *Advanced Topics in Data-Flow Computing*, J.L. Gaudiot, L. Bic Eds, Prentice Hall.
- [18] G. GONTHIER, thesis, Univ. de Nice and Ecole des Mines, 1988.
- [19] D. HAREL, “Statecharts: a visual approach to complex systems”, *Science of Computer Programming*, vol. 8-3, 231-275, 1987.
- [20] D. HAREL, A. PNUELI, “On the development of reactive systems: logic and models of concurrent systems”, proc. NATO Advanced Study Institute on Logics and Models for Verification and Specification of Concurrent Systems, NATO ASI Series F., vol 13, Springer Verlag, 477-498, 1985.
- [21] G. KAHN, “The semantics of a Simple Language for Parallel Programming” in Proceedings IFIP 74 , J.L. Rosenfeld Ed., North Holland, Amsterdam, 471-475, 1974.
- [22] G. KAHN, D.B. MAC QUEEN, “Coroutines and Network of Parallel Processes” in Proceedings IFIP 77, B. Gilchrist Ed., North Holland, Amsterdam, 993-998, 1977.
- [23] E.A. LEE, “Consistency in Data-Flow graphs”, Research Report UCB/ERL M89/125, Electronics Research Lab., College of Eng., U.C. Berkeley, 1989, to appear in *IEEE Trans. on Parallel and Distributed Systems*.
- [24] R. MCNAUGHTON, “Testing and generating Infinite Sequences, by finite Automata”, *Inform. Control* vol 9, 1966, 521-530.

- [25] D.E. MULLER, “Infinite Sequences and Finite Machines”, Proc. 4th annual IEEE Symp. on Switching Theory and Logical Design, 1963, 3-16.
- [26] D.S. ORNSTEIN, *Ergodic Theory, Randomness, and Dynamical Systems*, Yale Univ. Press, 1974.
- [27] J.A. PLAICE, “Sémantique et compilation de LUSTRE, un langage déclaratif synchrone”, Thesis, Institut National Polytechnique de Grenoble, 1988
- [28] G.D. PLOTKIN, *A Structural Approach to operational Semantics*, Lect. Notes, Aarhus Univ, 1981.