Constructive probability and the SIGNalea language : building and processes via programming

Albert Benveniste

UNITÉ DE RECHERCHE
INRIA-RENNES

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P. 105
78153 Le Chesnay Cedex
France
Tél.: (1) 39 63 55 11

Rapports de Recherche

N° 1532

Programme 5
Traitement du Signal,
Automatique et Productique

CONSTRUCTIVE PROBABILITY AND THE SIGNalea LANGUAGE : BUILDING AND HANDLING RANDOM PROCESSES VIA PROGRAMMING

Albert BENVENISTE

Octobre 1991

RR-1532

IRISA

INSTITUT DE RECHERCHE EN INFORMATIQUE
ET SYSTEMES ALEATOIRES

Campus Universitaire de Beaulieu
35042 - RENNES CEDEX FRANCE
Tél. : 99 84 71 00 - Télex : UNIRISA 950 473 F
Télécopie : 99 38 38 32

Constructive Probability and the SIGN*alca* language: building and handling random processes via programming

Publication Interne n°606 - Septembre 1991 - 60 pages - Programme 2

Albert Benveniste

Probabilistic models and stochastic methods to some extend have failed to apply to complex systems and models such as encountered in real-time applications. The reason is that the very basis of any technique based on probability is the famous $\{\Omega, \mathcal{F}, \mathbf{P}\}$ triple ({space of random experiments, associated σ-algebra, probability}). Unfortunately this object is almost inaccessible when complex applications such as queuing networks with synchronization or real-time information processing systems are in consideration. Whence the success of fuzzy reasoning techniques to support information processing applications: fuzzy reasoning is accessible to the user via programming, which makes the construction of complex models feasible. This paper is a first attempt to establish the foundations of a theory which allows us to construct and use random processes via programming. This is why we call it *constructive probability*. The formalism we propose for this is based on the best tool we know to describe dynamical systems, namely SIGNAL. The name of this extension is obviously SIGN*alca*. This formalism allows to specify equally well different objects such as queuing networks or fuzzy-like reasoning systems, this is achieved by extending to dynamical systems the notion of Gibbs random fields on graphs. This paper presents the SIGN*alca* formalism with its associated mathematical model. We show how to specify (partially) randomized dynamical systems with SIGN*alca*, and how to simulate them.

Probabilités Constructives et le langage SIGN*alca*

Les modèles probabilistes et les méthodes stochastiques ont un succès mitigé dans le domaine des grosses applications de traitement de l'information temps-réel. La raison en est sans doute que le fameux $\{\Omega, \mathcal{F}, \mathbf{P}\}$ ({espace des épreuves, tribu, probabilité}) est, de par sa complexité combinatoire, inaccessible à la main pour de telles applications. Ceci explique en partie le succès des techniques floues, qui, elles, proposent une approche par règles, c'est-à-dire de type "programmation". L'objectif de cet article est précisément d'introduire un outil de programmation adapté aux processus stochastiques. Il s'agit d'une extension du langage synchrone SIGNAL, que nous appelons évidemment SIGN*alca*. En fait, SIGN*alca* se présente comme une extension aux systèmes dynamiques de la notion de champ aléatoire. SIGN*alca* permet de décrire et construire des objets aussi variés que des réseaux de files d'attentes, ou des systèmes temps-réels de traitement de l'information incertaine (par exemple pour les applications de diagnostic).

CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE (U.R.A. 227) UNIVERSITE DE RENNES I I.N.S.A. DE RENNES
INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE (UNITE DE RECHERCHE DE RENNES)

Constructive Probability and the SIGN*alea* language: building and handling random processes via programming

Albert Benveniste*

September 27, 1991

Abstract

Probabilistic models and stochastic methods to some extend have failed to apply to complex systems and models such as encountered in real-time applications. The reason is that the very basis of any technique based on probability is the famous $\{\Omega, \mathcal{F}, \mathbf{P}\}$ triple ({space of random experiments, associated σ-algebra, probability}). Unfortunately this object is almost inaccessible when complex applications such as queuing networks with synchronization or real-time information processing systems are in consideration. Whence the success of fuzzy reasoning techniques to support information processing applications: fuzzy reasoning is accessible to the user via programming, which makes the construction of complex models feasible. This paper is a first attempt to establish the foundations of a theory which allows us to construct and use random processes via programming. This is why we call it *constructive probability*. The formalism we propose for this is based on the best tool we know to describe dynamical systems, namely SIGNAL. The name of this extension is obviously SIGN*alea*. This formalism allows to specify equally well different objects such as queuing networks or fuzzy-like reasoning systems, this is achieved by extending to dynamical systems the notion of Gibbs random fields on graphs[1]. This paper presents the SIGN*alea* formalism with its associated mathematical model. We show how to specify (partially) randomized dynamical systems with SIGN*alea*, and how to simulate them.

*IRISA-INRIA, Campus de Beaulieu, 35042 RENNES CEDEX, FRANCE, benveniste@irisa.fr. **Keywords: semantics of programming languages, random or uncertain real-time systems.**

[1] Gibbs random fields on graphs are often referred to as "cellular automata" or as "neural networks" in the AI litterature, we prefer however to term them "Gibbs random fields on graphs" since this precise mathematical name is free from ambiguity.

Contents

1 Introduction and motivations

Development of large real-time systems is one of the most challenging tasks facing the engineering community today. Typical instances of such systems are found in process control industry, power industry, transportation systems, telecommunication systems, military systems, to mention just a few. In these applications, real-time processing of uncertain complex information emerges as a key difficulty. Such a task occurs for instance in diagnostics and monitoring of industrial plants (recall the Three Miles Island accident, where poor alarm processing has been recognized as largely responsible), or in real-time pattern recognition (speech recognition, military information processing systems). On the other hand, specifying, developing, and simulating complex systems involving synchronization and uncertainty in timing is another difficulty: it is for instance encountered in large communication networks.

The author of this paper, among others, investigated issues related to this challenge in the framework of the joint IEEE-IFAC project "*Facing the challenge of computer science in the industrial applications of control*", see [1]. Meetings with industrialists revealed the following interesting points:

- A large part (typically over 90%) of the application software is devoted to nonmathematical processing which is heuristic in nature. The reason for this is that, while the very heart of most of the above mentioned systems can be modelled and handled with known mathematical methods, this is generally not the case for the whole system.

- This heuristic part of the processing is generally of large combinatorial complexity (filtering and processing the thousands of alarms in nuclear power plants is a good instance of such a situation, and so is the signalling of modern transportation systems such as the french TGV [1]).

- From this situation results the popular nightmare of the engineer in developing such systems, namely the "software error": a software error is something wrong somewhere between the specification of the problem and its final implementation. For processings exhibiting a large combinatorial complexity, it is generally difficult to precisely locate such errors: the *design* of the heuristics or their *programming* may be faulted.

- This situation explains why programming-in-the-large methodology is considered as worth of a major effort in the above mentioned industries: it is expected that good programming methodologies may help coping with the (accepted) situation of large combinatorial complexity. One way of doing this is to rely on expert systems and object oriented programming technologies.

- The best way to cope with the above mentioned issue, however, seems to transfer new parts of the processing from heuristic to mathematical approach. For instance, uncertain information processing systems are sometimes handled by *fuzzy reasoning* techniques. Fuzzy reasoning is an attempt to mix the advantages of handling combinatorial complexity in a modular way, i.e., via programming using rules, with those

of dealing with uncertainty.

This paper is mainly concerned with *real-time uncertain information processing systems.* It is our opinion that the venerable cultural background of probability, statistics, and stochastic processes, form a natural framework for such an application area: deep mathematical results are available, which provide us with a rigorous formalism, ways to approximate with simple asymptotics the behaviour of large systems, rationales to infer unknowns from measurements and models from data. Surprisingly enough, this framework failed to be dominant in the engineering community dealing with real-time uncertain information processing systems, as mentioned above. The reason is that the famous $\{\Omega, \mathcal{F}, \mathbf{P}\}$ triple ({space of random experiments, associated σ-algebra, probability}) is the prerequisite for any probabilistic processing, and it happens that the structure of such Ω's when models of complex systems are considered is almost intractable with pen, paper, and human neurons.

As far as we know, little work has been performed with the objective of providing access to probability tools via programming. An attempt in this direction may be found in [11], where probabilities are introduced in the "branching time temporal logic" model of Manna and Pnueli; but the resulting model seems hard to use in practice. The only practical attempt to build stochastic processes in a modular way seems to be the stochastic (timed) Petri net model [20]; this model, however, is more oriented toward labelling Petri nets with probabilistic attributes rather than a way by itself to construct probability spaces. On the other hand, recent work from the area of statistical mechanics [13] and AI oriented statistics [21] have put emphazis on the interest of building complex stochastic processes from "local characteristics". Typical instances of such situations are (finite or infinite) systems of randomly interacting particles, similar models have proven useful in the area of neural networks, we shall refer to them as *Gibbs random fields on graphs.* This point of view is one important component of the foundations of our theory of *constructive probability.* Basically, what we show in this paper is the following:

- Gibbs random fields on graphs provide a clean framework to specify probabilities on large combinatorial Ω spaces from local characteristics that are of low complexity.
- An elegant way to build complex probabilities is to define them starting from simpler ones, by taking the conditional probability given some subset which in turn is specified via a set of constraints. Such a method can be applied to derive complex dynamics from sequences of independent random variables[2].
- Dynamical systems can be conveniently specified using the real-time programming language SIGNAL [15, 3], this is the way constraints on dynamics will be specified in our approach.
- A new operator to combine given random processes will be introduced, the "shuffle

[2] this principle has some flavour of the Cameron-Martin formula which is used to derive the law of any diffusion from that of the Brownian motion in the area of stochastic calculus.

product", which will be the basis for a modular construction of complex random processes from small primitives.

- All these principles can be used in practice thanks to the availability of some programming language we call SIGN*alea* which implements them.

The paper is organized as follows. Section 2 is devoted to elementary facts from the theory of random fields on graphs. In Section 3, the languages SIGNAL and its extension SIGN*alea* will be presented, and some toy examples will be discussed. Section 4 is devoted to the mathematical model of SIGN*alea*: we define formally which mathematical object is associated with a given SIGN*alea* program. This section is a very important one: the new concepts of shuffle product, conditioning, and communication are introduced, technical notions are rejected in appendix A, and proofs of mains theorems are found in appendix B. Then the expressive power of SIGN*alea* is investigated in Section 5, it is shown in particular that multiple clocked stochastic automata (i.e., automata with multiple time scales) can be built with SIGN*alea*, as well as (stochastic, timed) Petri nets. In software engineering, compilation consists of deriving from the source program (the "specification") an equivalent executable form; in our case, compilation consists of showing how the (random) system specified by a given SIGN*alea* program can be simulated: this is the subject of Section 6.

2 Random fields and potentials: some recalls and remarks

What is a random field?

We review here some basic facts from elementary random field theory [13][3]. We are given some finite set Ω. A probability $\mathbf{P}$ on Ω can be defined via some *energy* function $-\infty < U \leq +\infty$ by setting

$$\mathbf{P}(\omega) = \frac{1}{Z} e^{-U(\omega)} \ , \ Z = \sum_{\omega} e^{-U(\omega)} \tag{1}$$

It is convenient to build U from "local" components in the following sense. We assume that Ω is of the following form:

- $G = \{V, E\}$ is a nondirected graph with vertices V and edges E, elements of V are written v and are usually referred to as *sites*. Two sites are said to be neighbours if they are connected by an edge. A subset C of V is said to be a *clique* if every pair of sites of C are neighbours.

- Every site v is labelled by a symbol ω_v from some finite alphabet S_v, we write $\omega = (\omega_v)_{v \in V}$ and assume that Ω is the set of such ω's. More generally, we write $\omega_B = (\omega_v)_{v \in B}$ so that, in particular,

$$\omega_{B \cup C} = (\omega_B, \omega_C) \text{ if } B \cap C = \emptyset \tag{2}$$

[3] we use here the notations of [2], chapter 2 of part I, exercise 2.7.15.

A *potential* is a family of functions $U_B(\omega)$ where B ranges over the subsets of V, such that $U_B(\omega)$ actually depends only on ω_B. We usually associate energy functions with potentials via the formula

$$U(\omega) = \sum_B U_B(\omega) = \sum_B U_B(\omega_B) \tag{3}$$

Such an object $\{\Omega, \mathbf{P}\}$ where $\mathbf{P}$ is built from U, is called a *random field*, and $\mathbf{P}$ is said to be the Gibbs measure associated with the potential (U_B). The potential (U_B) is said to be *local* if $U_B \equiv 0$ when B is not clique. Giving a local potential is a convenient way to specify a probability on Ω (a large combinatorial object) via some "local interactions": it is thus a constructive way to build probability distributions on large spaces. Local potential are known to give raise to *Markov* random fields, where Markovianity is defined based on the neighbouring system associated with the graph G, we shall not need to elaborate further on this point.

A practical way to specify a random field according to the definition above is the following. Assume we are given a collection $(x_1, \cdots, x_p)$ of variables representing, say, measurements, indicators, or observations of some phenomenon that are subject to uncertainty. Furthermore, assume that we have some prior knowledge about the possible linkings between these variables, and we represent it via local interactions of the form $U_i(x_{k_{i,1}}, \cdots, x_{k_{i,p_i}})$ where i is some index and $k_{i,j} \in \{1, \cdots, p\}$. The interpretation of this is that a tuple $(x_1, \cdots, x_p)$ is more likely if its total energyy

$$U(x_1, \cdots, x_p) = \sum_i U_i(x_{k_{i,1}}, \cdots, x_{k_{i,p_i}})$$

is low. This defines a random field as follows. Take $V = \{x_1, \cdots, x_p\}$ as set of vertices. Draw an edge between x_k and x_l if both indices k and l occur in the arguments of the same U_i interation for some i. This yields the graph G. Then denote by S_i the domain of x_i, and take $\Omega = S_1 \times \cdots \times S_p$. Obviously the family (U_i) defines a local potential on this graph G. This method of building random fields on graphs will be widely used in the sequel.

Some advantages of defining probabilities via local interactions

A first advantage relates to issues of *computational complexity.* When Ω is large, computing the normalizing constant factor Z in (1) is of excessive computational cost. In contrast, each U_B, depending only on ω_B, typically involves a small amount of sites. Hence everything that can be computed using the local interactions U_B only will be of low computational cost.

One possible way of computing $\mathbf{P}$ using local interactions only is to interpret the decomposition (3) as a *Bayes rule.* This is not always possible as we shall see later. But this is for instance the case for Markov chains as explained next. Let $\pi(x, y)$ denote the transition matrix of some finite state Markov chain, and introduce $U(x, y) = -\log \pi(x, y)$. Then the probability of a sample path $(x_0, \cdots, x_n)$ of the chain with initial condition x_0 is exactly

$$\mathbf{P}(x_0, \cdots, x_n) = \prod_{i=1}^{n} \pi(x_{i-1}, x_i) = \exp - \sum_{i=1}^{n} U(x_{i-1}, x_i)$$

without the need for a normalizing constant factor Z, so that the interpretation of (3) as a Bayes rule is justified here.

Finally, there are statistical reasons for dealing with random fields and defining probabilities via local interactions, as lengthly discussed in the nice book [21]. Assume it is wished to infer $\mathbf{P}$ from observations, and available experiments only involve ω_B's for B any clique, i.e., only local experiments are available. Then the random fields associated with the graph G are the *maximum entropy* distributions based on the available experiments, and the maximum entropy principle is known as the proper rationale for fitting models from data.

Computing conditional distributions and potentials

An interesting byproduct of specifying probabilities via potentials based on formulae (1,3) is that, for $\mathcal{W} \subset \Omega$ such that $\mathbf{P}(\mathcal{W}) > 0$, the formula[4]

$$[U_B|\mathcal{W}](\omega) =_{\text{def}} U_B(\omega)\ 1_{\mathcal{W}}(\omega) + \infty\ 1_{\mathcal{W}^c}(\omega) \tag{4}$$

defines a potential which has the conditional probability $\mathbf{P}(.|\mathcal{W})$ as associated Gibbs measure. Let us now investigate more specifically on a simple case how such a facility can be used in practice. Select three sites v_0, v_1, v_2 belonging to the same clique, and assume that the subset $\mathcal{W}$ above is specified via the constraint

$$\mathcal{W} =_{\text{def}} \{\omega\ :\ \omega_{v_0} = f(\omega_{v_1}, \omega_{v_2})\} \tag{5}$$

for some function $f : S \times S \mapsto S$. Then formula (4) reduces in this case to a very appealing form: recalling that $\omega = (\omega_v)_{v \in V}$, just

$$\text{substitute } f(\omega_{v_1}, \omega_{v_2}) \text{ for } \omega_{v_0} \text{ in } U(\omega), \tag{6}$$

this yields a new function depending on $\omega_{\{v_0\}^c}$ (i.e. those coordinates ω_v for $v \neq v_0$) only. This function is exactly the desired $[U_B|\mathcal{W}](\omega)$ energy function!

Let us investigate when the formula (3) can be interpreted as a Bayes rule. Assume that U can be decomposed in the following way:

$$U = U_{B^c} + U_{C \cup B} \tag{7}$$

where B, C are subsets of V with empty intersection, and, as before, the notation U_A where $A \subset V$ means that $U_A(\omega)$ depends only on ω_A. Then it holds that, for ω_{B^c} given and recalling (2),

$$\begin{aligned} \mathbf{P}(\omega|\omega_{B^c}) &\sim e^{-U_{B^c}(\omega_{B^c})}\, e^{-U_{C \cup B}(\omega_C, \omega_B)} \\ &\sim e^{-U_{C \cup B}(\omega_C, \omega_B)} \end{aligned}$$

(where $\sim$ means "proportional to") since ω_{B^c} is constant and only ω_B is a variable. Hence, if (7) holds, then the energy function $U_{C \cup B}(\omega_C, \omega_B)$ specifies the conditional probability

[4] 1_A denotes the functions taking the value 1 if its argument lies in the set A, and 0 otherwise, and A^c denotes the complement of the set A.

$\mathbf{P}(.|\omega_{B^c})$. Note that in this case $\mathbf{P}(.|\omega_{B^c})$ is a function of ω_C only, a kind of Markov property. However it is *not* true that $U_{B^c}(\omega_{B^c})$ specifies the marginal of $\mathbf{P}$ on B^c which is defined by

$$\mathbf{P}_{B^c}(\omega_{B^c}) = \sum_{\omega_B} \mathbf{P}(\omega_{B^c}, \omega_B)$$

In fact, it holds that

$$\begin{aligned} \mathbf{P}_{B^c}(\omega_{B^c}) &= \frac{1}{Z}\, e^{-U_{B^c}(\omega_{B^c})} \sum_{\omega_B} e^{-U_{C\cup B}(\omega_C,\omega_B)} \\ &= \frac{1}{Z} \exp - \left\{ U_{B^c}(\omega_{B^c}) \; - \log \sum_{\omega_B} e^{-U_{C\cup B}(\omega_C,\omega_B)} \right\} \end{aligned}$$

(where Z denotes again the appropriate normalizing constant factor) so that a corrrective term must be substracted from U_{B^c} to get the marginal. So, if it is wanted that the additive decomposition (7) be interpreted as a Bayes rule, just make sure that

$$\tilde{U}(\omega_C) =_{\text{def}} \log \sum_{\omega_B} e^{-U_{C\cup B}(\omega_C,\omega_B)} \text{ be indeed independent from } \omega_C \tag{8}$$

Finally, let us indicate how to solve the following problem: we are given U_{B^c} and $U_{C\cup B}$ with B, C as above, and we want to build an energy function U such that U_{B^c} and $U_{C\cup B}$ respectively define the marginal and conditional probabilities of the Gibbs measure associated with this U. This is done as follows:

$$\begin{aligned} U_{C\cup B|C}(\omega_C,\omega_B) &=_{\text{def}} \quad U_{C\cup B}(\omega_C,\omega_B) + \log \sum_{\omega_B} e^{-U_{C\cup B}(\omega_C,\omega_B)} \\ U(\omega) &=_{\text{def}} \quad U_{B^c} + U_{C\cup B|C} \end{aligned} \tag{9}$$

Then the "compensated" potential $U_{C\cup B|C}$ defines the same conditional distribution as $U_{C\cup B}$ and satisfies (8), so that the decomposition (9) can now be interpreted as a Bayes rule. This remark will be widely used in the sequel.

To summarize, conditioning is performed in a very elegant way within this framework. Our aim in the sequel is to

1. generalize the notion of Gibbs random fields on graphs to (real-time) dynamical systems,
2. perform such a generalization for dynamical systems that are multiple clocked,
3. generalize this to systems that are "partially randomized" in a sense we shall define later,
4. provide some effective syntax to perform such specification in practice.

This justifies the name of *constructive probability* for our theory.

3 SIGN*alea*

We first discuss the nature of time for the kind of dynamical system we have to consider. Then we briefly describe the SIGNAL language and introduce and discuss in details its SIGN*alea* extension.

3.1 What is time?

What is the nature of time for the kind of models of dynamical system we want to consider? Complex applications such as the one mentioned in the introduction are inherently distributed in nature. 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 externally. Hence the nature of time in the systems in consideration is by no means universal, but rather local to each subsystem, and consequently multiform. This very fundamental remark justifies the kind of model SIGNAL relies on, we discuss it informally now.

Our model handles infinite sequences of data with a certain kind of restricted asynchronism. Assume that each sequence, 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 data (we shall refer to informally as a *signal* in this discussion) may look like

$$1, -4, \perp, \perp, 4, 2, \perp, ... \tag{10}$$

which is interpreted as the signal being absent at the instants $n = 3, 4, 7, ...$ etc. Dynamical systems specified via constraints on signals of the form (10) will be considered here. A typical way of specifying such constraints will be to write equations relating different signals. The following questions are immediate from this definition:

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

$$\{1, -4, \perp, \perp, 4, 2, \perp, ...\}\ ,\ \{\perp, 1, \perp, -4, \perp, 4, \perp, 2, \perp, ...\}\ ,\ \{1, -4, 4, 2, ...\}$$

Consider an "observer"[5] 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.

[5] in the common sense, no mathematical definition is referred to here

(2) How to interconnect two systems? Consider the following two systems specified via equations:

$$y_n = \text{ if } x_n > 0 \text{ then } x_n \text{ else } \perp \tag{11}$$

and the usual addition on sequences, namely

$$z_n = y_n + u_n \tag{12}$$

In combining these systems, it is certainly preferable to match the successive occurrences $y_1, y_2, \ldots$ in (12) with the corresponding *present* occurrences in (11) so that the usual meaning of addition be met. But this is in contradiction with the bruteforce conjunction of equations (11,12)

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

which yields $z_n = \perp + u_n$ whenever $x_n \leq 0$. Motivated by the above discussion, we proceed now on presenting SIGNAL.

3.2 Brief review of SIGNAL

We shall introduce only the primitives of the SIGNAL language[6], and drop any reference to typing, modular structure, and various declarations; the interested reader is referred to [15]. 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 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(x1,...,xp)
 (ii)  y := x $1 init x0
(iii)  y := x when b
 (iv)  y := u default v
  (v)  P | Q
 (vi)  P {x1,...,xp}
```

Their intuitive meaning is as follows (for a formal definition, see the section 2):

[6] SIGNAL is a joint TradeMark from INRIA and CNET; a graphical integrated development environment called SILDEX is commercially available from TNI, Brest, France, which implements the SIGNAL language.

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

$$\mathtt{R(x1,...,xp)} \iff \forall n : R(\mathtt{x1}_n, ..., \mathtt{xp}_n) \text{ holds}$$

where $R(...)$ 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` ($\forall n : \mathtt{z}_n = \mathtt{x}_n + \mathtt{y}_n$). 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 `R(...)` is the universal relation, i.e., it contains all the p-tuples of the relevant domains, the resulting SIGNAL instruction only constrains the involved signals to have the same clock: the so obtained instruction will be written `x ^= y ^=` $\cdots$ and only forces the listed signals to have the same clock.

(ii) shift register.

$$\mathtt{y := x\ \$1\ init\ x0} \iff \forall n > 1 : \mathtt{y}_n = \mathtt{x}_{n-1}, \mathtt{y}_1 = \mathtt{x0}$$

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 diagram summarizing this instruction:

x :	1	2	$\perp$	$\perp$	3	4	$\perp$	$\perp$	5	6	9	...
b :	*t*	*f*	*t*	$\perp$	*f*	*t*	*f*	$\perp$	$\perp$	*f*	*t*	...
y :	1	$\perp$	$\perp$	$\perp$	$\perp$	4	$\perp$	$\perp$	$\perp$	$\perp$	9	...

(iv) `y` merges `u` and `v`, with priority to `u` when both signals are simultaneously present. Here follows a diagram summarizing this instruction:

u :	1	2	$\perp$	$\perp$	3	4	$\perp$	$\perp$	5	$\perp$	9	...
v :	$\perp$	$\perp$	$\perp$	3	4	10	$\perp$	8	9	2	$\perp$	...
y :	1	2	$\perp$	3	3	4	$\perp$	8	5	2	9	...

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 $1 init x0
 |)
```

denotes the system of recurrent equations:

$$\begin{aligned} y_n &= zy_n + a_n \\ zy_n &= y_{n-1},\ zy_1 = \mathtt{x0} \end{aligned}$$

On the other hand, the program

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

yields

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

where (x_n) denotes the sequence of present values of x. Hence the communication | causes $\perp$ to be inserted whenever needed in the second system z:=y+u. This is what we wanted for the example (11,12).

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

Two different mathematical models of SIGNAL are provided in [4]. It is shown there that SIGNAL is a convenient tool to effectively construct complex multiple-clocked dynamical systems. It is a sort of a "RISC" programming language to specify complex real-time applications. SIGNAL is currently used as a real-time programming language and belongs to the so-called family of "synchronous languages", see [3, 8, 10, 15].

3.3 The SIGN*alea* extension

SIGN*alea* turns out to be a very little extension of SIGNAL. Its instruction set is:

```
  (i)   R(x1,...,xp)
 (ip)   potential U(x1,...,xp)
 (ii)   y := x $1 init x0
(iii)   y := x when b
 (iv)   y := u default v
  (v)   P | Q
 (vi)   P {x1,...,xp}
```

Thus only the instruction (ip) has been introduced. The notation "U(x1,...,xp)" refers to any real-valued function of the mentioned arguments. First, this instruction is of type (i): it forces the signals (x1,...,xp) to have the *same* clock. Then, considered alone, this

instruction causes the signals (x1,...,xp) to behave jointly according to the distribution[7]

$$\left(\frac{1}{Z}e^{-U(x_1,\ldots,x_p)}\right)^{\otimes N}, \; Z = \sum_{(x_1,\ldots,x_p)} e^{-U(x_1,\ldots,x_p)}$$

where Z is the usual normalizing constant factor. Thus instruction (ip) is used to specify *independent identically distributed (i.i.d.)* sequences of random variables via the negative loglikelihood of their joint marginal distribution. On the other hand, the initial condition x0 in instruction (ii) can be made random too. Then some SIGNAL program can be used to specify a subset of "legal" behaviours, thus generalizing the simple example of equation (5). Combining this SIGNAL program with previously mentioned "potential" statement via the operator | amounts to specifying the given i.i.d. random process as conditioned to live on the set of legal behaviours, thus generalizing formula (6). More generally, several "potential" statements can be combined with a SIGNAL program.

The following macro will be useful to define conditional probabilities in a proper way:

```
(ipc)   given x1,...,xp potential U(x1,...,xp,y1,...,yq)
```

This instruction is by definition equivalent to

```
potential U_x(x1,...,xp,y1,...,yq)
```

where U_x and U are related as follows:

$$U_x(x,y) =_{\text{def}} \; U(x,y) + \log \sum_y e^{-U(x,y)} \tag{13}$$

where x and y denote the tuples $(x_1,...,x_p)$ and $(y_1,...,y_q)$ respectively. Formula (13) is just the translation of formula (9) with the present notations. The (ipc) instruction allows us to specify some conditional distribution of y given x via U without having to write explicitly the compensator in the righthand side of (13). Recall that the potential $V(x) + U_x(x,y)$ can be interpreted as a Bayes rule where the first term specifies the marginal distribution of x and the second one the conditional distribution of y given x.

3.4 Some simple SIGN*alea* programs

A Markov chain. We claim that the following program specifies a Markov chain:

```
(| given y potential U(y,x)
 | y := x $1 init x0
 |)
```

Let us check this. Using notation (13), the first instruction specifies an i.i.d. sequence with distribution

$$\left(\frac{1}{Z}e^{-U_x(y,x)}\right)^{\otimes N}$$

[7] to simplify, we consider in this paper only finitely valued random variables.

In particular the distribution of an observation up to instant n is given by

$$\frac{1}{Z^n} e^{-\sum_{i=1}^{n} U_x(y_i, x_i)}$$

Next, we take into account the constraint specified by the second instruction. According to formula (6) this yields the conditional distribution

$$e^{-\sum_{i=1}^{n} U_x(x_{i-1}, x_i)}$$

(with no need for a further normalizing constant factor). From this follows immediately that $(x_n)_{n\geq 0}$ is a Markov chain with transition matrix $\pi(u, v)$ given by

$$\pi(u, v) = \frac{e^{-U(u,v)}}{\sum_v e^{-U(u,v)}} = e^{-U_x(u,v)}$$

and initial condition x_0. Finally, if it is wished to have the initial condition `x0` as being random, just specify the corresponding distribution when declaring initial conditions. The examples to follow are slightly more involved. Therefore we cannot expect to analyze them easily without relying on the adequate mathematical tools we shall introduce later. Hence we just comment them informally.

A simple stochastic automaton. We claim that the following program specifies some kind of a stochastic automaton:

```
(| given z potential U(x,z)
 | potential V(a)
 | z := a default zz
 | zz := z $1
 |)
```

This SIGN*alea* program consists of two "`potential`" instructions which specify the random characteristics of the system, plus the following SIGNAL program:

```
(| z := a default zz
 | zz := z $1
 |)
```

which specifies the constraint. A behaviour of the latter program is depicted below, where z_0 denotes some initial condition:

time :	1	2	3	4	5	6	7	8	9	10	11	...
a :	$\perp$	$\perp$	a_1	$\perp$	a_2	a_3	$\perp$	$\perp$	$\perp$	$\perp$	a_4	...
z :	z_0	z_0	a_1	a_1	a_2	a_3	a_3	a_3	a_3	a_3	a_4	...
zz :	z_0	z_0	z_0	a_1	a_1	a_2	a_3	a_3	a_3	a_3	a_3	...

Hence, when a is received, it is fed into some memory and at the same time is delivered; then this memory can be read an *unspecified* amount of times between two successive a's;

the resulting signal is z. On the other hand, the input signal a is i.i.d. with distribution proportional to $e^{-V(a)}$, and each time z is produced, a random variable x is emitted according to a conditional distribution proportional to $e^{-U(x,z)}$. Here the macro "**given ... potential ...**" is used in order to guarantee that $U(x,z)$ does not modify the distribution of a, cf. (8,9). We insist on the fact that the amount of successive x's between two a's is unspecified and not random: the last two instructions just specify that z carries the last received or current value of a and that it is more frequently delivered than a is received (cf. the meaning of the **default** instruction). If it is wished to produce between successive a's an amount of x's according to some specific distribution, just combine this program with another one via a "^=" statement:

```
(|(| given z potential U(x,z)
   | potential V(a)
   | z := a default zz
   | zz := z $1
   |)
 |(| x ^= N
   |)
 |(| potential W(R)                    % R positive integer %
   | N := R default (ZN-1)
   | ZN := N $1 init 1
   | R ^= (true when (ZN=1)) ^= a
   |)
 |)
```

Recall that the "^=" instruction is an instruction of type (i) which only forces the listed signals to have the same clock and does nothing else. The last module specifies a decreasing counter with reset R. Due to the last instruction, reset must occur exactly when the counter is empty, and simultaneously, a is read. Thus x and N have the same clock and a is read when the counter gets empty. Due to the first instruction of the last block, R has some desired distribution and is i.i.d., so that the third block of this program specifies a renewal process.

If this renewal process was indeed a (discrete time) Poisson process, the following simpler program can be used:

```
(|(| given z potential U(x,z)
   | potential V(a)
   | z := a default zz
   | zz := z $1
   |)
 |(| x ^= c
   |)
 |(| potential if c=true then 1 else p fi
   | a ^= (true when c)
   |)
 |)
```

where the boolean signal c is the outcome of some coin tossing; the potential specified by the expression "`if c=true then 1 else p fi`" is equal to

$$W(c) = 1_{\{c=true\}} + p1_{\{c=false\}}$$

so that the bias of the coin is specified by the parameter p. In both programs above, the resulting effect is to produce a random amount (with some specified distribution) of samples of x between successive a's.

Finally, if it is desired that a be a Markov chain instead of i.i.d. random variables, just combine the first example with the previous one:

```
(|(| potential PI(b,a)
   | b := a $1
   |)
 |(| given z potential U(x,z)
   | z := a default zz
   | zz := z $1
   |)
 |(| x ^= N
   |)
 |(| potential W(R)                    % R positive integer %
   | N := R default (ZN-1)
   | ZN := N $1 init 1
   | R ^= (true when (ZN=1)) ^= a
   |)
 |)
```

Modular, or even graphical[8] programming can be easily provided that explicitly exhibits the modular structure shown here with the help of parentheses. Renewal processes and synchronizations are useful basic tools for modelling complex queuing networks. Also stochastic automata of the above kind are for instance typically encountered in Hidden Markov Models (HMM) such as used in speech recognition systems [19]; in fact, building complex hierarchical HMM's is often performed using computer assisted generation tools.

Programming with rules. Most popular expert systems rely on rules. Fuzzy rules are provided in fuzzy programming systems. Here we provide a SIGN*alea* program that may be interpreted as the following sentence: *if events* A *and* B *happen to occur at the same time, then it is likely (with truth value p) that* C *follows to happen.* The SIGN*alea* program is:

```
(| AB := A when B
 | potential p(C,AB)
 |)
```

AB is the event which occurs when both A and B occur, and the second instruction quantifies the interaction between the two events. Here we do not rely on the macro "`given ...`

[8] in the style of the SIGNAL block-diagram interface, see [15] or the SILDEX system.

potential ..." since we want C and AB to play a symmetric role: they are just related, we do not care and maybe do not know which one is the cause and which one the effect. Then if C is observed we can infer about A and B, and conversely, if A and B are observed, we can infer about C. This is typical of probabilistic reasoning in the presence of uncertainty.

Justifying these examples is certainly beyond the capacity of such elementary arguments we used to analyze the first Markov chain example. Some further mathematical machinery has to be introduced to help for this. This is the purpose of the remainder of the paper.

4 The mathematical model of SIGN*alea*.

Stochastic models of random real-time systems generally involve a triple $\{\Omega, (\mathcal{F}_n), \mathbf{P}\}$ called *probability space*, where Ω is the set of possible behaviours (or trajectories), $(\mathcal{F}_n)$ is an increasing family of σ-algebras representing the flow of incoming information as time n grows, and $\mathbf{P}$ is a probability. We shall now introduce 1/ elementary such probability spaces, and 2/ some operators to combine given probability spaces to build another one.

4.1 The mathematical model

This subsection is organized following two levels of difficulty: simple notions are presented in the core of the text, and more general and difficult notions are given in appendix A. For a first reading, this appendix may be skipped.

4.1.1 Primitives

Consider a finite domain S of values, and introduce

$$\begin{aligned} \Omega &= \mathbf{N} \mapsto S \ , \quad \text{for } \omega \in \Omega \text{ we write } x_n(\omega) =_{\text{def}} \omega(n) \\ \mathcal{F}_n &= \sigma\{x_m \ : \ m \leq n\} \end{aligned} \tag{14}$$

where $\mathbf{N} \mapsto S$ denotes the set of all functions from $\mathbf{N}$ into the domain S. The notation $\sigma\{x_m \ : \ m \leq n\}$ denotes the σ-algebra generated by the coordinates x_m for $m \leq n$, i.e., for $A \subseteq \Omega$,

$$\begin{array}{c} A \in \mathcal{F}_n \\ \Updownarrow \quad \text{by definition} \\ \left.\begin{array}{r} \omega_1 \in A \\ x_m(\omega_1) = x_m(\omega_2) \ \forall m \leq n \end{array}\right\} \Rightarrow \ \omega_2 \in A \end{array} \tag{15}$$

that is to say, the statement "$\omega \in A$" involves only initial segments of ω of length at most n. Similarly, a function F defined on Ω is said to be *$\mathcal{F}_n$-measurable* if $F(\omega)$ actually depends only on the initial segment of ω of length at most n, i.e., if

$$x_m(\omega_1) = x_m(\omega_2) \ \forall m \leq n \ \Rightarrow \ F(\omega_2) = F(\omega_1)$$

The pair $\{\Omega, (\mathcal{F}_n)\}$ models what can be observed while monitoring a S-valued signal as time n increases when no prior constraint is set on the possible behaviours of this signal: this is our first primitive object, we call it a primitive *process*. It models "all possible behaviours" of an S-valued signal. A general and abstract definition of a process is provided in appendix A.

If we want this S-valued primitive process to be randomized, we proceed as follows. Given some real-valued function $U(x)$, we consider the probability $\nu(x) = \frac{1}{Z}e^{-U(x)}$ where Z is a proper normalizing constant factor, and we equip $\{\Omega, \mathcal{F}_\infty\}$ with the probability $\mathbf{P} = \nu^{\otimes \mathbf{N}}$, i.e., we define the sequence (x_n) as being random, independent, and identically distributed (*i.i.d.*) with marginal distribution ν. The resulting primitive is written $\{\Omega, (\mathcal{F}_n), \mathbf{P}\}$, we call it a primitive *random process*.

4.1.2 Shuffle products

We first define how to compose primitive processes. Consider two primitive processes $\{\Omega^x, (\mathcal{F}_n^x)\}$ and $\{\Omega^y, (\mathcal{F}_n^y)\}$ as above, with respective domains S^x and S^y. Introduce the alphabets

$$S_\perp^x = S^x \cup \{\perp\} \ , \ S_\perp^y = S^y \cup \{\perp\}$$

where $\perp$ is some distinguished element to be interpreted as the absence of value. Then set

$$S = (S_\perp^x \times S_\perp^y) - \{(\perp, \perp)\}$$

The *shuffle product* is by definition the process

$$\{\Omega^x, (\mathcal{F}_n^x)\} \sqcup\!\sqcup \{\Omega^y, (\mathcal{F}_n^y)\} =_{\text{def}} \{\mathcal{W}, (\mathcal{G}_n)\} \tag{16}$$

where

$$\begin{aligned} \mathcal{W} &= \mathbf{N} \mapsto S \\ \mathcal{G}_n &= \sigma\{w(m) \ : \ w \in \mathcal{W}, m \leq n\} \end{aligned} \tag{17}$$

The trajectory $w(n)$ at instant n has two components we denote by $x_n(w)$ and $y_n(w)$. Then $x_n = \perp$ refers to the absence of x in the shuffle at the considered instant. Thanks to our definition of S, it never happens that both components are absent simultaneously. An observer watching the component x only and ignoring the $\perp$'s will exactly observe the original space $\{\Omega^x, (\mathcal{F}_n^x)\}$: this is the characterization of shuffle products (and communications we shall introduce later) which [4] relied on. This definition is related to that of shuffle product of languages in automata theory. The reader is referred to appendix A for a definition of the shuffle product of general processes.

How to define shuffle product of random processes is explained next. Consider two random processes $\{\Omega^x, (\mathcal{F}_n^x), \mathbf{P}^x\}$ and $\{\Omega^y, (\mathcal{F}_n^y), \mathbf{P}^y\}$. By constructing the shuffle product $\{\Omega^x, (\mathcal{F}_n^x)\} \sqcup\!\sqcup \{\Omega^y, (\mathcal{F}_n^y)\}$ we introduce some interleaving of the two components. This interleaving is unknown to an observer monitoring the present occurrences of one of the components, joint observation of the two components is required for this interleaving to

be known: mathematically speaking, the interleaving is not $\mathcal{F}^x_\infty \otimes \mathcal{F}^y_\infty$-measurable. Hence, defining some probability on the space $\{\mathcal{W}, \mathcal{G}_\infty\}$ would require to randomize in some way the interleaving, something arbitrary we refuse to do. We rather proceed as follows:

$$\begin{aligned} \{\mathcal{W}, (\mathcal{G}_n)\} &= \{\Omega^x, (\mathcal{F}^x_n)\} \sqcup\!\sqcup \{\Omega^y, (\mathcal{F}^y_n)\} \\ \mathbf{Q} = \mathbf{P}^x \otimes \mathbf{P}^y & \qquad \text{is defined only on } \mathcal{F}^x_\infty \otimes \mathcal{F}^y_\infty \overset{\neq}{\subset} \mathcal{G}_\infty \end{aligned}$$

Hence we end up with a *partially random process*, i.e., a compound process where each component is random, but their interleaving is unknown but nonrandom. This is exactly the kind of mathematical object we specified in the first simple stochastic automaton. In general, partially random processes are tuples of the form $\{\Omega, (\mathcal{F}_n), (\mathcal{G}, \mathbf{P})\}$ where $\{\Omega, (\mathcal{F}_n)\}$ is a process, $\mathcal{G} \subseteq \mathcal{F}_\infty$, and $\mathbf{P}$ is a probability on $\mathcal{G}$. Partially random processes can be further combined via shuffle, see appendix A for a more general machinery allowing this.

4.1.3 Clocks, signals, specifying constraints, and performing conditioning

Consider a shuffle product $\{\mathcal{W}, (\mathcal{G}_n)\} = \{\Omega^x, (\mathcal{F}^x_n)\} \sqcup\!\sqcup \{\Omega^y, (\mathcal{F}^y_n)\}$. We can define the *clock* of x as the increasing sequence $0 = H^x_0(w) < H^x_1(w) < H^x_2(w) < \cdots$ of instants n where $x_n(w) \neq \perp$ holds, i.e., the x component is present. H^y is defined similarly. Clocks satisfy some kind of a *causality* condition which is fundamentally studied and exploited in [4], namely for instance:

$$\{w \in \mathcal{W} : H^x_k(w) = n\} \in \mathcal{G}_n \tag{18}$$

i.e., to decide whether H^x_k occurs at time n, it is enough to observe w up to time n. In conjunction with the clock H^x, we consider the *signal* $X(w) = (X_k(w))_{k>0}$ which is the sequence of values of the x component when it is present, so that $X_k(w)$ is available at time $H^x_k(w)$. More generally, a sequence $(H_k(w))_{k>0}$ of $\mathbf{N}$-valued functions satisfying the causality condition (18) will be called a *clock*. Thanks to (18), the following relation $\approx_{H_k}$

$$w_1 \approx_{H_k} w_2 \quad \overset{\text{by definition}}{\Leftrightarrow} \quad \begin{cases} H_k(w_1) = n \text{ and} \\ w_1(m) = w_2(m) \ \forall m \leq n \end{cases}$$

is an equivalence relation: its equivalence classes are called *initial segments of w of length H_k*, this generalizes the notion of initial segment to variable instants. A sequence $X(w) = (X_k(w))_{k>0}$ of functions such that $X_k(w)$ depends only on the initial segment of w of length $H_k(w)$ is termed a *signal with clock H*[9]. The present occurrences of the x component of a shuffle is an example of a signal, and so is $x = (x_n(\omega))$ in (14). How signals and clocks are *lifted* when their supporting processes are shuffled with other ones is discussed formally in appendix A. Expressing constraints on clocks and/or signals is the basic way to specify a subset of "legal" behaviours w: thus new processes can be specified from given ones by expressing constraints on clocks and/or signals. This is the way the semantics of SIGNAL is provided in [4], and SIGNAL itself can be used for such specifications.

[9] the reader who is familiar with probability theory may have recognized that clocks are chains of stopping times, while signals are sequences of $\mathcal{G}_{H_k}$-measurable variables

How constraints apply to *partially random* processes is a bit more involved. We are given some partially random process $\{\Omega,(\mathcal{F}_n),(\mathcal{G},\mathbf{P})\}$, where $\mathcal{G}\subset\mathcal{F}_\infty$ is some σ-algebra and $\mathbf{P}$ is a probability defined on $\mathcal{G}$. Next consider some subset $\mathcal{W}\subset\Omega$ of "legal" behaviours as mentioned above. The way we interpret the restriction of $\mathbf{P}$ to $\mathcal{W}$ is in accordance with the principle of conditioning in random fields: the resulting probability $\mathbf{Q}$ on $\mathcal{W}$ should be the conditional probability $\mathbf{P}(.|\mathcal{W})$ given $\mathcal{W}$. However, as in the Markov chain example, we must be careful in defining this since in general $\mathbf{P}(\mathcal{W})=0$ will hold so that the elementary Bayes rule does not apply to define conditional probability. This can be however easily overcomed when $\{\Omega,\mathcal{G},\mathbf{P}\}$ is the inductive limit of some sequence $(\{\Omega_n,\mathcal{G}_n,\mathbf{P}_n\})_{n>0}$ of probability spaces, and there exists, for each n, $\mathcal{W}_n\subset\Omega_n$ such that $\mathbf{P}_n(\mathcal{W}_n)>0$ holds and $\{\mathcal{W},\mathcal{G}\}$ is the inductive limit of $(\{\mathcal{W}_n,\mathcal{G}_n\})_{n>0}$ (this was typically the case in the Markov chain example, where the "$._n$" objects are associated with the observations up to time n). In this case, we can use the Bayes rule for each n and set

$$\mathbf{Q}_n(A_n)=\frac{\mathbf{P}_n(A_n\cap\mathcal{W}_n)}{\mathbf{P}_n(\mathcal{W}_n)}$$

Then $\{\mathcal{W},\mathcal{G},\mathbf{Q}\}$ is defined as the inductive limit of $(\{\mathcal{W}_n,\mathcal{G}_n,\mathbf{Q}_n\})_{n>0}$. It is shown in appendix B, proof of theorem 4.1, how conditioning is performed in general for partially random processes specified via SIGN*alea* programs.

4.1.4 The algebra of clocks and timers

We are given a process $\{\Omega,(\mathcal{F}_n)\}$ and a clock H. Clocks provide the successive dates of a sequence of events, we shall now introduce *timers*, which count how many such events occur during some time interval. The timer associated with H, we denote by μ^H, is defined by

$$\text{for } A\subset\mathbf{N}\ ,\ \mu^H(\omega,A)=\#\{k\ :\ H_k(\omega)\in A\}$$

where $\#\{\cdots\}$ denotes cardinal. The causality property of clocks is transferred to timers as follows:

$$\omega\mapsto\mu^H(\omega,A)\text{ is }\mathcal{F}_n\text{-measurable if }A\subseteq[0,n]$$

i.e., the timer needs to know only the initial segment of length n of ω to provide any information about subintervals of $[0,n]$. We define now the *filtering* of a clock by a boolean signal. Consider a boolean signal B with clock H. The filtering of H by B, written $H\downarrow B$, is the clock associated with the timer

$$\mu^{H\downarrow B}(\omega,A)=_{\text{def}}\ \mu^H(\omega,A\cap\{n:H_k(\omega)=n\text{ and }B_k(\omega)=true\}) \tag{19}$$

hence $H\downarrow B$ is selects those instants of H where the boolean signal B is true. The following $\wedge$ and $\vee$ operators are also useful:

$$\begin{aligned}\mu^{H\wedge K}(\omega,\{a\}) &= \mu^H(\omega,\{a\})\,.\,\mu^K(\omega,\{a\}) \\ \mu^{H\vee K}(\omega,\{a\}) &= \min\left\{\left(\mu^H(\omega,\{a\})+\mu^K(\omega,\{a\})\right),1\right\}\end{aligned} \tag{20}$$

Thus $\wedge$ and $\vee$ respectively denote the supremum (union of instants) and infimum (intersection of instants) of clocks. Formula (19) has the following converse:

$$K \wedge H = K \quad \Rightarrow \quad K = H \downarrow B \tag{21}$$

for some boolean signal B of clock H. Together with the $\wedge$ and $\vee$ of clocks, the $\downarrow$ operator provides the set of clocks defined on a given process with a nice structure which is extensively studied in [4]. Clock equations written using these operators are used to specify constraints on the set of legal behaviors of signals, SIGNAL programs can be used to perform this in practice.

4.1.5 Communication

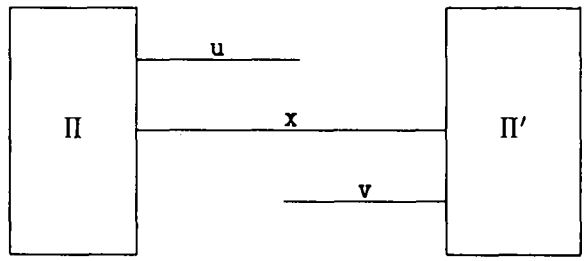
We are given some process $\Pi = \{\Omega, (\mathcal{F}_n)\}$. A *port* on Π is a triple $[\mathbf{x}, H, X]$ where H is some clock, X is a signal with clock H, and $\mathbf{x}$ is a name. Ports can be used to establish communication between processes in a fairly simple way. Given two processes Π and Π' with ports $[\mathbf{x}, H, X]$ and $[\mathbf{x'}, H', X']$ respectively, we define the *communication*

$$\Pi | \Pi' \tag{22}$$

as being

1. the shuffle product $\Pi \sqcup\!\sqcup \Pi'$, if the names $\mathbf{x}$ and $\mathbf{x'}$ are different,
2. the restriction of this shuffle product $\Pi \sqcup\!\sqcup \Pi'$ to those behaviours w such that $X(w) = X'(w), H(w) = H'(w)$ holds, otherwise.

This definition generalizes to processes with several ports. Hence the principle of the communication | is that ports with common names in the two components must carry identical clock and signal. This is depicted in the diagram below:



Communication can be extended to partially random processes. The idea is to construct the shuffle of the considered partially random processes, and then to consider that the constraints set by the communication result in taking the "conditional probability given these constraints". This is a bit a delicate programme since the above mentioned constraints generally define a set of zero probability, so that conditioning involves some technicalities that are discussed in appendix A.

4.1.6 Summary:

The tools we have at this point in our toolbox are recalled now:

- primitive processes corresponding to all possible S-valued behaviours where S is some domain,
- primitive random processes corresponding to S-valued i.i.d. random sequences with marginal distribution ν,
- shuffle products of such objects, which generally result in partially random processes, and then (recursively) shuffle products of partially random processes,
- clocks, signals, that can be used to specify via constraints a subset of "legal" behaviours of a given process, thus specifying by the way a new process; as far as probabilities are concerned, constraints are to be interpreted as performing conditioning,
- ports, and communication | of partially random processes, where ports with identical name are forced to possess identical clocks and signals.

As this summary shows, we need to extend the key notion of shuffle product to (partially random) processes that need not to be primitives. This requires a more abstract definition of the shuffle which is presented in the appendix A.

4.2 The semantics of SIGNalea

In this section, we provide a precise mathematical meaning for the kind of SIGNalea program we wrote. This will be done by giving a precise translation of the instructions `(i)--(vi)` of SIGNalea in terms of the mathematical objects we introduced in subsection 4.1.

Notations

Our mathematical model is presented in the following form:

$$\texttt{program} :: \begin{bmatrix} \text{process} \\ \text{potential} \\ \text{constraints} \end{bmatrix}$$

where

- `program` denotes some SIGNalea program;
- "::" indicates that the array on the right handside is the mathematical model associated with the program mentioned on the left handside;

- "process" refers to some process $\Pi = \{\Omega, (\mathcal{F}_n), \texttt{list of ports}\}$ equipped with some ports, we provide later adequate syntax to specify such processes;
- the field "potential" describes some (static) energy function, say U, with $\nu = \frac{1}{Z}\, e^{-U}$ as associated Gibbs measure; this specifies some probability $\mathbf{P} = \nu^{\otimes \mathbf{N}}$ defined on the space $\{\Omega, \mathcal{G}\}$, where $\mathcal{G} \subseteq \mathcal{F}_\infty$. Again will appropriate syntax be provided to specify them; note that the pair {process, potential} together build the "partially randomized process" we introduced before;
- "constraints" refers to a list of constraints specifying the restriction of the above mentioned partially randomized space to those trajectories which satisfy the required constraints; applying such constraints results in specifying the conditional probability law specified in the "potential" field, given the constraints; how these constraints are specified will be introduced later.

We shall also write $\mathcal{S}(\texttt{program})$ to refer to the model of the right handside.

Canonical process associated with a port. For a port name $\mathbf{x}$ with domain S, we set

$$\begin{array}{rcl} \Pi\{\mathbf{x}\} & =_{\text{def}} & \{\Omega, (\mathcal{F}_n), [\mathbf{x}, Id, x]\}, \text{ where} \\ \Omega & = & \mathbf{N} \mapsto S \ , \text{ we write } x_n(\omega) =_{\text{def}} \ \omega(n) \ , \ x = (x_n)_{n \in \mathbf{N}} \\ \mathcal{F}_n & = & \sigma\{x_m \ : \ m \leq n\} \\ [\mathbf{x}, Id, x] & \text{is} & \text{a port with name } \mathbf{x}, \text{ clock } Id \text{ and signal } x \end{array}$$

Recall that Id denote the "identity" clock defined by $Id_n(\omega) = n$. No confusion should result from using the same label to refer to running points of the data types and to signals. Thus $\Pi\{\mathbf{x}\}$ is the set of all possible behaviours of S-valued sequences.

Canonical process associated with a set of ports.

$$\Pi\{\mathbf{x}_1, ..., \mathbf{x}_p\} =_{\text{def}} \ \Pi\{\mathbf{x}_1\} \sqcup\!\sqcup \cdots \sqcup\!\sqcup \Pi\{\mathbf{x}_p\}$$

Canonical random space associated with a set of ports. We use the notation

$$U(\mathbf{x}_1, ..., \mathbf{x}_p)$$

to refer to the probability

$$\left(\frac{1}{Z} e^{-U(x_1, ..., x_p)}\right)^{\otimes \mathbf{N}}$$

defined on the space $\Pi\{\mathbf{x}_1, ..., \mathbf{x}_p\}$ where the constraint $H(\mathbf{x}_1) = \cdots = H(\mathbf{x}_p)$ is enforced.

Specifying constraints. We have to specify constraints involving clocks and signals. Constraints involving clocks are expressed with the operators "$\wedge, \vee, \downarrow$" we introduced before. It will be useful to write generically $H(\mathbf{x})$ to refer to the clock of the port $\mathbf{x}$ or $H(x)$ to refer to the clock of the signal x. Then, given two ports $[\mathbf{x}, H, x]$ and $[\mathbf{x}', H', x']$, we shall write

$$K : R(x, x') \tag{23}$$

where K denotes a clock satisfying $K \subseteq H \wedge H'$ and $R(.,.)$ is some relation, to mention that

$$\forall(m,n,p): \quad H_m(\omega) = H'_n(\omega) = K_p(\omega) \;\Rightarrow\; R(x_m(\omega), x'_n(\omega)) \text{ holds}$$

This may be rephrazed: "$R(x, x')$ holds at clock K".

The semantics

Instruction (i)

$$\mathtt{R}(\mathbf{x}_1, ..., \mathbf{x}_p) :: \left[\begin{array}{c} \Pi\{\mathbf{x}_1, ..., \mathbf{x}_p\} \\ \\ \text{none} \\ \\ H(x_1) = ... = H(x_p) \\ Id \;:\; R(x_1, ..., x_p) \end{array}\right]$$

Instruction (ip)

$$\mathtt{potential\ U}(\mathbf{x}_1, ..., \mathbf{x}_p) :: \left[\begin{array}{c} \Pi\{\mathbf{x}_1, ..., \mathbf{x}_p\} \\ \\ U(\mathbf{x}_1, ..., \mathbf{x}_p) \\ \\ H(x_1) = ... = H(x_p) \end{array}\right]$$

Instruction (ii)

$$\mathtt{y := x\ \$1\ init\ x0} :: \left[\begin{array}{c} \Pi\{\mathbf{x}, \mathbf{y}\} \\ \\ \text{none} \\ \\ H(y) = H(x) \\ y_n(\omega) = x_{n-1}(\omega)\ ,\ y_0(\omega) = \mathbf{x0} \end{array}\right]$$

Instruction (iii)

$$\mathtt{y := x\ when\ b} :: \left[\begin{array}{c} \Pi\{\mathbf{x}, \mathbf{b}, \mathbf{y}\} \\ \\ \text{none} \\ \\ H(y) = H(x) \wedge (H(b) \downarrow b) \\ H(y) \;:\; y = x \end{array}\right]$$

Instruction (iv)

$$\texttt{y := u default v} :: \left[\begin{array}{c} \Pi\{\texttt{u},\texttt{v},\texttt{y}\} \\ \\ \text{none} \\ \\ H(y) = H(u) \vee H(v) \\ H(u) \ : \ y = u \\ H(v) \ominus H(u) \ : \ y = v \end{array}\right]$$

where $H(v) \ominus H(u)$ denotes the clock such that $(H(v) \ominus H(u)) \vee (H(v) \wedge H(u)) = H(v)$.

Instruction (v)

$$\mathcal{S}(\texttt{P|Q}) = \mathcal{S}(\texttt{P})|\mathcal{S}(\texttt{Q})$$

in the sense of the section 4.1.

Instruction (vi)

$$\texttt{P} \ \{\texttt{x1},...,\texttt{xp}\} :: \text{ remove all but } \texttt{x1},\ldots,\texttt{xp} \text{ ports in } \mathcal{S}(P)$$

Due to the coding of instruction (v), this semantics is rather abstract and not useful to develop the effective formal calculi which will be involved in the SIGN*alea* compiler. In the next subsection, an alternative *canonical* form of the semantics of a SIGN*alea* program will be presented which will be more concrete and useful.

4.3 A canonical form

We first investigate on the small SIGN*alea* programs how we would like this canonical form to look like. Then we introduce the canonical form mathematically.

4.3.1 Guessing the semantics of some simple example

A simple stochastic automaton. It has been claimed that the following program specifies some kind of a stochastic automaton:

```
(| given z potential U(x,z)
 | potential V(a)
 | z := a default zz
 | zz := z $1
 |)
```

The second instruction specifies an i.i.d. sequence (a_n) with marginal distribution

$$\frac{1}{Z} e^{-U(a)}$$

Next, considering the last two instructions yields the (nonrandomized) process

$$\left[\begin{array}{c} \Pi\{\mathbf{a},\mathbf{z},\mathbf{zz}\} \\ \\ \text{none} \\ \\ H(z) = H(zz) = H(a) \vee H(z) \\ H(a) : z = a \ , \ H(z) \ominus H(a) : z = zz \\ H(z) : zz_n = z_{n-1} \end{array}\right]$$

Then, combining the last two instructions with the second one yields the (partially randomized) process

$$\left[\begin{array}{c} \Pi\{\mathbf{a},\mathbf{z},\mathbf{zz}\} \\ \\ U(\mathbf{a}) \\ \\ H(z) = H(zz) = H(a) \vee H(z) \\ H(a) : z = a \ , \ H(z) \ominus H(a) : z = zz \\ H(z) : zz_n = z_{n-1} \end{array}\right]$$

Finally, taking into account the first instruction yields

$$\left[\begin{array}{ll} (i) & \Pi\{\mathbf{a},\mathbf{z},\mathbf{zz},\mathbf{x}\} \\ \\ (ii) & U(\mathbf{a}) \\ (iii) & U_z(\mathbf{x},\mathbf{z}) \\ \\ (iv) & H(z) = H(zz) = H(a) \vee H(z) \\ (v) & H(a) : z = a \ , \ H(z) \ominus H(a) : z = zz \\ (vi) & H(z) : zz_n = z_{n-1} \\ (vii) & H(x) = H(z) \end{array}\right] \tag{24}$$

Formula (i) specifies the process which the semantics of this program is based on. Formula (ii) specifies that **a** is i.i.d.: since no further constraint result on **a** from equations $(iv)-(vii)$, this is the actual distribution of **a** as resulting from the global program. Next, formula (vii) asserts that **x** and **z** have the same clock. Then equation (iv) specifies that **z** has to be available at least when **a** is, but can also be more frequently available in some unspecified (but nonrandom) way. Finally, formulae (v, vi) express that the value of **z** is that of **a** when the latter occurs, and is unchanged otherwise. Finally, formula (iii) can be thought of as specifying how **x** behaves via its conditional distribution given **z**: the notation $U_z(\mathbf{x},\mathbf{z})$ refers to the potential U compensated as in formula (13). This yields a simple stochastic automaton, as claimed. The clocks $H(z)$ and $H(a)$ are related only via the constraint (iv) (an "inequality" constraint), whence the clock of **z** is not entirely determined by that of the input **a**: this is how nondeterminism is exhibited in the semantics.

It is interesting to notice that we have carefully checked how the dependencies are oriented. But actual dependencies vary with time (they depend on whether **a** is present or

not), hence some kind of *Conditional Dependency Graph* must be considered, as in SIGNAL, see [4]. We shall see later how to perform this. It is also interesting to note that we may want to combine formulae (ii, iii) into the single following one

$$(ii, iii) \quad \mu^a U(\mathbf{a}) + U_z(\mathbf{x}, \mathbf{z})$$

where the expression $\mu^a U(\mathbf{a}) + U_z(\mathbf{x}, \mathbf{z})$ denotes an energy function which is the sum of two local components: $U_z(\mathbf{x}, \mathbf{z})$, and $\mu^a U(\mathbf{a})$. The latter one indicates that the local potential $U(\mathbf{a})$ has to be *modulated* by the timer μ^a, i.e. is taken into account only when $\mathbf{a}$ is present. We shall formalize and generalize this procedure later on.

Discussion

The reader may have guessed that we somewhat cheated in presenting this semantics. We properly used the models of instructions (i-iv), but we did not use the semantics of the communication | as stated in the section 4.2. Instead we have combined the basic canonical spaces via simple shuffle operators, then we have performed effective communication by specifying systems of constraints, and finally we specified the random behaviour via "potentials modulated by timers". The next subsection is devoted to a formal study to support this intuitive and effective approach. In particular it is needed to show that the conditional distributions that are specified via the pair {modulated potentials, constraints} are properly defined.

4.3.2 Getting the canonical form

The following results are proved in appendix B. Consider first the instruction

```
(| potential U(x) | potential V(y) |)
```

The associated process is the shuffle $\Pi\{\mathbf{x}, \mathbf{y}\}$ of the primitives $\Pi\{\mathbf{x}\}$ and $\Pi\{\mathbf{y}\}$ as defined in paragraph 4.1.2. The formula

$$\tilde{\mathbf{Q}}_{|\mathcal{G}_n}(w(1), ..., w(n)) = \frac{1}{Z_n} e^{-\sum_{i=1}^{n} W(w(i))} \tag{25}$$

$$W(w(n)) = \mathbf{1}_{\{x_n(w) \neq \perp\}} U(x_n(w)) + \mathbf{1}_{\{y_n(w) \neq \perp\}} V(y_n(w)) \tag{26}$$

equip $\Pi\{\mathbf{x}, \mathbf{y}\}$ with a probability $\tilde{\mathbf{Q}}$ making $(w(n))_{n \in \mathbf{N}}$ to behave as an i.i.d. sequence with distribution proportional to e^{-W}. Note that $U(x_n(w))$ is well defined at those instants n where $x_n(w)$ is present, so that (26) is meaningful. We shall use the notation

$$\mu^x U(\mathbf{x}) + \mu^y V(\mathbf{y}) \tag{27}$$

to refer to the potential W as defined in (26), i.e., $\mu^x = 1$ if x is present at the considered instant, and 0 otherwise, so that μ^x is the timer counting the occurrences of x in the shuffle. The following result holds:

Proposition 4.1 (representing shuffles) .

1. *The following formula holds*

$$(|\texttt{potential U(x)}|\texttt{potential V(y)}|) :: \left[\begin{array}{c} \Pi\{\mathtt{x},\mathtt{y}\} \\ \mu^x U(\mathtt{x}) + \mu^y V(\mathtt{y}) \\ none \end{array}\right]$$

2. *Only the restriction of $\tilde{\mathbf{Q}}$ to the σ-algebra $\mathcal{F}^x_\infty \otimes \mathcal{F}^y_\infty$ we shall denote by $\mathbf{Q}$ has to be considered and it holds that*

$$\mathbf{Q} \;=\; \mathbf{P}^x \otimes \mathbf{P}^y \quad \textit{on the } \sigma\textit{-algebra } \mathcal{F}^x_\infty \otimes \mathcal{F}^y_\infty \tag{28}$$

This proposition generalizes immediately to several such instructions.

Statement 2. expresses that the restriction of $\tilde{\mathbf{Q}}$ to $\mathcal{F}^x_\infty \otimes \mathcal{F}^y_\infty$ (i.e., ignoring the interleaving of the two components) equals $\mathbf{P}^x \otimes \mathbf{P}^y$, hence $\mathbf{Q}$ is the desired distribution on the shuffle, as defined in the paragraph 4.1.2. Hence only point 2. needs to be proved to get proposition 4.1, this is done in appendix B. The following result is essential in getting equivalences of SIGN*alea* programs:

Proposition 4.2 (combining synchronized potentials) *The following formula holds true:*

$$(|\texttt{potential U(x)}|\texttt{potential V(y)}|\texttt{x\^{}=y}|) \equiv \texttt{potential U(x)+V(y)} :: \left[\begin{array}{c} \Pi\{\mathtt{x},\mathtt{y}\} \\ U(\mathtt{x}) + V(\mathtt{y}) \\ H(\mathtt{x}) = H(\mathtt{y}) \end{array}\right]$$

In particular, it expresses that, when considered alone, the statement "`potential U(x)+V(y)`" does not cause any mutual dependence between `x` and `y`; this can also be verified by recalling that $e^{-(U(x)+V(y))} = e^{-U(x)}e^{-V(y)}$ so that x and y remain independent signals. This will be taken into account when associating labelled graphs with SIGN*alea* programs in the following section. The following result provides the desired canonical form:

Theorem 4.1 (canonical form using potentials) *Every* SIGN*alea program* $\mathtt{P}\{\mathtt{x}_1,...,\mathtt{x}_p;\mathtt{y}_1,...,\mathtt{y}_q\}$ *with inputs* $\{\mathtt{x}_1,...,\mathtt{x}_p\}$ *and outputs* $\{\mathtt{y}_1,...,\mathtt{y}_q\}$ *can be expressed as follows:*

$$\mathtt{P}\{\mathtt{x}_1,...,\mathtt{x}_p;\mathtt{y}_1,...,\mathtt{y}_q\} :: \left[\begin{array}{c} \Pi\{\mathtt{x}_1,...,\mathtt{x}_p;\mathtt{y}_1,...,\mathtt{y}_q\} \\ U(\mathtt{x}_1,...,\mathtt{x}_p) = \sum_{i=1}^{I} \mu^i U_i(\mathtt{x}^i_1,...,\mathtt{x}^i_{p_i}) \\ C(\mathtt{x}_1,...,\mathtt{x}_p;\mathtt{y}_1,...,\mathtt{y}_q) \end{array}\right]$$

where $\mu^i = 1$ when the $\mathtt{x}^i_1,...,\mathtt{x}^i_{p_i}$'s are present and $= 0$ otherwise, and C denotes some constraint involving the listed signals and their clocks.

In fact, μ^i is just the timer associated with the common clock of the signals $\mathbf{x}_1^i, ..., \mathbf{x}_{p_i}^i$, so that $U(\mathbf{x}_1, ..., \mathbf{x}_p)$ is the sum of potentials modulated by timers. Thanks to this formula, conditioning is performed using potentials in the very same way it was done in (5,6) for elementary random fields, see appendix B for details. This canonical form can be built in a mudular way, as stated in the following immediate corollary of the preceding theorem:

Corollary 4.1 (an explicit formula for the composition |) *The following alternative coding of the composition | holds:*

$$\textit{if}\quad \mathtt{P} :: \begin{bmatrix} \Pi \\ U \\ C \end{bmatrix} \quad , \quad \mathtt{Q} :: \begin{bmatrix} \Pi' \\ U' \\ C' \end{bmatrix} \quad \textit{then} \quad \mathtt{P|Q} :: \begin{bmatrix} \Pi \sqcup \Pi' \\ \mu^{\Pi}.U + \mu^{\Pi'}.U' \\ C \cup C' \end{bmatrix}$$

holds.

In these formulae, μ^{Π} refers to the counter associated with the clock of the events in the composition in which Π is involved; $\mu^{\Pi}.U$ denotes the energy U *modulated by* μ^{Π}, which means that U is considered only at those events Π is involved in. Then $C \cup C'$ denotes the conjunction of both constraints C and C'.

4.3.3 Back to the stochastic automaton example

Theorem 4.1 and its corollary justifies the semantics we provided to the stochastic automaton example in section 4.3.1. In particular, the inequality *(iv)* in (24) expresses that $H(z)$ is not entirely determined by $H(a)$. So if a alone is to be considered as the input (as probably wished), the interleaving is *unknown* and not randomized.

Next consider again the program of the stochastic automaton with reading times distributed according to a Poisson process:

```
(|(| given z potential U(x,z)
   | potential V(a)
   | z := a default zz
   | zz := z $1
   |)
 |(| x ^= c
   |)
 |(| potential if c=true then 1 else p fi
   | a ^= (true when c)
   |)
 |)
```

Its semantics is derived in the same way:

$$\left[\begin{array}{ll} (i) & \Pi\{\mathtt{a},\mathtt{c},\mathtt{z},\mathtt{zz},\mathtt{x}\} \\ (ii) & \mu^a U(\mathtt{a}) + W(\mathtt{c}) + U_z(\mathtt{x},\mathtt{z}) \\ (iii) & H(c) = H(z) = H(zz) = H(a) \vee H(z) \\ (iv) & H(a) = H(c) \downarrow c \\ (v) & H(a) : z = a \ , \ H(z) \ominus H(a) : z = zz \\ (vi) & H(z) : zz_n = z_{n-1} \\ (vii) & H(x) = H(z) \end{array}\right]$$

The constraints relating clocks are (iii, iv, vii), they can be equivalently written as

$$\left[\begin{array}{ll} (iii) & H(z) = H(zz) = H(x) = H(c) \\ (iv) & H(a) = H(c) \downarrow c \end{array}\right]$$

so that all clocks are determined knowing the two inputs `a,c`. Hence no unspecified interleaving remains any more in this case, so that the potential (ii) as conditioned via the constraints $(iii, \cdots, vii)$ exactly specifies the distribution of the automaton.

5 Expressive power of SIGN*alea*

It is difficult to characterize exactly this expressive power, we shall rather show examples of wellknown formalisms that can be embedded in SIGN*alea*.

Theorem 5.1 *Every stochastic Büchi automaton*[10] *can be specified in* SIGN*alea*.

Proof. A semi-Markov chain is a stochastic process (y_n) defined as follows: there exists some (hidden) Markov chain (x_n) such that

$$\mathbf{E}[y_n | y_{n-1}, y_{n-2}, \cdots; x_{n-1}, x_{n-2}, \cdots] = \mathbf{E}[y_n | x_{n-1}]$$

An homogeneous semi-Markov chain can be specified by the following SIGN*alea* program:

```
(| given z potential U(y,z)
 | given z potential V(z,x)
 | z := x $1 init x0
 |)
```

Here, `x` is the Markov chain, `z` its delayed value, and the potential `U` specifies the conditional distribution of `y` given `z`. If it is wanted that (say) this latter conditional distribution be time varying, just replace the first instruction by the following program:

[10] or, equivalently, every semi-Markov chain, or every Hidden Markov Model, depending on the area where these objects are referred to.

```
(| given z potential U(y,z,n)
 | n := zn + 1
 | zn := n $1 init 0
 |)
```

By the first instruction, the signals `y,z,n` have the same clock, and `n` is an increasing counter which thus counts the occurrences of `y,z`, so that it represents the time. Then `U` is now a function of `n`.

Theorem 5.2 *Every stochastic timed Petri net [20] can be specified in* SIGN*alea.*

Proof: see appendix C.

Summary

SIGN*alea* allows us to specify

- stochastic automata, and their interconnection via down- and upsampling,
- Petri net-like asynchronous (random) systems such as widely used in manufacturing systems or queuing networks,
- random fields on graphs (by the very definition of SIGN*alea*),

and the combination of such types of systems. The next section is devoted to simulation and compilation.

6 Compilation and simulation

In this section we show how the canonical form of a SIGN*alea* program can be exploited to perform simulation using local interactions only. From this analysis the notion of compilation of a SIGN*alea* program will emerge.

6.1 Examples continued

The Markov chain. Recall this program:

```
(| given y potential U(y,x)
 | y := x $1 init x0
 |)
```

It specifies a Markov chain with distribution up to instant n proportional to $e^{-\sum_{i=1}^{n} U_x(x_{i-1},x_i)}$. It is wellknown that this joint distribution can be factorized using Bayes rule, which is the very basic reason for Markov chains being so useful. How could we have guessed this by direct checking of the SIGN*alea* program? The first instruction, 1/ builds a graph with `x`

and y as vertices, 2/ specifies a Gibbs random field on this graph, 3/ extends it to an i.i.d. sequence. From this we remember that the first instruction created a graph

$$\mathtt{y} \longrightarrow \mathtt{x}$$

We consider here a *directed* branch, although random fields are associated with nondirected graphs: this is because we interpret the "`given ...`" statement as an intension of the programmer to specify a conditional distribution. The second instruction "`y := x$1`" causes a dependency from x to y which is due to causality from past to future. Denote this as follows:

$$\mathtt{x} \dashrightarrow \mathtt{y}$$

Combining both graphs yields

$$\mathtt{x} \dashrightarrow \mathtt{y} \longrightarrow \mathtt{x}$$

Unfolding this directed graph yields the way Bayes rule has to be implemented

$$\cdots \longrightarrow \mathtt{x}_{n-1} = \mathtt{y}_n \longrightarrow \mathtt{x}_n = \mathtt{y}_{n+1} \longrightarrow \cdots$$

and simulation is straightforward using only the transition probability of the Markov chain.

A simple stochastic automaton. Recall this program:

```
(| given z potential U(x,z)
 | potential V(a)
 | z := a default zz
 | zz := z $1
 |)
```

We outline how the preceding argument can be generalized to the present more complex case. The new point here is that two different clocks are involved, namely $H(\mathtt{a})$ and $H(\mathtt{z})$ that are related via

$$H(\mathtt{z}) = H(\mathtt{z}) \vee H(\mathtt{a})$$

The graph coding of this program is the following (each line corresponds to an instruction):

$$\begin{array}{rcl} H(\mathtt{z}) = H(\mathtt{x}) & ; & \mathtt{z} \longrightarrow \mathtt{x} \\ \emptyset & ; & \emptyset \\ H(\mathtt{z}) = H(\mathtt{z}) \vee H(\mathtt{a}) & ; & \mathtt{a} \longrightarrow \mathtt{z} \xleftarrow{H(\mathtt{z}) \ominus H(\mathtt{a})} \mathtt{zz} \\ H(\mathtt{zz}) = H(\mathtt{z}) & ; & \mathtt{z} \dashrightarrow \mathtt{zz} \end{array}$$

where we recall that $H(\mathtt{z}) \ominus H(\mathtt{a})$ denotes the clock such that $(H(\mathtt{z}) \ominus H(\mathtt{a})) \vee (H(\mathtt{z}) \wedge H(\mathtt{a})) = H(\mathtt{z})$. Since several clocks are involved, we must indicate precisely the meaning of the branches, possibly labelled by clocks. By definition, each branch is effective at the infimum of the label and the clocks of their extremities: for instance, branch $\mathtt{a} \longrightarrow \mathtt{z}$ is effective at clock $H(\mathtt{z}) \wedge H(\mathtt{a}) = H(\mathtt{a})$, while branch $\mathtt{z} \xleftarrow{H(\mathtt{z}) \ominus H(\mathtt{a})} \mathtt{zz}$ is effective at clock $(H(\mathtt{z}) \ominus H(\mathtt{a})) \vee H(\mathtt{z}) = H(\mathtt{z}) \ominus H(\mathtt{a})$. Unfolding the resulting global directed graph yields

the way Bayes rule has to be implemented. From this follows how simulation is implemented using only "small" conditional distributions. In the present case, the branch $\mathbf{z} \longrightarrow \mathbf{x}$ holds effective when $\mathbf{z}$ is present, i.e., always, but the distribution of $\mathbf{a}$ is called for only when $\mathbf{a}$ is present. Finally, we must recall that this SIGN*alea* program specifies a partially randomized dynamical system.

The Hopfield model in neural networks. This model is also known as *Ising* or *spin-glass* model in statistical mechanics [13]. Let us discuss a simple version of it. We are given a square grid $V = \{1, \cdots, P\} \times \{1, \cdots, P\}$, the grid itself defines the considered graph (thus we have an "image"). Each site (or pixel) is labelled by some *spin*, i.e., some ± 1-valued variable we denote by $x[i,j]$. A family $\omega = \{x[i,j]\}_{i,j \in \{1,\cdots,P\}}$ is called a *configuration* (we may as well interpret it as a black-and-white image). Then a local quadratic energy function is associated with a given configuration via the following formula:

$$U(\omega) = - \sum_{|i-i'|+|j-j'|=1} x[i,j]x[i',j'] - \lambda \sum_{i,j} x[i,j]x_o[i,j] \tag{29}$$

where λ is some nonnegative constant parameter. In the righthand side of (29), the first term penalizes chaotic configurations, while the second one penalizes the discrepancy between the actual configuration ω and some desired ideal reference ω_o. Such a model may be specified with SIGN*alea* as follows:

```
array i to P-1 of
   array j to P-1 of
      (| potential - x[i,j] xr[i,j] - x[i,j] xd[i,j] - lambda x[i,j] xo[i,j]
       | xr[i,j] := x[i,j+1]
       | xd[i,j] := x[i+1,j]
       |)
   end
end
```

This program specifies an i.i.d. sequence of variables distributed according to the energy (29). The graph associated with this program coincides with the original 2-dimensional grid. A possible consistent orientation of this graph consists of drawing the grid with right- and downgoing arrows. However, since "`potential`" instruction have been used rather than "`given ... potential`" ones, no Bayes rule with local conditional probabilities emerges, cf. the discussion in section 2. In fact, any attempt to implement a Bayes rule according to the above mentioned orientation of the graph would result in computing normalizing factors that are of order 2^{P^2} complexity, so that no real advantage would result as compared to a global simulation! Such a situation is easily recognized by checking for the size of the strongly connected *nondirected* components of the graph associated with the considered program: in the present case, this size was P^2.

For the above discussed reasons it is in general preferred to perform the simulation of Hopfield models using *Gibbs sampling,* also known as "stochastic relaxation" or Hopfield dynamics, it is beyond the scope of the present paper to justify this method, we refer the

reader to [13, 2]. A SIGN*alea* program performing Gibbs sampling with periodic flipping of the spins is the following:

```
(|(|(| i := (1 when (zi=P-1)) default zi+1
     | zi := i $1 init 0
     |)
   |(| j := (1 when (zj=P-1)) default zj+1
     | zj := j $1 init 0
     |)
   |)
 |(| potential if Z>0 then Z else infinity          % Z a real variable
   |)
 |(| zx[i,j] := x[i,j] $1 init 1
   | zU[i,j] := - zx[i,j] zx[i,j+1] - zx[i,j] zx[i+1,j]
               - zx[i,j] zx[i,j-1] - zx[i,j] zx[i-1,j]
               - lambda zx[i,j] xo[i,j]
   | nU[i,j] := - zU[i,j]
   | dU[i,j] := nU[i,j] - zU[i,j]
   |)
 |(| x[i,j] := (- zx[i,j] when (dU[i,j] < Z)) default zx[i,j]
   |)
 |)
```

This program consists of four blocks. The first block specifies how the spin to be flipped (pair `[i,j]`) is selected: this is done here in a periodic way (a uniformly distributed random choice could be used as well). The second block specifies an i.i.d. sequence of random variables with exponential distribution proportional to $e^{-z}1_{\{z>0\}}$. The third block computes 1/ the local interaction `zU[i,j]` at `[i,j]` before a tentative flipping, 2/ the result `nU[i,j]` of flipping only spin `[i,j]`, 3/ the difference `dU[i,j]` in the energy that would result if flipping of the considered spin was performed. Then the fourth block accepts or refuses the flipping by comparing the tentative energy difference to the (random) threshold `Z`. It is shown in [13, 21, 2] that the invariant probability of the Markov chain on the grid which is specified by this "Gibbs sampler" program has precisely U given in (29) as energy function. To draw some conclusions from this example,

1. SIGN*alea* is able to detect if simulation can be performed via Bayes rule using local characteristics only (we have seen that is was not the case here),

2. SIGN*alea* can then be used to specify the Gibbs sampler which involves local computations only.

Such Gibbs samplers are the basis for Monte-Carlo simulation in statistical mechanics.

SIGNAL equation	clock calculus	graph
`x ^= y`	$H(x) = H(y)$	none
`y := f(u,v)`	$H(u) = H(v) = H(y)$	u ⟶ y ⟵ v
`y := x $1`	$H(y) = H(x)$	x ⇢ y
`y := x when b`	$H(y) = H(x) \wedge (H(b) \downarrow b)$	x $\xrightarrow{H(b)\downarrow b}$ y ⟵ b
`y := u default v`	$H(y) = H(u) \vee H(v)$	u ⟶ y $\xleftarrow{H(v)\ominus H(u)}$ v
`P|Q`	clock(P) ∪ clock(Q)	graph(P) ∪ graph(Q)

Table 1: Encoding SIGNAL programs

6.2 Compilation: formal results

Encoding SIGNAL programs

We present briefly how SIGNAL programs may be encoded using "clock calculi" and "conditional graphs", i.e., graphs that are labelled by clocks. For this we rely here on the formalism developed in the present papers (our clock algebra). An alternative formalism using dynamical systems over Galois fields is presented in [15, 5, 14] which is more powerful in analyzing or synthesizing synchronization mechanisms. This coding is shown in table 1, the notations used there were already introduced in the preceding subsection. In this table and in the subsequent ones, the notation

$$\mathtt{x} \xrightarrow{H} \mathtt{y}$$

is to be interpreted as "y *depends on* x *at those instants of the clock* $H(\mathtt{x}) \wedge H(\mathtt{y}) \wedge H$".

Encoding SIGNalea programs

This coding is shown in table 2. We consider that "`given x potential U(x,y)`" statements express the intension of the programmer of referring to y as distributed conditionnally to x, this explains why a directed arrow is used in this case. On the other hand, when

SIGN*alea* equation	clock calculus	graph
`potential U(x,y)`	$H(x) = H(y)$	x —— y
`given x potential U(x,y)`	$H(x) = H(y)$	x ⟶ y
`P|Q`	clock(P) ∪ clock(Q)	graph(P) ∪ graph(Q)

Table 2: Encoding SIGN*alea* programs

building the nondirected branches of the graph, we first replace each additively decomposed potential (say, `potential U(x)+V(y)`) by the communication of its components (namely `(| potential U(x) | potential V(y) | x^=y |)`), cf. the remark following proposition 4.2. It will be important for the sequel *not* to take the transitive closure of the so obtained graph.

Some useful notions

Throughout this paragraph we are considering a given SIGN*alea* program P with associated clock calculus "clock(P)" and graph "graph(P)" according to the rules of tables 1 and 2.

The clock of a path: for a path

$$[x_0, x_k] = \; x_0 \xrightarrow{H_1} x_1 \xrightarrow{H_2} x_2 \cdots x_{k-1} \xrightarrow{H_k} x_k$$

we define its *clock* as follows

$$H([x_0, x_k]) =_{\text{def}} \; H(x_0) \wedge \left(\bigwedge_{i=1}^{k} H(x_i) \right) \wedge \left(\bigwedge_{i=1}^{k} H_i \right) \tag{30}$$

This is the clock of the instants where all branches are simultaneously present.

Clusters: select in graph(P) those branches that are nondirected, this amounts to selecting the random components of P. The resulting graph is then partitioned into strongly connected components we call *clusters:* thanks to table 2, all vertices of a given cluster have the same clock. Bayes rule cannot be used to simulate clusters with local characteristics as we discussed above[11], so we handle them globally. This is achieved by *taking the transitive*

[11] this can be however achieved by relying on *Gibbs sampling* as we have seen in the Hopfield model example; but it is our opinion that Gibbs sampling or stochastic relaxation should not be applied automatically as a part of the compilation process, since there are many different ways to implement them and expertise of the user should be called for in this case.

closure of clusters within graph(P).

Directed graph associated with a program: given the graph "graph(P)" associated with P, we will call a *directed graph associated with* P any graph $\mathcal{DG}$(P) obtained by assigning some direction to those branches that are nondirected in graph(P) in such a way that the following condition be satisfied:

$$\begin{array}{c} x_0 \xrightarrow{H_1} x_1 \text{ —— } x_2 \in \text{graph(P)} \\ \Downarrow \\ x_0 \xrightarrow{H_1} x_1 \longrightarrow x_2 \in \mathcal{DG}(P) \end{array} \tag{31}$$

This condition ensures that no multiple definition arises from assigning a wrong direction to some nondirected path due to a "`potential`" statement.

Some results

The following result holds, which is the key result of the present paper:

Theorem 6.1 (fundamental result about compilation) *Denote by $\mathcal{DG}$(P) some directed graph associated with* P.

1. *We say that $\mathcal{DG}$(P) is a compiled form of* P *if the clock of every circuit in $\mathcal{DG}$(P) is zero.*
2. *Furthermore, if all clocks are uniquely determined from input clocks, then no unconstrained nonrandom behaviour occurs (the associated process is -totally- random).*
3. *In this case, $\mathcal{DG}$(P) provides a possible way to perform computations and simulate* P *based on local interactions only.*

The proof of this theorem is identical to that of the fundamental theorem in [6].

6.3 Back to the examples

In this subsection we apply the rules of compilation and simulation to rederive how the stochastic automaton example can be simulated using local characteristics only. Recall this program:

```
(| given z potential U(x,z)
 | potential V(a)
 | z := a default zz
 | zz := z $1
 |)
```

Its encoding is shown in table 3. This coding yields

SIGN*alea* instruction	clock calculus	graph
`given z potential U(x,z)`	$H(x) = H(z)$	z ⟶ x
`potential V(a)`	none	none
`z := a default zz`	$H(z) = H(a) \vee H(zz)$	a ⟶ z $\xleftarrow{H(zz) \ominus H(a)}$ zz
`zz := z $1`	$H(zz) = H(z)$	z ⇢ zz

Table 3: Encoding the simple stochastic automaton

1. a clock calculus which can be put in the form

$$H(x) = H(z) = H(zz)\ ,\ H(x) = H(a) \vee H(x) \tag{32}$$

2. a graph $\mathcal{DG}$. Add to this graph the branches of the form

$$H(\text{signal}) \longrightarrow \text{signal}$$

for any signal, this yields the *Conditional Dependency Graph* introduced in [5, 6, 15], denote it by $\mathcal{CDG}$: it consists of the branches:

$$H(\mathtt{x}) \longrightarrow \mathtt{z} \longrightarrow \mathtt{x}\ ,\ \mathtt{z} \dashrightarrow \mathtt{zz} \tag{33}$$

$$H(\mathtt{a}) \longrightarrow \mathtt{a} \longrightarrow \mathtt{z} \xleftarrow{H(\mathtt{zz}) \ominus H(\mathtt{a})} \mathtt{zz} \longleftarrow H(\mathtt{x})$$

The simulation will be based on the double coding (32,33). First, (32) shows that a alone cannot be considered as the only input, since the clock $H(x)$ is not a function of $H(a)$. Hence, to perform the simulation, we consider $H(x)$ as an additional input, this input is subject to the inequality constraint specified in the second relation of (32). Thus $H(x)$ will be the fastest clock of the program, and we may assume $H(x) \equiv Id$, where Id denotes the identity clock $Id_n = n$; then $H(a)$ will be an arbitrary clock extracted from $H(x)$. By doing this, the pair (32,33) satisfies the conditions of theorem 6.1, so that simulation can be performed with local characteristics only. In fact, the simulation scheme follows immediately by *peeling the graph* $\mathcal{CDG}$ given in (33): at the beginning of each instant, the source nodes of this graph are known, and those vertices with the source nodes as only predecessors can be evaluated (this requires either some computation or the simulation of some distribution or conditional distribution), and so on. We show in tables 4 and 5 the

step	clock calculus	graph	action
1	$H(x) = t, H(a) = \perp$	zz $\xrightarrow{t}$ z $\xrightarrow{t}$ x , z ---→ zz	assignment $z := zz$
2	$H(x) = t, H(a) = \perp$	z $\xrightarrow{t}$ x , z ---→ zz	given z, simulate x
3	$H(x) = t, H(a) = \perp$	z ---→ zz	assignment $zz := z$

Table 4: Instant n of the run

example of such a run, where a is supposed absent at instant n and present at $n + 1$. In these tables, branches labelled with "t" are known to be present at the considered instant, and branches that are known to be absent or have been evaluated are removed.

7 Conclusion

We have provided an account of constructive probability theory and its associated SIGN*alea* formalism. There are several new contributions in this paper:

- it has been shown that independent identically distributed (i.i.d.) random sequences are sufficient primitives to construct fairly general random processes, including multiple clocked stochastic Büchi automata and (stochastic timed) Petri nets, provided that proper composition operators be introduced,
- shuffle products, communication of random processes, and conditioning via constraints specification have been defined, that are the above mentioned composition operators; the clock algebra with its operators is a basic tool to express constraints,
- the SIGN*alea* formalism has been proposed which provides adequate syntax for handling the above mentioned primitives and operators,
- constructive probability generalizes Gibbs random fields on graphs to multiple-clocked dynamical systems,
- various services can be provided (simulation,...) by only handling local characteristics of random processes,
- we claim that SIGN*alea* is a convenient tool for various applications such as queuing networks with complex synchronization, or real-time uncertain information processing systems (industrial plant monitoring, real-time decision making,...), since it allows us

step	clock calculus	graph	action
0	$H(x) = t, H(a) = t$	a $\xrightarrow{t}$ z $\xrightarrow{t}$ x , z ---→ zz	simulate a
1	$H(x) = t, H(a) = t$	a $\xrightarrow{t}$ z $\xrightarrow{t}$ x , z ---→ zz	assignment $z := a$
2	$H(x) = t, H(a) = t$	z $\xrightarrow{t}$ x , z ---→ zz	given z, simulate x
3	$H(x) = t, H(a) = t$	z ---→ zz	assignment $zz := z$

Table 5: Instant $n + 1$ of the run

to handle *within a single framework* different features that are usually handled via different models, namely

- real-time programming and verification,
- quantitative aspects of real-time in random systems,
- uncertain real-time information processing systems.

Much has to be done, however, to include other essential services such as Maximum Likelihood or Maximum A Posteriori estimation, inference from data. A very interesting related work in the area of AI as applied to (non real-time) diagnostics is found in [21]. Finally the current SIGNAL compiler has to be extended to handle SIGN*alea*.

Appendices

A More on the mathematical model

The reader will easily recognize that the notions we introduced in subsection 4.1 are just particular instances of the present ones.

Processes

A *process* is a pair $\{\Omega, (\mathcal{F}_n)\}$ where Ω is a set , and $(\mathcal{F}_n)$ is a *filtration,* i.e., an increasing family of σ-algebras. Elements of Ω are denoted by ω and are called *behaviours.* To simplify, the reader who is not familiar with the foundations of probability theory may consider equivalently that a family of equivalence relations written

$$\omega \approx_n \omega'$$

is available on elements of Ω such that

$$\omega \approx_n \omega' \text{ and } m < n \quad \Rightarrow \quad \omega \approx_m \omega'$$

which means that $\approx_n$ distinguishes more behaviours as n increases. Then, similarly as in (15), we define $\mathcal{F}_n$ as the collection of subsets of Ω such that

$$\begin{array}{c} \Omega \supseteq A \in \mathcal{F}_n \\ \Updownarrow \quad \text{by definition} \\ \left.\begin{array}{r} \omega_1 \in A \\ \omega_2 \approx_n \omega_1 \end{array}\right\} \Rightarrow \ \omega_2 \in A \end{array} \tag{34}$$

This definition is indeed mathematically correct if considered measurable spaces are Blackwell spaces [9].

Timers, clocks, and time changes

A *timer* is a map $\mu(\omega, A), A \in \mathbf{N}$ such that

$$\begin{array}{l} A \mapsto \mu(\omega, A) \text{ is an integer valued positive measure, i.e., is additive in } A, \\ \omega \mapsto \mu(\omega, A) \text{ is } \mathcal{F}_n - \text{measurable if } A \subseteq [0, n] \end{array} \tag{35}$$

An interesting particular case arises when $\mu(., \{a\}) \leq 1$ holds for any $a \in \mathbf{N}$. Then the formula

$$H_n^\mu(\omega) = \min \{a \in \mathbf{N} : \mu(\omega, [0, a]) \geq n\} \tag{36}$$

defines a *clock*, i.e. an increasing sequence of $\mathbf{N}$-valued variables satisfying the causality condition

$$\{\omega : H_n^\mu(\omega) \leq n\} \in \mathcal{F}_n \tag{37}$$

Conversely, a timer μ^H can be associated with a clock H via the formula

$$\mu^H(\omega, A) = \#\{n : H_n(\omega) \in A\}$$

and μ^H satisfies (35) provided that H satisfies (37). Then, given a process $\{\Omega, (\mathcal{F}_n)\}$ and a clock H on it, we may consider the new process $\{\Omega, (\mathcal{F}_{H_n})\}$ defined as follows: the relation

$$\omega \approx_{H_n} \omega' \Leftrightarrow m = H_n(\omega) \text{ and } \omega \approx_m \omega'$$

is an equivalence relation, and, using (34) with $\approx_{H_n}$ instead of $\approx_n$, we define a σ-algebra we denote by $\mathcal{F}_{H_n}$. Clocks can be built using the following *filtering* operator [4]: let $B = (B_n)$ is a sequence of boolean variables such that B_n is $\mathcal{F}_{H_n}$-measurable, the filtering of a clock H by B, written $H \downarrow B$, is defined via formula (19). The $\wedge$ and $\vee$ operators are also defined as in (20). The notion of clock can be generalized to allow us to handle *dense* time index sets [16, 12], i.e., time index sets where additional instants can be inserted whenever needed, this is presented in the next subsection.

Generalizing clocks

Here we summarize some of the results of [4, 7] concerning the possibility to define clocks that correspond to upsampling of the *Id* basic clock. This relies on the technique of stacks due to Ornstein [17] for isomorphism theorems in ergodic theory. $\mathcal{T}$ will denote any denumerable, totally ordered set containing $\mathbf{N}$ such that the natural injection from $\mathbf{N}$ into $\mathcal{T}$ be order preserving. We assume that the minimal (resp. maximal) element of $\mathcal{T}$ is 0 (resp. ∞). Elements of $\mathcal{T}$ will be denoted generically by s, t, u, v while integers will be denoted by $m.n, p$. We will use the embedding of $\mathbf{N}$ into $\mathcal{T}$ and write expressions such as $s \leq n$.

A *timer* is a map $\mu(\omega, A), A \in \mathcal{T}$ such that

$$A \mapsto \mu(\omega, A) \text{ is an integer valued positive measure} \tag{38}$$

$$\omega \mapsto \mu(\omega, A) \text{ is } \mathcal{F}_n - \text{measurable if } A \subseteq [0, n+1) \tag{39}$$

An interesting particular case arises when $\mu(., \{a\}) \leq 1$ holds for any $a \in \mathcal{T}$. Then the formula

$$H^\mu_n(\omega) = \min\{a \in \mathcal{T} : \mu(\omega, [0, a]) \geq n\}$$

defines a *clock*, i.e. an increasing sequence of $\mathcal{T}$-valued stopping times. We will also write μ^H to refer to the timer associated with a clock H. Then, given a process $\{\Omega, (\mathcal{F}_n)\}$ and a clock H on it, we may consider the new process $\{\Omega, (\mathcal{F}_{H_n})\}$ defined as follows: the relation

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

is an equivalence relation, and thus defines a σ-algebra we denote by $\mathcal{F}_{H_n}$. It is shown in [4] that clocks can be built using the following two operators:

1. The *filtering* written $H \downarrow B$, where $B = (B_n)$ is a sequence of boolean variables such that B_n is $\mathcal{F}_{H_n}$-measurable: $H \downarrow B$ is the clock associated with the timer

   $$\mu^{H \downarrow B}(\omega, A) = \mu^H(\omega,\ A\ 1_{\{B.(\omega)=true\}})$$

   This formula reduces to (19) when $\mathcal{T} = \mathbf{N}$.

2. The *multiplexing* written $H \uparrow C$, where $C = (C_n)$ is a sequence of nonnegative integer variables such that C_n is $\mathcal{F}_{H_n}$-measurable: $H \uparrow C$ is $\mathcal{T}^H$-valued where $\mathcal{T}^H = \mathcal{T} \times \mathbf{N}$ endowed with the lexicographic order[12], and is the clock associated with the timer defined as follows: for a rectangle $A = B \times \mathbf{N}$

$$\mu^{H\uparrow C}(\omega, A) = \int_B (1 + C_n(\omega))\, d\mu^H(\omega, \{H_n(\omega)\})$$

then, if $\max B \triangleq b - 1$ is finite and $H_m(\omega) = b$

$$\mu^{H\uparrow C}(\omega, A \cup (\{b\} \times [0, n])) = \mu^{H\uparrow C}(\omega, A) + \min\{C_m(\omega), n\}$$

This suffices to define the timer $\mu^{H\uparrow C}$ on segments of $\mathcal{T}^H$, which is enough. See [4] for a picture of this.

Together with the $\wedge$ and $\vee$ of clocks[13], the $\downarrow$ and $\uparrow$ operators provide the set of clocks defined on a given filtered space with a nice structure. It is shown in [4] that SIGNAL allows to "simulate" in some sense all the above introduced operators.

Construction of shuffle products in the general case

Here we generalize the notion of shuffle product to arbitrary processes. For general processes $\{\Omega, (\mathcal{F}_n)\}$ and $\{\Omega', (\mathcal{F}'_n)\}$, their shuffle product is defined as the process $\{\mathcal{W}, (\mathcal{G}_n)\}$ built as follows:

1. Consider a sequence of integers $\gamma \in \Gamma = \{\mathbf{N} \mapsto \mathbf{N} \cup \{0\}\}$ and set $\Omega^\Gamma = \Omega \times \Gamma$; $\gamma(n)$ is to be interpreted as the number of silent instants inserted between n and $n+1$. A filtration $(\mathcal{F}^\Gamma_n)$ on Ω^Γ is defined as follows:

$$(\omega_1, \gamma_1) \approx_n (\omega_2, \gamma_2) \text{ holds iff, for } k : \sum_{i=0}^{k}(1 + \gamma_1(i)) \le n - 1 < \sum_{i=0}^{k+1}(1 + \gamma_1(i))$$

it holds that

$$\begin{array}{ccccc} \omega_1 & \approx_k & \omega_2 & & \\ \gamma_1(i) & = & \gamma_2(i) & \text{for } i \le k & \\ \sum_{i=0}^{k}(1 + \gamma_2(i)) & \le & n - 1 & < & \sum_{i=0}^{k+1}(1 + \gamma_2(i)) \end{array} \tag{40}$$

2. Then take

$$\overline{\mathcal{W}} = \Omega^\Gamma \times \Omega'^\Gamma \ , \ \overline{\mathcal{G}}_n = \mathcal{F}^\Gamma_n \times \mathcal{F}'^\Gamma_n$$

Elements of $\overline{\mathcal{W}}$ are obtained by shuffling two arbitrary trajectories ω and ω' of Ω and Ω' respectively. It is not useful to keep those trajectories $\overline{w}$ for which it occurs that

[12] $(t, n) \le (t', n')$ if $\{t \le t'\}$ or $\{t = t' \text{ and } n \le n'\}$

[13] be careful that the $\vee$ of clocks is not always defined, see [4]

both ω and ω' be absent simultaneously at least once: we remove them. Such events where nothing happens are called *silent* events. This is done formally as follows: for $w = (\omega, \gamma; \omega', \gamma')$, define

$$\mu^{\Omega}(w, [0, n]) = \ k \text{ such that } \sum_{i=0}^{k}(1 + \gamma_1(i)) \leq n - 1 < \sum_{i=0}^{k+1}(1 + \gamma_1(i)) \qquad (41)$$

It is easily verified that this defines a timer on $\{\overline{\mathcal{W}}, (\overline{\mathcal{G}}_n)\}$ which counts the occurrence of the first component ω in the considered shuffle. The timer $\mu^{\Omega'}$ is defined similarly. Finally we set

$$\begin{aligned} \mathcal{W} &= \left\{ w \in \overline{\mathcal{W}} \ : \ \mu^{\Omega}(w, \{n\}) + \mu^{\Omega'}(w, \{n\}) > 0 \ \ \forall n \right\} \\ \mathcal{G}_n &= \overline{\mathcal{G}}_{n|\mathcal{W}} \end{aligned}$$

where $*_{|A}$ denotes the restriction of $*$ to A.

This defines the shuffle product $\{\Omega, (\mathcal{F}_n)\} \sqcup\!\sqcup \{\Omega', (\mathcal{F}'_n)\}$. It is left to the reader to verify that this abstract definition reduces to that of the section 4.1 for processes of the particular form (14).

Liftings

Consider a σ-algebra $\mathcal{H} \subset \mathcal{F}_\infty$, our aim is to lift in a natural way events of $\mathcal{H}$ into the previously constructed shuffle product. The resulting σ-algebra will be still denoted by $\mathcal{H}$ when no confusion is likely to occur. Lifting $\mathcal{H}$ is performed by defining some equivalence relation on $\mathcal{W}$ as follows:

$$(\omega_1, \gamma_1; \omega'_1, \gamma'_1) \approx_{\mathcal{H}} (\omega_2, \gamma_2; \omega'_2, \gamma'_2) \quad \Leftrightarrow \quad \omega_1 \approx_{\mathcal{H}} \omega_2 \qquad (42)$$

This formula allows in particular to lift the original filtration $(\mathcal{F}_n)$ into $\mathcal{W}$, we denote it again by $(\mathcal{F}_n)$.

Lifting clocks and timers is performed as follows. Using formula (21), we can write every clock H of the space $\{\Omega, (\mathcal{F}_n)\}$ in the form $H = Id \downarrow B$ for some boolean signal B. Denote by H^{Ω} the clock associated with the timer introduced in (41): this clock selects the events at which the component ω is present. Then lifting H to the shuffle $\{\Omega, (\mathcal{F}_n)\} \sqcup\!\sqcup \{\Omega', (\mathcal{F}'_n)\}$ yields by definition the clock

$$H = H^{\Omega} \downarrow B \qquad (43)$$

(we denote it again by H) where B is lifted in an obvious way by taking

$$B_n(\omega, \gamma; \omega', \gamma') \triangleq B_n(\omega) \qquad (44)$$

This in turn allows us to lift the filtration $(\mathcal{F}_{H_n})$ to the shuffle, using (42,43,44). From this follows that variables that are $(\mathcal{F}_{H_n})$-measurable on $\{\Omega, (\mathcal{F}_n)\}$ can be lifted to the shuffle as well, this is the way signals can be lifted to shuffle products.

Probabilities on shuffle products and partially randomized spaces

The reader is referred to the abstract construction of the shuffle we have given before. The key step is the insertion of "silent" events using the Γ space. There are two possible ways to perform this: either we consider that we insert a *random* amount of silent events, or we consider that the amount of inserted silent events is unknown but nonrandom. This yields two different approaches.

Randomizing inserted events. Here we want to equip $\{\Omega^\Gamma, (\mathcal{F}_n^\Gamma)\}$ with some probability $\mathbf{P}^\Gamma$ which is "naturally" associated with $\mathbf{P}$. For this we equip Γ with a "universal" law Λ for which the $\gamma(n)$ variables are i.i.d with discrete exponential distribution. Equivalently, considering that $\{\Gamma, \Lambda\}$ defines a point process, we choose Λ for its probability of presence to be $1/2$ (so that we get a Bernoulli shift). Then we equip $\{\Omega^\Gamma, \mathcal{F}_n^\Gamma\}$ with the product

$$\mathbf{P}^\Gamma = \mathbf{P} \otimes \Lambda$$

and we consider the direct product of probability spaces

$$\{\overline{\mathcal{W}}, (\overline{\mathcal{G}}_n), \overline{\mathbf{P}}\} = \{\Omega^\Gamma, \mathcal{F}_n^\Gamma, \mathbf{P}^\Gamma\} \otimes \{\Omega'^\Gamma, \mathcal{F}_n'^\Gamma, \mathbf{P}'^\Gamma\}$$

Then we would have to consider the restriction of $\overline{\mathbf{P}}$ to $\mathcal{W}$. Unfortunately $\overline{\mathbf{P}}(\mathcal{W}) = 0$ so that we should perform this with some care. But recall that $\mathcal{W}$ is defined in terms of the product space $\Lambda^{\otimes 2}$ of the two copies of the Bernoulli shift Λ, and that $\overline{\mathbf{P}} = (\mathbf{P} \otimes \Lambda) \otimes (\mathbf{P}' \otimes \Lambda)$. Hence what we have to define is the "conditional law of (γ, γ) given $\mathcal{W}$": this involves only $\Lambda^{\otimes 2}$. The desired law is then immediate: it is equal to

$$\beta^{\otimes \mathbf{N}} \ , \text{ where } \beta(0,1) = \beta(1,0) = \beta(1,1) = \frac{1}{3}$$

Finally, we equip $\{\mathcal{W}, (\mathcal{G}_n)\}$ with the law

$$\mathbf{Q} = \mathbf{P} \otimes \mathbf{P}' \otimes \beta^{\otimes \mathbf{N}}$$

The relative presence/absence of one component versus the other in the shuffle is random. It is an immediate property of the Bernoulli shifts that, for an observer observing the system only when the first component is present, the presence of the second component is still a Bernoulli process with parameter $1/2$. It should be clear that the particular choice for β we have done is quite arbitrary, thus the alternative approach, although nonstandard, will be preferred.

Not randomizing inserted events. Here it is not possible any more to have a probability on the space $\{\Omega^\Gamma, \mathcal{F}_n^\Gamma\}$ since the component γ is not random any more (but just a variable). But we still can do something. We can lift the filtration $(\mathcal{F}_n)$ into Ω^Γ and call it again $(\mathcal{F}_n)$[14]. Hence we can just consider the probability space

$$\{\Omega^\Gamma, (\mathcal{F}_n), \mathbf{P}\}$$

[14] it is in fact the filtration $(\mathcal{F}^\Gamma_{H^\Omega_n})$ where (H^Ω) is the clock of the presence of the component ω in the shuffle; note that (H^Ω) is not a clock w.r.t. the filtration $(\mathcal{F}_n)$ of the information available to an observer of this firts component only, it is needed to observe jointly both components to have access to it.

i.e. on the same filtration as before, only the underlying space of random elements has been enlarged. Finally, we may consider the probability space

$$\{\Omega^\Gamma, \mathcal{F}_\infty, \mathbf{P}\} \otimes \{\Omega'^\Gamma, \mathcal{F}'_\infty, \mathbf{P}'\}$$

Notice that the σ-algebra $\mathcal{F}_\infty \otimes \mathcal{F}'_\infty$ is strictly smaller than $\mathcal{G}_\infty$: the information on the interleaving of the two components ω and ω' is lost. As it stands, it makes little sense to equip this probability space with a filtration, say $(\mathcal{F}_n \otimes \mathcal{F}'_n)$ since the latter one consists in matching the nth instant of ω with the nth instant of ω', which is irrelevant. To conclude, what we end up with is a "partially randomized" process. It turns out that this is the convenient approach to follow, and this is the one we used in the core of the paper. If the two components of the shuffle were indeed *partially* randomized processes, with probabilities defined only on the σ-algebras $\mathcal{F}^p_\infty$ and $\mathcal{F}'^p_\infty$ respectively, just equip $\mathcal{F}^p_\infty \otimes \mathcal{F}'^p_\infty$ with $\mathbf{P} \otimes \mathbf{P}'$.

Defining communications

Process communications are defined by restricting the shuffle product of the two processes to those trajectories for which the clocks and signals with identical names match together. Again we must consider conditional distribution w.r.t. sets of zero probability. We prove in theorem 4.1 that such conditional distributions do exist when mathematical models of SIGN*alea* are considered.

Referring to (22), it is interesting to notice the following. It may occur that, on the set $H_\cdot = H'_\cdot$, $X_\cdot = X'_\cdot$, the two σ-algebras $\mathcal{G}_\infty$ and $\mathcal{F}_\infty \otimes \mathcal{F}'_\infty$ coincide: this is typically the case when successive communications have completely removed uncertainty in the synchronization of the two processes Π and Π'.[15] Then the result of process communication does not depend on the way we decided to consider silent events (random, or unknown but non-random). An example of such a situation was encountered while discussing the "stochastic automaton".

B Proofs of the fundamental results

The communication operator is a combination of the following two ingredients, namely shuffling, and then restricting the behaviours via specified constraints. In our theoretical analysis, we shall consider both aspects successively.

Proof of proposition 4.1

As indicated while stating proposition 4.1, it is enough to prove point 2. For this we use the notations of paragraphs 4.1.2 and 4.1.3. In particular we denote by $X_1(w), X_2(w), \cdots$ the sequence of the *present* occurrences of the component x in the shuffle, and similarly for the second component y. The σ-algebra $\mathcal{F}^x_\infty \otimes \mathcal{F}^y_\infty$ is spaned by the variables of the product form $f(X_1(w), X_2(w), \cdots, X_m(w))g(Y_1(w), Y_2(w), \cdots, Y_n(w))$ where m, n range over the integers and f, g range over positive real valued functions with proper domains. Hence to prove point

[15] we may say that the communication is *deterministic* in this case

2. of the proposition it is enough to prove that the following formula holds true, where $\tilde{\mathbf{E}}$ and $\mathbf{E}^x$ denotes expectation with respect to the probabilities $\tilde{\mathbf{Q}}$ and $\mathbf{P}^x$ respectively:

$$\begin{aligned} & \tilde{\mathbf{E}}\left[f(X_1, X_2, \cdots, X_m) g(Y_1, Y_2, \cdots, Y_n)\right] \\ = \; & \mathbf{E}^x\left[f(X_1, X_2, \cdots, X_m)\right] \; \mathbf{E}^y\left[g(Y_1, Y_2, \cdots, Y_n)\right] \end{aligned} \tag{45}$$

Denote by H^x and H^y the clocks of the present occurrences of the components x and y respectively. We have, by the monotone convergence theorem,

$$\begin{aligned} & \tilde{\mathbf{E}}\left[f(X_1, X_2, \cdots, X_m) g(Y_1, Y_2, \cdots, Y_n)\right] \\ = \; & \lim_{k \nearrow \infty} \tilde{\mathbf{E}}\left[f(X_1, X_2, \cdots, X_m) g(Y_1, Y_2, \cdots, Y_n) \; \mathbf{1}_{\{H^x_m \le k, H^y_n \le k\}}\right] \\ = \; & \lim_{k \nearrow \infty} \frac{1}{Z_k} \sum_{(w(1), \cdots, w(k))} \Big[f(X_1, X_2, \cdots, X_m) g(Y_1, Y_2, \cdots, Y_n) \; \mathbf{1}_{\{H^x_m \le k, H^y_n \le k\}} \\ & \qquad e^{-\sum_{i=1}^{k} \mu^x(\{i\}) U(x_i) + \mu^y(\{i\}) V(y_i)}\Big] \\ = \; & \left[\frac{1}{Z^x_m} \sum_{(x_1, \cdots, x_m)} f(x_1, x_2, \cdots, x_m) \; e^{-\sum_{j=1}^{m} U(x_j)}\right] \\ & \left[\frac{1}{Z^y_n} \sum_{(y_1, \cdots, y_n)} f(y_1, y_2, \cdots, y_n) \; e^{-\sum_{j=1}^{n} V(y_j)}\right] \\ = \; & \mathbf{E}^x\left[f(X_1, X_2, \cdots, X_m)\right] \; \mathbf{E}^y\left[g(Y_1, Y_2, \cdots, Y_n)\right] \end{aligned}$$

where Z_k, Z^x_m, Z^y_n denote the various normalizing constant factors. This proves (45) and the proposition.

On the other hand, the proof of proposition 4.2 is immediate.

Proof of theorem 4.1

Consider the program $\mathtt{P}\{\mathtt{x}_1, ..., \mathtt{x}_p; \mathtt{y}_1, ..., \mathtt{y}_q\}$. Extract from it the subset of instructions of the form "`potential ...`" or "`given ... potential ...`", such instruction can involve only inputs. This yields another program we denote by $\mathtt{Q}\{\mathtt{x}_1, ..., \mathtt{x}_p\}$. By proposition 4.1, its semantics is

$$\mathtt{Q}\{\mathtt{x}_1, ..., \mathtt{x}_p\} :: \left[\begin{array}{c} \Pi\{\mathtt{x}_1, ..., \mathtt{x}_p\} \\ \mu^{x_1} U(\mathtt{x}_1) + \cdots + \mu^{x_p} U(\mathtt{x}_p) \\ \text{none} \end{array}\right] \tag{46}$$

Next partition the signals $\{\mathtt{x}_1, ..., \mathtt{x}_p\}$ according to the equality of clocks as specified by the original program P, and apply proposition 4.2. This allows us to replace the second field of

formula (46) by the potential

$$U(\mathbf{x}_1, ..., \mathbf{x}_p) = \sum_{i=1}^{I} \mu^i U_i(\mathbf{x}_1^i, ..., \mathbf{x}_{p_i}^i) \tag{47}$$

where the sets of ports denoted by $\mathbf{x}_1^i, ..., \mathbf{x}_{p_i}^i$ have common clock and μ^i is the counter associated with the clock of these signals. Now, it remains to take into account the constraints corresponding to the last field of the specification. The constraint

$$C(\mathbf{x}_1, ..., \mathbf{x}_p; \mathbf{y}_1, ..., \mathbf{y}_q)$$

specifies a subset of the trajectories of the filtered space $\Pi\{\mathbf{x}_1, ..., \mathbf{x}_p; \mathbf{y}_1, ..., \mathbf{y}_q\}$, denote it by $\mathcal{W}$. Also denote by $\mathcal{W}_n$ the projection of $\mathcal{W}$ onto the initial segments of length n. And apply formula (4) for conditioning. This yields the potential

$$U_n = \left(\sum_{m=1}^{n} \sum_{i=1}^{I} \mu^i(\{m\}) U_i((\mathbf{x}_1^i)_m, ..., (\mathbf{x}_{p_i}^i)_m) \right) \mathbf{1}_{\mathcal{W}_n} + \infty \, \mathbf{1}_{\mathcal{W}_n^c} \tag{48}$$

Denote by $\mathcal{F}_n$ the σ-algebra generated by the components $\mathbf{x}_l^i$ for (i, l) ranging over the proper domains. This sequence (48) of potentials yields a consistent family of probability spaces $\{\mathcal{W}_n, \mathcal{F}_n, \mathbf{P}_n\}$ where $\mathbf{P}_n(\omega) = 1/Z_n \ \exp(-U_n(\omega))$ from which the existence of a unique probability $\mathbf{P}$ follows on $\{\mathcal{W}, \mathcal{F}_\infty\}$. Again *only the restriction of* $\mathbf{P}$ *to the* σ*-algebra* $\mathcal{G}_\infty$ *is meaningful, where, as before,* $\mathcal{G}_\infty = \otimes_{i=1}^{I} \mathcal{F}_\infty^i$ and $(\mathcal{F}_n^i)$ is the filtration generated via the observations $\mathbf{x}_l^i \ : \ l = 1, ..., p_i$ up to instant n. This proves the theorem.

C Proof that every stochastic timed Petri net can be specified in SIGN*alea*.

Petri Nets

We borrow from [18] our presentation of Petri Nets. Recall that a Petri Net is first specified by a bipartite graph, i.e., a four-tuple $\mathbf{PN} = \{P, T, I, O\}$, where

1. $P = \{p_1, \cdots, p_{n_p}\}$ is a finite set of *places* which represent system conditions;
2. $T = \{t_1, \cdots, t_{n_t}\}$ is a finite set of *transitions* such that $P \cap T = \emptyset$, they represent system actions;
3. $I(.,.) : P \times T \mapsto \mathbf{N}$, where $I(p_i, t_j)$ equals the number of directed paths from place p_i to transition t_j and is called the *input function*;
4. $O(.,.) : T \times P \mapsto \mathbf{N}$, where $O(t_j, p_i)$ equals the number of directed paths from transition t_j to place p_i and is called the *output function*;

A *token* ($\bullet$) residing in a place indicates that the condition, represented by the place, is met. The dynamic behaviour of the system is modeled by the movement of tokens through the

net via the "firing" of transitions. A transition fires if a certain pre-condition is met. The pre-conditions are met if enough tokens reside in each place incident on the transition (input place). Upon firing, an appropriate number of tokens is deleted from the input places and deposited onto each excident place (output place). Mathematically, a marking μ indicates the distribution of tokens throughout the Petri Net. Let $\mu(p_i)$ equal the number of tokens contained in place p_i. Then the marking μ is a vector $\mu^T = [\mu(p_1), \cdots, \mu(p_{n_p})]$, where each $\mu(p_i)$ is nonnegative. With these notations, it is possible to define the notions of enabled transition and firing:

Enabling of a transition: t_i is enabled if

$$\mu(p_i) \geq I(p_i, t_j) \quad \forall p_i \in P \tag{49}$$

Firing of an enabled transition results in a new marking:

$$\mu'(p_i) = \mu(p_i) + O(t_j, p_i) - I(p_i, t_j) \tag{50}$$

It is customary to observe the rule that no two transitions fire simultaneously. The SIGNAL program encoding such a Petri Net is given next. First we encode the life of transition j:

```
TRANSITION
  (integer j,P,T; [P,T] integer I, [T,P] integer O)
  { ? event SELECTED, [P] integer old_mu
    ! logical ENABLED; [P] integer mu } =

(|(| PRE_TOKEN := [ {i to P} : (old_mu[i] >= I[i,j]) ]
   | ENABLED := apply and to PRE_TOKEN
   |)
 |(| mu := ([ {i to P} : old_mu[i] + O[j,i] - I[i,j] ]) when SELECTED
   |)
 |)
where
   [P] integer PRE_TOKEN
```

This program deserves somme comments, since several additional features of the SIGNAL language are used. Firstly,

```
NAME ( parameters ) { ? inputs ! outputs } =
```

describes the interface of this program, the body of which follows the symbol "=". Using an instance of NAME with particular parameters within another program is performed by writing

```
... | NAME ( p,q,... ) | ...
```

Then "[P] integer mu" declares a vector integer signal of dimension P, and "[P,T] integer I" is a matrix whose (i,j) entry is written I[i,j]. The instruction

```
PRE_TOKEN := [ {i to P} : (old_mu[i] >= I[i,j]) ]
```

yields a P-vector whose i-th component is the outcome of the condition $\mu(p_i) \geq I(p_i, t_j)$ (the μ marking is called `old_mu` here, and we use `mu` to refer to the next marking μ'). The expression, "`apply and to B`" where `B` is a logical `N`-vector yields "`B[1] and ··· and B[N]`". Finally, local signals are declared in the "`where`" part. This program consists of two blocks which implement formulae (49,50) respectively. Note that the firing is actually performed if the transition is enabled and if it is selected (event `SELECTED`). To summarize, the `TRANSITION` module performs locally the following:

1. it receives the current status `old_mu` of the marking, a global information,
2. it checks for enabling condition (49), and emit the outcome of this boolean for global use,
3. it receives the information of whether it has been selected for a firing, a global information,
4. it proceeds on firing accordingly.

Then the program specifying the Petri Net is the following:

```
PETRI
   (integer P,T; [P,T] integer I, [T,P] integer O)
   { ! [P] integer mu } =

(| array j to T of
      (|(| TRANSITION (j,P,T,I,O)
                      ? SELECTED: SELECTED[j], old_mu: old_mu[j]
                      ! ENABLED: ENABLED[j], mu: mu[j]
        | S[j] := S[j-1] default (when ENABLED[j])
        | SELECTED[j] ^= (S[j] ^- S[j-1])
        | global_mu[j] := global_mu[j-1] default mu[j]
        |)
   end
 | S[0] ^= global_mu[0] ^= absent
 | mu := global_mu[T]
 | old_mu := mu $1
 |)
where
   % declarations of local signals ...
```

Two new features of SIGNAL have been used there. The

```
   array j to T of
      PROGRAM[j]
   end
```

is equivalent to

```
PROGRAM[1] | ... | PROGRAM[j] | ... | PROGRAM[T]
```

Next, the instruction

```
TRANSITION (j,P,T,I,O)
           ? SELECTED: SELECTED[j], old_mu: old_mu[j]
           ! ENABLED: ENABLED[j], mu: mu[j]
```

instanciates TRANSITION with actual values of the parameters (j,P,T,I,O) and performs renaming of the input signals SELECTED, old_mu by SELECTED[j], old_mu[j] and output signals ENABLED , mu by ENABLED[j], mu[j] respectively. In the instructions

```
| S[j] := S[j-1] default (when ENABLED[j])
| SELECTED[j] ^= (S[j] ^- S[j-1])
```

the event (when ENABLED[j]) is present when transition j is enabled, and is absent otherwise. Since S[0] ^= absent, the always absent event, this instruction yields S[1] = true if the first transition was enabled, and S[1] absent otherwise. More generally, the first instruction yields S[j] absent until the first j=jo such that the corresponding transition is enabled, and then remains present until S[T]. The notations "^+, ^*, ^-" (only the latter one is used here) respectively denote the supremum, infimum, and complement of clocks. Hence the second instruction yields exactly SELECTED[j] present for j=jo and absent otherwise, this information is then broadcast to each transition for local use. Similarly, the instruction

```
global_mu[j] := global_mu[j-1] default (mu[j] when ENABLED[j])
```

yields, as resulting new marking mu, the one proposed by the first enabled transition according to the indexing by j. To summarize, the global program performs

1. receiving from each transition its current status (enabled or disabled),
2. selecting some transition among those which are enabled, broadcasting this information to the transitions,
3. receiving from the selected transition the increment of the marking,
4. updating the marking and broadcasting the updated marking for next firing.

This shows that SIGNAL can be used to describe a Petri Net. It is interesting to note about this program that the interdependencies between locally and globally performed tasks are quite involved, and depends upon the particular instant in consideration: this is why Conditional Dependency Graphs are essential in handling SIGNAL programs.

Timed Petri Nets

Next, we show how to introduce *timing constraints* in Petri nets. First we introduce the following timer with reset:

```
TIMER { ? event TICK, RESET_GO;  integer DELAY  ! logical GO } =
(| N := DELAY default (ZN-1)
 | ZN := N $1 init 0
 | N ^= TICK ^= GO
 | GO := (false when RESET_GO) default
         (when (ZN=1)) default
         (GO $1 init false)
 |)
```

The first two instructions specify a decreasing counter with reset signal DELAY. The third instruction (N ^= TICK) specifies that the clock of this counter is given by some TICK. The logical GO also has the same clock. When RESET_GO is received GO gets **false**, it gets **true** when the elapsed time since the last reset of the counter equals the current value of DELAY; otherwise, GO remains unchanged. Thus when a reset of the counter occurs, the timer waits a DELAY amount of TICK's until GO switches to **true**; then GO has its own RESET_GO event which makes it switching back to **false**. Note that this programs constrains TICK to be its fastest clock.

Next we have to combine this TIMER with the TRANSITION program. However the latter program has to be slightly modified for this combination to be performed, since the former version of TRANSITION does not keep track of how many times the considered transition has been enabled before firing. This modification is performed now and yields the following program we still call TRANSITION:

```
TRANSITION
  (integer j,P,T; [P,T] integer I, [T,P] integer O)
  { ? event SELECTED, [P] integer old_mu
    ! event READY; [P] integer mu } =

(|(| PRE_TOKEN := [ {i to P} : (old_mu[i] >= I[i,j]) ]
   | ENOUGH_TOKEN := apply and to PRE_TOKEN
   | READY := when (ENOUGH_TOKEN and not (ENOUGH_TOKEN $1))
   |)
 |(| mu := ([ {i to P} : old_mu[i] + O[j,i] - I[i,j] ]) when SELECTED
   |)
 |)
where
   [P] integer PRE_TOKEN
```

The instruction computing ENABLED has been replaced by the following two instructions

```
   | ENOUGH_TOKEN := apply and to PRE_TOKEN
   | READY := when (ENOUGH_TOKEN and not (ENOUGH_TOKEN $1))
```

which emit the event READY the first time the enabling condition has been met by the considered transition. The combination can now be performed:

```
TIMED_TRANSITION
  (integer j,P,T; [P,T] integer I, [T,P] integer O)
  { ? event TICK, SELECTED; integer DELAY; [P] integer old_mu
    ! logical ENABLED; [P] integer mu } =

(| TRANSITION (j,P,T,I,O)
 | TIMER ? RESET_GO: SELECTED  ! GO: ENABLED
 | DELAY ^= READY
 |)
```

The expression "(event mu)" denotes the event which has the same clock as mu, i.e., it represents the clock of mu. This program combines TRANSITION and TIMER as follows: the third instruction asserts that evaluating the holding time within the transition starts exactly when READY occurs, i.e., the first time inequality (49) is satisfied. Then GO indicates enabling of the considered transition, while GO is reset to false when this transition is selected. The program for the Timed Petri Net is finally given:

```
TIMED_PETRI
   (integer P,T; [P,T] integer I, [T,P] integer O)
   { ? event TICK; [P] integer DELAY  ! [P] integer mu } =

(| array j to T of
      (|(| TIMED_TRANSITION (j,P,T,I,O)
             ? DELAY: DELAY[j], SELECTED: SELECTED[j], old_mu: old_mu[j]
             ! ENABLED: ENABLED[j], mu: mu[j]
         | S[j] := S[j-1] default (when ENABLED[j])
         | SELECTED[j] ^= (S[j] ^- S[j-1])
         | global_mu[j] := global_mu[j-1] default mu[j]
         |)
   end
 | S[0] ^= absent
 | global_mu[0] ^= absent
 | mu := global_mu[T]
 | old_mu := mu $1
 |)
where
   % declarations of local signals ...
```

Stochastic Timed Petri net

We investigate here the case of a random holding time, the other cases are handled similarly. We first make a transition random:

```
STOCHASTIC_TIMED_TRANSITION
  (integer j,P,T; [P,T] integer I, [T,P] integer O)
  { ? event TICK, SELECTED, [P] integer old_mu
    ! event ENABLED; [P] integer mu } =

(| TIMED_TRANSITION (j,P,T,I,O)
 | potential U(DELAY)
 |)
```

and finally get the stochastic timed Petri net:

```
STOCHASTIC_TIMED_PETRI
   (integer P,T; [P,T] integer I, [T,P] integer O)
   { ? event TICK  ! [P] integer mu } =

(| array j to T of
      (|(| STOCHASTIC_TIMED_TRANSITION (j,P,T,I,O)
             ? SELECTED: SELECTED[j], old_mu: old_mu[j]
             ! ENABLED: ENABLED[j], mu: mu[j]
         | S[j] := S[j-1] default (when ENABLED[j])
         | SELECTED[j] ^= (S[j] ^- S[j-1])
         | global_mu[j] := global_mu[j-1] default mu[j]
         |)
   end
 | S[0] ^= absent
 | global_mu[0] ^= absent
 | mu := global_mu[T]
 | old_mu := mu $1
 |)
where
   % declarations of local signals ...
```

This finishes the proof of the theorem. The reader should not deduce from this theorem that we do recommand to translate Petri Net specifications into SIGNAL or SIGN*alea*! But we claim that applications that are usually handled via a Petri Net approach can be *directly* modelled with SIGNAL or SIGN*alea*, using an appropriate programming style: such a claim is indeed allowed by the above theorem.

References

[1] K.J. ASTRÖM, A. BENVENISTE (CHAIRMAN), P.E. CAINES, G. COHEN, L. LJUNG, P. VARAIYA, "Facing the challenge of computer science in the industrial applications of control, a joint IEEE-IFAC project, advance report", IRISA, 1991, see also the two reports by A. B. and G. C. in *IEEE Control Systems,* vol 11, No 4, 87-94, June 1991.

[2] A. BENVENISTE, M. MÉTIVER, P. PRIOURET, *Adaptive algorithms and stochastic approximations,* Applications of mathematics, vol 22, Springer Verlag, 1990.

[3] A. BENVENISTE, G. BERRY, "The Synchronous Approach to Reactive and Real-Time Systems", to appear in the special section of the *Proceedings of the IEEE* entitled *Another look at Real-Time Programming,* A. Benveniste and G. Berry Eds., September 1991.

[4] A. BENVENISTE, P. LE GUERNIC, Y. SOREL, M. SORINE "A denotational theory of synchronous reactive systems", to appear in *Information and Computation,* 1991.

[5] A. BENVENISTE, P. LE GUERNIC, "Hybrid Dynamical Systems Theory and the SIGNAL Language", *IEEE transactions on Automatic Control,* 35(5), May 1990, pp. 535–546.

[6] A. BENVENISTE, P. LE GUERNIC, C. JACQUEMOT, "Synchronous programming with events and relations: the SIGNAL language and its semantics", IRISA Research Report 459, Rennes, France, 1989, to appear in *Science of Computer Programming.*

[7] A. BENVENISTE, P. LE GUERNIC, "A denotational theory of synchronous communicating systems", INRIA research report No 685.

[8] F. BOUSSINOT, R. DE SIMONE, "The ESTEREL language", to appear in the special section of the *Proceedings of the IEEE* entitled *Another look at Real-Time Programming,* A. Benveniste and G. Berry Eds., September 1991.

[9] C. DELLACHERIE, P.A. MEYER, *Probabilités et Potentiels,* 2nd edition, Hermann, Paris, 1976.

[10] N. HALBWACHS, P. CASPI, P. RAYMOND, D. PILAUD, "The synchronous data-flow programming language LUSTRE", to appear in the special section of the *Proceedings of the IEEE* entitled *Another look at Real-Time Programming,* A. Benveniste and G. Berry Eds., September 1991.

[11] S. HART, M. SHARIR, "Probabilistic Propositional Temporal Logics", *Information and Control,* 70, 97-155, 1986.

[12] T.A. HENZINGER, Z. MANNA, A. PNUELI, "What good are digital clocks?", Dept. of Computer Science, Stanford University, submitted to *Workshop on Real-time,* Neijmigen, 1992.

[13] R. KINDERMANN, J.L. SNELL *Markov Random Fields and their Applications,* Contemporary Mathematics, vol. 1, American Mathematical Society, Providence, RI, 1980.

[14] M. LE BORGNE, A. BENVENISTE, P. LE GUERNIC, "Dynamical systems over Galois fields and DEDS control problems", *Proc. of the 30th IEEE-CDC conference,* Brighton, UK, December 1991.

[15] P. Le Guernic, T. Gautier, M. Le Borgne, C. Le Maire, "Programming Real-Time Applications with SIGNAL", to appear in the special section of the *Proceedings of the IEEE* entitled *Another look at Real-Time Programming,* A. Benveniste and G. Berry Eds., September 1991.

[16] X. Nicollin, J. Sifakis, S. Yovine, "From ATP to Timed Graphs and Hybrid Systems", REX workshop "Real-Time, theory in practice", Mook, The Netherlands, June 3-7, 1991.

[17] D.S. Ornstein, *Ergodic Theory, Randomness, and Dynamical Systems,* Yale Univ. Press, 1974.

[18] P. Peleties, R. Decarlo, "A Modeling Strategy for Hybrid Systems Based on Event Structures", to appear in *Int. J. of Discrete Event Dynamical Systems,* 1992.

[19] J. Picone, "Continuous Speech Recognition Using Hidden markov Models", *IEEE-ASSP Magazine,* vol 7 Nr 3, 26-41, 1990.

[20] W. Reisig, *Petri Nets,* Springer, N-Y, 1985.

[21] C. Robert, *Modèles Statistiques pour l'Intelligence Artificielle,* Techniques Stochastiques, Masson, Paris, 1991.

LISTE DES DERNIERES PUBLICATIONS INTERNES IRISA

PI 598 KOAN : A VERSATILE TOOL FOR PARALLELIZING REALISTIC RENDERING ALGORITHMS
Didier BADOUEL, Kadi BOUATOUCH, Zakaria LAHJOMRI, Thierry PRIOL
Juillet 1991, 28 pages.

PI 599 STATISTICAL ESTIMATION OF ROUNDOFF ERRORS AND CONDITION NUMBERS
Jocelyne ERHEL
Septembre 1991.

PI 600 NOMBRE DE SOLUTIONS ET SATISFIABILITE D'UN PROBLEME SAT ; UNE APPROCHE ENSEMBLISTE, COMBINATOIRE ET STATISTIQUE
Israël-César LERMAN
Septembre 1991.

PI 601 ACCELERATED STOCHASTIC APPROXIMATION
Bernard DELYON, Anatoli JUDITSKY
Septembre 1991, 22 pages.

PI 602 MINIMUM VARIANCE CONTROL : RANDOM AUTOREGRESSIVE PARAMETERS
Anatoli JUDITSKY
Septembre 1991, 26 pages.

PI 603 L'EXPERIMENTATION D'ALGORITHMES DISTRIBUES SUR MACHINES PARALLELES AVEC ECHIDNA
Jean-Marc JEZEQUEL
Claude JARD
Septembre 1991, 64 pages.

PI 604 A GENERAL METHOD TO DEFINE QUORUMS
Mitchell L. NEILSEN, Masaaki MIZUNO, Michel RAYNAL
Septembre 1991, 20 pages.

PI 605 OBTENTION DES EQUATIONS DYNAMIQUES D'UN SYSTEME PHYSIQUE A PARTIR DE SON MODELE BOND GRAPH
Bénédicte EDIBE
Septembre 1991, 26 pages.

PI 606 CONSTRUCTIVE PROBABILITY AND THE SIGNalea LANGUAGE : BUILDING AND HANDLING RANDOM PROCESSES VIA PROGRAMMING
Albert BENVENISTE
Septembre 1991, 60 pages.

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique.

ISSN 0249-6399