STOPA: A STOchastic Process Algebra for the formal representation of cognitive systems*

Natalia López, Manuel Núñez
Dept. Sistemas Informáticos y Programación
Facultad Informática.
Universidad Complutense de Madrid
C/. Juan del Rosal, 8. E-28040 Madrid. Spain
{natalia,mn}@sip.ucm.es

Fernando L. Pelayo
Dept. Informática. Sección de Albacete.
E. Politécnica Superior.
Universidad de Castilla-La Mancha
Campus Universitario. E-02071 Albacete. Spain
fpelayo@info-ab.uclm.es

Abstract

In this paper we present a formal language to specify cognitive systems. In addition to the usual characteristics of these formalisms, our language features the possibility of including stochastic time. This kind of time is useful to represent systems where the delays are not controlled by fix amounts of time, but they are given by a probability distribution function.

1. Introduction

Cognitive informatics [27] is a relatively new field of knowledge. In fact, we are still in a period where new mechanisms are being introduced in the field. These include implementation and representation languages, theoretical models, algorithms, etc. However, not all the advances in cognitive informatics are due to completely *novel* techniques. In fact, we may (and should!) take advantage of other more mature fields. For example, this is the case of the development of frameworks for the formal specification of cognitive systems. Even though these systems have some inherent characteristics, we may use several decades of study on formal methods to improve and expand the field. The introduction of RTPA [28, 29] represents a very adequate step in this direction. By conveniently putting together the ideas underlying the definition of classical process algebras, RTPA is a new mathematical framework to represent cognitive processes and systems.

In order to understand why we consider stochastic time, it is worth to briefly review the main milestones in the development of process algebras (see [3] for a good overview of the current research topics in the field and [15, 18, 2] for the presentation of the classical formalisms). Process algebras are very suitable to formally specify systems where concurrency is an essential key. This is so because they allow to model these systems in a compositional manner. The first work was very significant, mainly to shed light on concepts and to open research methodologies. However, due to the abstraction of the complicated features, models were still far from real systems. Therefore, some of the solutions were not specific enough, for instance, those related to real time systems. Thus, researchers in process algebras have tried to bridge the gap between formal models described by process algebras and real systems. In particular, features which were abstracted before have been introduced in the models. Thus, they allow the design of systems where not only functional requirements but also performance ones are included. The most significant of these additions are related to notions such as time (e.g. [26, 19, 30, 10, 1]) and probabilities (e.g.

* Research partially supported by the Spanish MCYT project TIC2003-07848-C02, the Junta de Castilla-La Mancha project PAC-03-001, and the Marie Curie project MRTN-CT-2003-505121/TAROT.

[11, 22, 21, 8, 7, 20]). An attempt to integrate time and probabilistic information has been given by introducing *stochastic process algebras* (e.g. [12, 14, 4, 24, 16]). The idea underlying the definition of stochastic time is that the actual amount of time is given by taking into account different probabilities. Thus, we will be able to specify not only *fix* delays (e.g. the process will last 2 milliseconds) but delays that can vary according to a probability distribution function. For example, we may specify a process finishing before 1 millisecond with probability $\frac{1}{3}$, finishing before 2 milliseconds with probability $\frac{1}{2}$, finishing before 4 milliseconds with probability $\frac{4}{5}$, and so on.

Most stochastic process algebras work exclusively with exponential distributions (some exceptions are [5, 9, 13, 6, 17]). The problem is that the combination of parallel/concurrency composition operators and general distributions strongly complicates the definition of semantic models. That is why stochastic process algebras are usually based on (semi)-Markov chains. However, this assumption decreases the expressiveness of the language, because it does not allow to properly express some behaviors where timed distributions are not exponential.

In this paper we build, on the one hand, on our work on stochastic and probabilistic process algebras (see for example [22, 7, 20, 17]) and, on the other hand, on the process algebra RTPA [28, 29] to introduce a new stochastic process algebra to formally represent cognitive processes containing stochastic information. We call this language `STOPA`. The main improvement of our framework with respect to RTPA is that we may specify timed information given by stochastic delays. Nevertheless, let us remark that deterministic delays (as presented in timed process algebras) can be specified by using Dirac distributions. As we will see in the following sections, the inclusion of stochastic time introduces some additional complexity in the definition of the operational semantics of the language.

2. The language `STOPA`

In this section we present our language and its operational semantics. The semantic model is strongly based on [28]. Our modifications are given mainly in the presentation of the operational semantics for process relations and the introduction of the stochastic time.

A process can be defined as a single *meta-process* or as a complex process based on meta-processes using *process relations*. Thus, `STOPA` is described by using the following structure:

`STOPA` ::=	Meta-processes
	| Primary types
	| Abstract data types
	| Process relations
	| System architectures
	| Specification refinement sequences

The syntax and operational semantics of the meta-processes are given in Tables 1 and 2. The definition of the primary data types are the meta-types defined from 2.1 to 2.10 in Table 3. The abstract data types are described in Table 4. The interested reader may find in [28] a detailed explanation of all the definitions appearing in Tables 1-4 as weel as the definitions of *system architectures* and *specification refinement sequences*. In Section 3 we will describe the syntax and operational semantics of the process relations.

As we have already commented in the introduction of this paper, our aim is to add stochastic information in the framework of `RTPA`. For that reason we have included random variables in the syntax of the process relations, that is, a process can be delayed according to a random variable. We will suppose that the sample space (that is, the domain of random variables) is the set of real numbers $\mathbb{R}$ and that random variables take only positive values, that is, given a random variable ξ we have $\mathrm{P}(\xi \leq t) = 0$ for any $t \leq 0$. The reason for this restriction is that random variables are always associated with time distributions.

Definition 1 Let ξ be a random variable. We define its *probability distribution function*, denoted by F_ξ, as the function $F_\xi : \mathbb{R} \longrightarrow (0, 1]$ such that $F_\xi(x) = \mathrm{P}(\xi \leq x)$, where $\mathrm{P}(\xi \leq x)$ is the probability that ξ assumes values less than or equal to x. □

The following probability distribution functions will be used in the rest of the paper.

Uniform Distributions. Let $a, b \in \mathbb{R}^+$ such that $a < b$. A random variable ξ follows a uniform distribution in the interval $[a, b]$, denoted by $U(a, b)$, if its associated probability distribution function is:

$$F_\xi(x) = \begin{cases} 0 & \text{if } x \leq a \\ \frac{x-a}{b-a} & \text{if } a < x < b \\ 1 & \text{if } x \geq b \end{cases}$$

These distributions allow us to keep compatibility with time intervals in timed process algebras in the sense that the same *weight* is assigned to all the times in the interval.

Discrete Distributions. Let $P_I = \{(t_i, p_i)\}_{i \in I}$ be a set of pairs such that for any $i \in I$ we have that $t_i \geq 0$, $p_i > 0$, for any $i, j \in I$, if $i \neq j$ then $t_i \neq t_j$,

No.	Meta-process	Syntax	Operational Semantics
1.1	System	$\S(SysID_S)$	Represents a system, $SysID$, identified by a string, S

1.2	Assignment	$y_{Type} := x_{Type}$	if $x.type = y.type$
			then $x.value \Rightarrow y.value$
			else $!(\text{@AssigmentTypeError}_S)$
			where $Type \in \text{Meta} - \text{Types}$

1.3	Addressing	$ptrP\hat{} := x_{Type}$	if $prt.type = x.type$
			then $x.value \Rightarrow ptr.value$
			else $!(\text{@AssigmentTypeError}_S)$
			where $Type \in \{H, Z, P\hat{}\}$

1.4	Input	$\text{Port}(ptrP\hat{})_{Type}\| > x_{Type}$	if $\text{Port}(ptrP\hat{}).type = x.type$
			then $\text{Port}(ptrP\hat{}).value \Rightarrow x.value$
			else $!(\text{@InputTypeError}_S)$
			where $Type \in \{B, H\}$, $P\hat{} \in \{H, N, Z\}$

1.5	Output	$x_{Type}\| < \text{Port}(ptrP\hat{})_{Type}$	if $\text{Port}(ptrP\hat{}).type = x.type$
			then $x.value \Rightarrow \text{Port}(ptrP\hat{}).value$
			else $!(\text{@OutputTypeError}_S)$
			where $Type \in \{B, H\}$, $P\hat{} \in \{H, N, Z\}$

1.6	Read	$\text{Mem}(ptrP\hat{})_{Type} > x_{Type}$	if $\text{Mem}(ptrP\hat{}).type = x.type$
			then $\text{Mem}(ptrP\hat{}).value \Rightarrow x.value$
			else $!(\text{@ReadTypeError}_S)$
			where $Type \in \{B, H\}$, $P\hat{} \in \{H, N, Z\}$

1.7	Write	$x_{Type} < \text{Mem}(ptrP\hat{})_{Type}$	if $\text{Mem}(ptrP\hat{}).type = x.type$
			then $x.value \Rightarrow \text{Mem}(ptrP\hat{}).value$
			else $!(\text{@WriteTypeError}_S)$
			where $Type \in \{B, H\}$, $P\hat{} \in \{H, N, Z\}$

1.8	Timing	a) $@t_{\texttt{hh:mm:ss:ms}} := \S t_{\texttt{hh:mm:ss:ms}}$
		b) $@t_{\texttt{yy:MM:dd}} := \S t_{\texttt{yy:MM:dd}}$
		c) $@t_{\texttt{yy:MM:dd:hh:mm:ss:ms}} :=$
		$\S t_{\texttt{yy:MM:dd:hh:mm:ss:ms}}$
			if $@t.type = \S t.type$
			then $\S t.value \Rightarrow @t.value$
			else $!(\text{@TimingTypeError}_S)$
			where $\texttt{yy} \in \{0, \ldots, 99\}$, $\texttt{MM} \in \{1, \ldots, 12\}$, $\texttt{dd} \in \{1, \ldots, 31\}$, $\texttt{hh} \in \{0, \ldots, 23\}$, $\texttt{mm}, \texttt{ss} \in \{0, \ldots, 59\}$, $\texttt{ms} \in \{0, \ldots, 999\}$

1.9	Duration	$@tn_Z := \S tn_Z + \Delta n_Z$	if $@tn.type = \Delta n.type = \S tn.type = Z$ then $(\S tn.value + \Delta n.value)$ mod MaxValue $\Rightarrow @tn.value$ where MaxValue is the upper bound of the system relative-clock, and the unit of all values is ms
1.10	Memory allocation	AllocateObject $(\text{ObjectID}_S$, NofElements_N, $\text{ElementType}_{RT})$	$n_N := \text{NofElements};$ $\mathcal{R}_{i=1}^{n}(\text{new ObjectID}(i_N) : \text{ElementType}_{RT});$ Ⓢ $\text{ObjectID.Existed}_{BL} := \texttt{true}$

Table 1: Meta-processes (1/2).

No.	Meta-process	Syntax	Operational Semantics
1.11	Memory release	ReleaseObject($\mathrm{ObjectID}_S$)	delete $\mathrm{ObjectID}_S$// System.Garbage Collection( ) ; $\mathrm{ObjectID}_S$:= null ; ⓢ $\mathrm{ObjectID.Release}_{BL}$:= `true`
1.12	Increase	$\uparrow(n_{Type})$	if $n.value <$ MaxValue then $n.value + 1 \Rightarrow n.value$ else !(@$\mathrm{ValueOutofRange}_S$) where $Type \in \{N, Z, B, H, P\hat{}\}$ MaxValue = min{run-time defined upper bound, nature upper bound of $Type$}

1.13	Decrease	$\downarrow(n_{Type})$	if $n.value > 0$
			then $n.value - 1 \Rightarrow n.value$
			else !(@$\mathrm{ValueOutofRange}_S$)
			where $Type \in \{N, Z, B, H, P\hat{}\}$

1.14	Exception detection	!(@e_S)	$\uparrow$ ($\mathrm{ExceptionLogPtr}_{P\hat{}}$); @$e_S \Rightarrow$ Mem($\mathrm{ExceptionLogPtr}_{P\hat{}})_S$
1.15	Skip	$\emptyset$	Exit a current control structure, such as loop, branch, or switch
1.16	Stop	stop	System stop

Table 2: Meta-processes (2/2).

and $\sum p_i = 1$. A random variable ξ follows a discrete distribution with respect to P_I, denoted by $D(P_I)$, if its associated probability distribution function is:

$$F_\xi(x) = \sum_{i \in I} \{\!|p_i \mid x \geq t_i|\!\}$$

Let us note that $\{\!|$ and $|\!\}$ represent multisets. Discrete distributions are important because they allow us to express *passive* actions, that is, actions that are willing, from a certain point of time, to be executed with probability 1.

Exponential Distributions. Let $0 < \lambda \in \mathbb{R}$. A random variable ξ follows an exponential distribution with parameter λ, denoted by $E(\lambda)$ or simply λ, if its associated probability distribution function is:

$$F_\xi(x) = \begin{cases} 1 - e^{-\lambda x} & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$

Poisson Distributions. Let $0 < \lambda \in \mathbb{R}$. A random variable ξ follows a Poisson distribution with parameter λ, denoted by $P(\lambda)$, if it takes positives values only in $\mathbb{N}$ and its associated probability distribution function is:

$$F_\xi(x) = \begin{cases} \sum_{t \in \mathbb{N}, t \leq x} \frac{\lambda^t}{t!} e^{-\lambda t} & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$

During the rest of the paper, mainly in the examples and when no confusion arises, we will identify random variables with their probability distribution functions.

Example 1 Let us consider the process $U(1, 3)$;P. This process will behave as P in less than 1 milliseconds with probability 0, in a time less than or equal to t milliseconds, with $1 < t < 3$, with probability $\frac{t-1}{2}$, and it will behave as P in less than 3 milliseconds with probability 1. □

Regarding *communication* actions, they can be divided into *input* and *output actions*. Next, we define our alphabet of actions.

Definition 2 We consider a set of *communication* actions Act = `Input` ∪ `Output`, where we assume `Input` ∩ `Output` = $\emptyset$. We suppose that there exists a bijection f : `Input` $\longrightarrow$ `Output`. For any *input* action $a? \in$ `Input`, $f(a?)$ is denoted by the *output* action $a! \in$ `Output`. If there exists a message transmission between $a?$ and $a!$ we say that a is the *channel* of the communication ($a, b, c, \cdots$ to range over Act).

We also consider a special action $\tau \notin Act$ that represents internal behavior of the processes. We denote by Act_τ the set $Act \cup \{\tau\}$ ($\alpha, \beta, \gamma, \cdots$ to range over Act_τ).

Besides, We consider a denumerable set Id of process identifiers. In addition, we denote by $\mathcal{V}$ the set of random variables ($\xi, \xi', \psi, \cdots$ to range over $\mathcal{V}$). $\square$

3. Process Relations

In this section, we define the syntax and operational semantics of our process algebra to describe process relations. Operational semantics is probably the simplest and more intuitive way to give semantics to any process language. In this part of the description of the language, operational behaviors will be defined by means of transitions $P \xrightarrow{\omega} P'$ that each process can execute. These are obtained in a structured way by applying a set of inference rules [25]. The intuitive meaning of a transition as $P \xrightarrow{\omega} P'$ is that the process P may perform the action ω and, after this action is performed, then it behaves as P'. Let us note that labels appearing in these operational transitions have the following types: $a? \in \mathtt{Input}$, $a! \in \mathtt{Output}$, $a \in Act$, $\alpha \in Act \cup \{\tau\}$, $\omega \in Act_\tau \cup \mathcal{V}$, and $\xi \in \mathcal{V}$.

The set of process relations, as well as their operational semantics, is given in Table 5. Next, we intuitively describe each of the operators.

The **external, internal**, and **stochastic** choice process relations are used to describe the choice among different actions. They are respectively denoted by

$$\sum a_i \,;\, P_i \qquad \sum\!\!\!\!\circ\ \tau \,;\, P_i \qquad \sum\!\!\!\!\bullet\ \xi_i \,;\, P_i$$

where $a_i \in Act$ and $\xi_i \in \mathcal{V}$. The inference rules describing the behavior of these process relations are (CHO1), (CHO2), and (CHO3).

The process $\sum a_i \,;\, P_i$ will perform one of the actions a_i and after that it behaves as P_i. The term $\sum\!\!\!\!\circ\ \tau \,;\, P_i$ represents the internal choice among the processes P_i. Once the choice is made, by performing an internal action τ, the process behaves as the chosen process. Finally, the process $\sum\!\!\!\!\bullet\ \xi_i \,;\, P_i$ will be delayed by ξ_i a certain amount of time t with probability $p = \mathrm{P}(\xi_i \leq t)$ and after that it will behave as P_i (see Example 1).

Sequence is a process relation in which two processes are consequently performed. This relation is denoted by:

$$P \,;\, Q$$

The rules (SEQ1), (SEQ2), (SEQ3), and (SEQ4) describe the behavior of these process relations. Intuitively, P is initially performed. Once P finishes, Q starts its performance.

If the process P can perform an action then $P \,;\, Q$ will perform it. If the process P finishes then the process $P \,;\, Q$ will behave as Q. A process will finish its execution if it performs the action $\surd$ (see rule (SEQ4)).

The **branch** and **switch** process relations are denoted by

$$(?\mathrm{expBL} = \mathtt{true}) \,;\, P \mid (?\mathrm{expBL} = \mathtt{false}) \,;\, Q$$

and

$$\mid (?\mathrm{expNUM} = i) \,;\, P_i$$

The behavior of the first operator is described in the rule (BRA). If the boolean expression expBL evaluates to $\mathtt{true}$ then P is performed; otherwise, Q is performed. In the second operator, if the value of the numerical expression expBL is equal to i then the process P_i is performed (see rule (SWI)).

The **FOR-DO, REPEAT**, and **WHILE-DO** process relations are denoted, respectively, by:

$$\mathop{\mathcal{R}}\limits_{i=1}^{\mathtt{n}} P \qquad \mathop{\mathcal{R}}\limits_{\geq 1}^{\mathtt{expBL} \neq \mathtt{true}} P \qquad \mathop{\mathcal{R}}\limits_{\geq 0}^{\mathtt{expBL} \neq \mathtt{true}} P$$

and their operational semantics are given in the rules (FOR), (REP), (WHI1) and (WHI2).

The *FOR-DO* process relation may be used to describe the performance of a certain process a fix amount of times n. Thus, $\mathcal{R}_{i=1}^{n} P$ behaves as P and after that as $\mathcal{R}_{i=1}^{n-1} P$.

The *REPEAT* process relation executes a process P at least once. It continues its execution until a certain expression takes the value $\mathtt{true}$. Thus, the term $\mathcal{R}_{\geq 1}^{\mathrm{expBL} \neq \mathtt{true}} P$ behaves as P, and after that, as a *WHILE-DO* process relation.

The *WHILE-DO* process relation $\mathcal{R}_{\geq 0}^{\mathrm{expBL} \neq \mathtt{true}} P$ behaves as P if the boolean expression is $\mathtt{true}$, and after that behaves again as $\mathcal{R}_{\geq 0}^{\mathrm{expBL} \neq \mathtt{true}} P$. If the boolean expression is $\mathtt{false}$ then the process finishes its performance.

Recursion is a process relation in which the definition of a process may contain a call to itself. We will use a notation slightly different to that in RTPA

$$X := P$$

where $X \in Id$, that is, a process identifier. The rule (REC1) applies to external and internal actions while (REC2) applies to stochastic actions. Let us also remark that $P[X/X := P]$ represents the substitution of all the free occurrences of X in P by $X := P$.

The **parallel, concurrence**, and **interleaving** process relations are denoted, respectively, by:

$$P \parallel_{tc} Q \qquad P \oint Q \qquad P \mid\mid\mid Q$$

Parallel is a process relation in which two processes are executed simultaneously, synchronized by a common system clock. The parallel process relation is designed to model behaviors of a multi-processor single-clock system. The parameter tc indicates the time of

No.	Meta-type	Syntax
2.1	Natural number	N
2.2	Integer	Z
2.3	Real	R
2.4	String	S
2.5	Boolean	$BL = \{\texttt{true}, \texttt{false}\}$
2.6	Byte	B
2.7	Hexadecimal	H
2.8	Pointer	$P\hat{}$
2.9	Time	$\texttt{hh} : \texttt{mm} : \texttt{ss} : \texttt{ms}$ where $\texttt{hh} \in \{0, \ldots, 23\}, \texttt{mm}, \texttt{ss} \in \{0, \ldots, 59\}, \texttt{ms} \in \{0, \ldots, 999\}$
2.10	Date	$\texttt{yy} : \texttt{MM} : \texttt{dd}$ where $\texttt{yy} \in \{0, \ldots, 99\}, \texttt{MM} \in \{1, \ldots, 12\}, \texttt{dd} \in \{1, \ldots, 31\}$
2.11	Date/Time	$\texttt{yyyy} : \texttt{MM} : \texttt{dd} : \texttt{hh} : \texttt{mm} : \texttt{ss} : \texttt{ms}$ where $\texttt{yyyy} \in \{0, \ldots, 9999\}, \texttt{MM} \in \{1, \ldots, 12\}, \texttt{dd} \in \{1, \ldots, 31\}$, $\texttt{hh} \in \{0, \ldots, 23\}, \texttt{mm}, \texttt{ss} \in \{0, \ldots, 59\}, \texttt{ms} \in \{0, \ldots, 999\}$
2.12	Run-time determinable type	RT
2.13	System architectural type	ST
2.14	Event	$@e_S$
2.15	Status	Ⓢ s_{BL}

Table 3: Meta-types.

No.	ADT	Syntax	Designed behaviors
3.1	Stack	`Stack`:ST	Stack. (Create, Push, Pop, Clear, EmptyTest, FullTest, Release)
3.2	Record	`Record`:ST	Record. (Create, fieldUpdate, Update, FieldRetrieve, Retrieve, Release)
3.3	Array	`Array`:ST	Array. (Create, Enqueue, Serve, Clear, EmptyTest, FullTest, Release)
3.4	Queue (FIFO)	`Queue`:ST	Queue.(Create, Enqueue, Serve, Clear, EmptyTest, FullTest, Release)
3.5	Sequence	`Sequence`:ST	Sequence.(Create, Retrieve, Append, Clear, EmptyTest, FullTest, Release)
3.6	List	`List`:ST	List. (Create, FindNext, FindPrior, Findith, FindKey, Retrieve, Update, InsetAfter, InsertBefore, Delete, CurrentPos, FullTest, EmptyTest,SizeTest,Clear,Release)
3.7	Set	`Set`:ST	Set. (Create, Assign, In, Intersection, Union, Difference, Equal, Subset, Release)
3.8	File (Sequential)	`SeqFile`:ST	SeqFile. (Create, Reset, Read, Append, Clear, EndTest, Release)
3.9	File (Random)	`RandFile`:ST	RandFile. (Create, Reset, Read, Write, Clear, EndTest, Release)
3.10	Binary Tree	`BTree`:ST	BTree. (Create, TRaverse, Insert, DeleteSub, Update, Retrieve, Find, Characteristics, EmptyTest, Clear, Release)

Table 4: Abstract data types.

$$(\text{CHO1})\frac{}{\sum a_i;P_i \xrightarrow{a_i} P_i} \qquad (\text{CHO2})\frac{}{\sum \tau;P_i \xrightarrow{\tau} P_i} \qquad (\text{CHO3})\frac{}{\sum \xi_i;P_i \xrightarrow{\xi_i} P_i}$$

$$(\text{SEQ1})\frac{P \xrightarrow{\alpha} P'}{P;Q \xrightarrow{\alpha} P';Q} \qquad (\text{SEQ2})\frac{P \xrightarrow{\surd} P'}{P;Q \xrightarrow{\tau} Q} \qquad (\text{SEQ3})\frac{P \xrightarrow{\xi} P'}{P;Q \xrightarrow{\xi} pP';Q} \qquad (\text{SEQ4})\frac{}{\text{exit} \xrightarrow{\surd} \text{stop}}$$

$$(\text{BRA})\frac{expBL{=}\texttt{true}}{(?expBL{=}\texttt{true});P \mid (?expBL{=}\texttt{false});Q \xrightarrow{\tau} P} \qquad (\text{SWI})\frac{expNUM{=}i}{\mid (?expNUM{=}i);P_i \xrightarrow{\tau} P_i}$$

$$(\text{FOR})\frac{}{\mathcal{R}_{i=1}^{n} P \xrightarrow{\tau} P;\mathcal{R}_{i=1}^{n-1} P} \qquad (\text{REP})\frac{}{\mathcal{R}_{\geq 1}^{expBL\neq\texttt{true}} P \xrightarrow{\tau} P;\mathcal{R}_{\geq 0}^{expBL\neq\texttt{true}} P}$$

$$(\text{WHI1})\frac{expBL{=}\texttt{true}}{\mathcal{R}_{\geq 0}^{expBL\neq\texttt{true}} P \xrightarrow{\tau} P;\mathcal{R}_{\geq 0}^{expBL\neq\texttt{true}} P} \qquad (\text{WHI2})\frac{expBL{=}\texttt{false}}{\mathcal{R}_{\geq 0}^{expBL\neq\texttt{true}} P \xrightarrow{\tau} \text{exit}}$$

$$(\text{REC1})\frac{P[X/X{:=}P] \xrightarrow{\alpha} P'}{X{:=}P \xrightarrow{\alpha} P'} \qquad (\text{REC2})\frac{P[X/X{:=}P] \xrightarrow{\xi} P'}{X{:=}P \xrightarrow{\xi} P'}$$

$$(\text{PAR1})\frac{P \xrightarrow{\alpha} P'}{P\|_{tc}Q \xrightarrow{\alpha} P'\|_{tc}Q} \qquad (\text{PAR2})\frac{Q \xrightarrow{\alpha} Q'}{P\|_{tc}Q \xrightarrow{\alpha} P\|_{tc}Q'} \qquad (\text{PAR3})\frac{P \xrightarrow{a} P',\ Q \xrightarrow{b} Q',\ a{*}b\neq\tau}{P\|_{tc}Q \xrightarrow{a*b} P'\|_{tc}Q'}$$

$$(\text{PAR4})\frac{P \xrightarrow{\xi} P',\ P\|_{tc}Q \not\xrightarrow{\alpha}}{P\|_{tc}Q \xrightarrow{\xi} P'\|_{tc'}\text{cond}(Q,tc'-tc)} \qquad (\text{PAR5})\frac{Q \xrightarrow{\xi} Q',\ P\|_{tc}Q \not\xrightarrow{\alpha}}{P\|_{tc}Q \xrightarrow{\xi} \text{cond}(P,tc'-tc)\|_{tc'}Q'}$$

$$(\text{CON1})\frac{P \xrightarrow{\alpha} P'}{P\oint Q \xrightarrow{\alpha} P'\oint Q} \qquad (\text{CON2})\frac{Q \xrightarrow{\alpha} Q'}{P\oint Q \xrightarrow{\alpha} P\oint Q'} \qquad (\text{CON3})\frac{P \xrightarrow{a?} P',\ Q \xrightarrow{a!} Q'}{P\oint Q \xrightarrow{\tau} P'\oint Q'}$$

$$(\text{CON4})\frac{P \xrightarrow{a!} P',\ Q \xrightarrow{a?} Q'}{P\oint Q \xrightarrow{\tau} P'\oint Q'} \qquad (\text{CON5})\frac{P \xrightarrow{\xi} P'}{P\oint Q \xrightarrow{\xi} P'\oint Q} \qquad (\text{CON6})\frac{Q \xrightarrow{\xi} Q'}{P\oint Q \xrightarrow{\xi} P\oint Q'}$$

$$(\text{INT1})\frac{P \xrightarrow{\alpha} P'}{P|||Q \xrightarrow{\alpha} P'|||Q} \qquad (\text{INT2})\frac{Q \xrightarrow{\alpha} Q'}{P|||Q \xrightarrow{\alpha} P|||Q'} \qquad (\text{INT3})\frac{P \xrightarrow{\xi} P'}{P|||Q \xrightarrow{\xi} P'|||Q} \qquad (\text{INT4})\frac{Q \xrightarrow{\xi} Q'}{P|||Q \xrightarrow{\xi} P|||Q'}$$

$$(\text{PIPE1})\frac{P \xrightarrow{\alpha} P'}{P{>>}Q \xrightarrow{\alpha} P';Q} \qquad (\text{PIPE2})\frac{P \xrightarrow{\surd} P',\ \texttt{Input}(Q){=}\texttt{Output}(P)}{P{>>}Q \xrightarrow{\tau} Q} \qquad (\text{PIPE3})\frac{P \xrightarrow{\xi} P'}{P{>>}Q \xrightarrow{\xi} pP'{>>}Q}$$

$$(\text{TDD})\frac{t\text{system}_{\texttt{hh:mm:ss:ms}}{=}ti_{\texttt{hh:mm:ss:ms}}}{@ti_{\texttt{hh:mm:ss:ms}}\hookrightarrow P_i,\ i\in\{1,\dots,n\} \xrightarrow{\tau} P_i} \qquad (\text{EDD})\frac{e\text{system}{=}ei_S}{@ei_S\hookrightarrow P_i,\ i\in\{1,\dots,n\} \xrightarrow{\tau} P_i}$$

$$(\text{INTER1})\frac{@e_S\text{captured}{=}\texttt{true}}{P\|_{tc}\odot(@e_S\nearrow Q\searrow\odot) \xrightarrow{\tau} Q;P} \qquad (\text{INTER2})\frac{@e_S\text{captured}{=}\texttt{false},\ P \xrightarrow{\alpha} P'}{P\|_{tc}\odot(@e_S\nearrow Q\searrow\odot) \xrightarrow{\alpha} P'\|_{tc}\odot(@e_S\nearrow Q\searrow\odot)}$$

$$(\text{INTER3})\frac{@e_S\text{captured}{=}\texttt{false},\ P \xrightarrow{\xi} P'}{P\|_{tc}\odot(@e_S\nearrow Q\searrow\odot) \xrightarrow{\xi} P'\|_{tc}\odot(@e_S\nearrow Q\searrow\odot)}$$

Table 5: Operational semantics of the process relations.

the system clock. It will vary as time passes. If one of the processes of the parallel composition can perform a non-stochastic action then the composition will perform it (see rules (PAR1) and (PAR2)). We suppose that there is an operation $*$ on the set of actions Act such that $(Act, *)$ is a monoid and τ is its identity element. Thus, by rule (PAR3), if we have the parallel composition of P and Q, P may perform a, Q may perform b, and $a * b \neq \tau$ then they will evolve together. Let us suppose that P can perform a stochastic action, and neither external nor internal actions can be performed by the composition. Then $P \,\|_{tc}\, Q$ will perform the temporal action and Q will evolve into $\mathrm{cond}(A, \Delta t)$, being Δt the actual time consumed by the stochastic action. That is, $\Delta t = tc' - tc$, where tc is the system time in the moment that the performance of ξ started and tc' is the system time after the execution of ξ.

The definition of the function $\mathrm{cond}(P, \Delta t)$ is given in Table 6. In that table we have included, for the sake of completeness, all the cases of the function. However, the most relevant part of this definition is the one concerning choice process relations. In this case,

$$\mathrm{cond}(P, \Delta t) = \begin{cases} \sum a_i \,;\, \mathrm{cond}(P_i, \Delta t) & \text{if } P = \sum a_i \,;\, P_i \\ \sum \tau \,;\, \mathrm{cond}(P_i, \Delta t) & \text{if } P = \sum \tau \,;\, P_i \\ \sum \xi'_i \,;\, P_i & \text{if } P = \sum \xi_i \,;\, P_i \end{cases}$$

where ξ'_i is the random variable whose probability distribution function is given as a *conditional probability*. We have that $\mathrm{P}(\xi'_i \leq t') = \mathrm{P}(\xi_i \leq t'' | \Delta t + t' \leq t'')$, that is, the probability of the random variable ξ'_i to finish before t' units of time have passed is equal to the probability of the original random variable ξ, to be less than or equal to t'', provided that $\Delta t + t' \geq t''$. Let us remark that the modification of the process of the parallel composition that does not perform the stochastic action is needed. This is so because, in this case, we have a multi processor single-clock system, so the time consumed by the stochastic action has to be taken into account in both sides of the parallel composition.

Example 2 Let $P = \xi_1 \,;\, P_1 \,\|_{tc}\, \xi_2 \,;\, P_2$. Let us suppose that we have transitions of the form

$$P \xrightarrow{\xi_1} P_1 \,\|_{tc'}\, \xi_2 \,;\, P_2$$

The passage of time would not be reflected in the right hand side of the parallel composition. Nevertheless, the random variable associated with the action b cannot be ξ_2 because some time has passed. That is why we generate the transition $P \xrightarrow{\xi_1} P_1 \,\|_{tc'}\, \mathrm{cond}(\xi_2 \,;\, P_2, tc' - tc)$, where $\mathrm{cond}(\xi_2 \,;\, P_2, tc' - tc) = \xi'_2 \,;\, P_2$ and ξ'_2 is the random variable with probability distribution function defined as:

$$F_{\xi'_2}(t) = \mathrm{P}(\xi'_2 \leq t) = \mathrm{P}(\xi_2 \leq t' \mid t + (tc' - tc) \leq t')$$

$\square$

Concurrence is a process relation in which two processes are simultaneously and asynchronously executed, according to separate system clocks. This process relation is designed to model behaviors of a multi-processor multi-clock system. The rules (CON1) and (CON2) indicate that if one of the processes of the composition can perform an action then the composition will asynchronously perform it. However, if one of the processes of the composition can perform an input action and the other can perform the corresponding output action then there is a communication and the process relation will perform it (see rules (CON3) and (CON4)). Since we have in this case a multi-clock system, if one of the processes can perform a stochastic action then the composition will perform it without modifying the other part of the composition (see rules (CON5) and (CON6)).

Interleaving is a process relation in which two processes are simultaneously executed while maintaining by a common system clock. The interleaving process relation is designed to model behaviors of a single-processor single-clock system. If one of the processes can perform a non-stochastic action then the composition will perform it (see rules (INT1) and (INT2)). Regarding stochastic actions, see rules (INT3) and (INT4), we consider that a stochastic action , once it has started, cannot be interrupted. Besides, if one of the processes performs a stochastic action then the other component is not modified. In fact, as we suppose a single processor single-clock system, the other side of the composition is not active, so no time has passed for it.

Pipeline is a process relation in which two processes are interconnected to each other. The second process takes the inputs from the outputs of the first one. This process relation is denoted by:

$$P >> Q$$

Thus, if P can perform an action then $P >> Q$ will perform it (see rules (PIPE1) and (PIPE3)). Once the process P is finished, that is, P can perform the action $\surd$, $P >> Q$ behaves as Q (see rule (PIPE2)).

Time-driven dispatch is a process relation in which the i-th process is triggered by a predefined system time $ti_{\mathtt{hh:mm:ss:ms}}$. It is denoted as follows:

$$@ti_{\mathtt{hh:mm:ss:ms}} \hookrightarrow P_i,\ i \in \{1, \ldots, n\}$$

$$\mathrm{cond}\Big(\sum a_i\,;P_i,\Delta t\Big) = \sum a_i\,;\mathrm{cond}\Big(P_i,\Delta t\Big) \qquad \mathrm{cond}\Big(\sum\,\tau\,;P_i,\Delta t\Big) = \sum\tau\,;\mathrm{cond}\Big(P_i,\Delta t\Big)$$

$$\mathrm{cond}\Big(\sum\,\xi_i\,;P_i,\Delta t\Big) = \sum\xi_i'\,;P_i \qquad \mathrm{cond}\Big(P\,;Q,\Delta t\Big) = \mathrm{cond}\Big(P,\Delta t\Big)\,;Q$$

$$\mathrm{cond}\Big(P >> Q,\Delta t\Big) = \mathrm{cond}\Big(P,\Delta t\Big) >> Q \qquad \mathrm{cond}\Big(\,|\,(?\mathrm{expNUM}=i)\,;P_i,\Delta t\Big) = \,|\,(?\mathrm{expNUM}=i)\,;\mathrm{cond}\Big(P_i,\Delta t\Big)$$

$$\mathrm{cond}\Big((?\mathrm{expBL}=\mathtt{true})\,;P\,|\,(?\mathrm{expBL}=\mathtt{false})\,;Q,\Delta t\Big) = (?\mathrm{expBL}=\mathtt{true})\,;\mathrm{cond}\Big(P,\Delta t\Big)\,|\,(?\mathrm{expBL}=\mathtt{false})\,;\mathrm{cond}\Big(Q,\Delta t\Big)$$

$$\mathrm{cond}\left(\overset{\mathtt{n}}{\underset{i=1}{\mathcal{R}}}P,\Delta t\right) = \begin{cases} \mathrm{cond}\Big(P,\Delta t\Big)\,;\overset{\mathtt{n-1}}{\underset{i=1}{\mathcal{R}}}P & \text{if } n\geq 1 \\ \overset{\mathtt{n}}{\underset{i=1}{\mathcal{R}}}P & \text{otherwise} \end{cases}$$

$$\mathrm{cond}\left(\overset{\mathtt{expBL}\neq\mathtt{true}}{\underset{\geq 1}{\mathcal{R}}}P,\Delta t\right) = \mathrm{cond}\Big(P,\Delta t\Big)\,;\overset{\mathtt{expBL}\neq\mathtt{true}}{\underset{\geq 0}{\mathcal{R}}}P$$

$$\mathrm{cond}\left(\overset{\mathtt{expBL}\neq\mathtt{true}}{\underset{\geq 0}{\mathcal{R}}}P,\Delta t\right) = \begin{cases} \mathrm{cond}\Big(P,\Delta t\Big);\overset{\mathtt{expBL}\neq\mathtt{true}}{\underset{\geq 0}{\mathcal{R}}}P & \text{if } expBL=\mathtt{true} \\ \overset{\mathtt{expBL}\neq\mathtt{true}}{\underset{\geq 0}{\mathcal{R}}}P & \text{otherwise} \end{cases}$$

$$\mathrm{cond}\Big(X := P,\Delta t\Big) = \mathrm{cond}\Big(P[X/X := P],\Delta t\Big) \qquad \mathrm{cond}\Big(P\,\|_{tc}\,Q,\Delta t\Big) = \mathrm{cond}\Big(P,\Delta t\Big)\,\|_{tc}\,\mathrm{cond}\Big(Q,\Delta t\Big)$$

$$\mathrm{cond}\Big(P\oint Q,\Delta t\Big) = \mathrm{cond}\Big(P,\Delta t\Big)\oint\mathrm{cond}\Big(Q,\Delta t\Big) \qquad \mathrm{cond}\Big(P\,|||\,Q,\Delta t\Big) = \mathrm{cond}\Big(P,\Delta t\Big)\,|||\,\mathrm{cond}\Big(Q,\Delta t\Big)$$

$$\mathrm{cond}\Big(@ti_{\mathtt{hh:mm:ss:ms}} \hookrightarrow P_i,\ i\in\{1,\ldots,n\},\Delta t\Big) = @ti_{\mathtt{hh:mm:ss:ms}} \hookrightarrow P_i,\ i\in\{1,\ldots,n\}$$

$$\mathrm{cond}\Big(@ei_S \hookrightarrow P_i,\ i\in\{1,\ldots,n\},\Delta t\Big) = @ei_S \hookrightarrow \mathrm{cond}\Big(P_i,\Delta t\Big),\ i\in\{1,\ldots,n\}$$

$$\mathrm{cond}\Big(P\parallel\odot(@e_S\nearrow Q\searrow\odot),\Delta t\Big) = \mathrm{cond}\Big(P,\Delta t\Big)\parallel\odot(@e_S\nearrow\mathrm{cond}\Big(Q,\Delta t\Big)\searrow\odot)$$

Table 6: Definition of function $\mathrm{cond}(P,\Delta t)$.

According to the rule (TDD), the process P_i is performed when the value of the system time is equal to $ti_{\mathtt{hh:mm:ss:ms}}$.

Event-driven dispatch is a process relation in which the i-th process is triggered by a system event $@ei_S$. It is denoted as follows:

$$@ei_S \hookrightarrow P_i,\ i\in\{1,\ldots,n\}$$

When the event occurred is ei_S, then the process P_i is performed, rule (EDD).

Interrupt is a process relation in which a process is temporarily held by another with higher priority. The term describing that the process P is interrupted by the process Q when the event $@e_S$ is captured at interrupt point $\odot$ will be represented by

$$P\parallel\odot(@e_S\nearrow Q\searrow\odot)$$

If the event e_S is captured (rule (INTER1)), the process $P\parallel\odot(@e_S\nearrow Q\searrow\odot)$ will behave as Q, and after that the process will continue its execution behaving as P. If the event e_S is not captured and P can perform an action (rules (INTER2) and (INTER3)), the process $P\parallel\odot(@e_S\nearrow Q\searrow\odot)$ will also perform it.

4. Conclusions

In this paper we have presented the algebraic formalism STOPA. The main new feature of our language with respect to previous work is the ability to represent *stochastic time*, that is, to consider that time is not given by a fix amount but that it depends on a certain probability distribution function. In order to show the usefulness of our formalism, we have formally describe the memorization processes. Unfortunately, due to space limitations, we could not include this system in this paper. The formal description can be found in the extended version of the paper.

We contemplate two lines for future work. First, we have to perform a more thorough study of the semantic framework. In particular, it would be very adequate to define bisimulations semantics to allow the comparison of different representations of a cognitive process. Besides, we plan to study in a detailed way other cognitive processes and systems, so that we can better test the capabilities of STOPA. Actually, we have started to give a formal description of the algorithms and systems presented in [23].

References

[1] J. Baeten and C. Middelburg. *Process algebra with timing*. EATCS Monograph. Springer, 2002.

[2] J. Baeten and W. Weijland. *Process Algebra*. Cambridge Tracts in Computer Science 18. Cambridge University Press, 1990.

[3] J. Bergstra, A. Ponse, and S. Smolka, editors. *Handbook of Process Algebra*. North Holland, 2001.

[4] M. Bernardo and R. Gorrieri. A tutorial on EMPA: A theory of concurrent processes with nondeterminism, priorities, probabilities and time. *Theoretical Computer Science*, 202:1–54, 1998.

[5] M. Bravetti, M. Bernardo, and R. Gorrieri. Towards performance evaluation with general distributions in process algebras. In *CONCUR'98, LNCS 1466*, pages 405–422. Springer, 1998.

[6] M. Bravetti and R. Gorrieri. The theory of interactive generalized semi-Markov processes. *Theoretical Computer Science*, 282(1):5–32, 2002.

[7] D. Cazorla, F. Cuartero, V. Valero, F. Pelayo, and J. Pardo. Algebraic theory of probabilistic and nondeterministic processes. *Journal of Logic and Algebraic Programming*, 55(1–2):57–103, 2003.

[8] R. Cleaveland, Z. Dayar, S. Smolka, and S. Yuen. Testing preorders for probabilistic processes. *Information and Computation*, 154(2):93–148, 1999.

[9] P. D'Argenio, J.-P. Katoen, and E. Brinksma. An algebraic approach to the specification of stochastic systems. In *Programming Concepts and Methods*, pages 126–147. Chapman & Hall, 1998.

[10] J. Davies and S. Schneider. A brief history of timed CSP. *Theoretical Computer Science*, 138:243–271, 1995.

[11] R. v. Glabbeek, S. Smolka, and B. Steffen. Reactive, generative and stratified models of probabilistic processes. *Information and Computation*, 121(1):59–80, 1995.

[12] N. Götz, U. Herzog, and M. Rettelbach. Multiprocessor and distributed system design: The integration of functional specification and performance analysis using stochastic process algebras. In *16th Int. Symp. on Computer Performance Modelling, Measurement and Evaluation (PERFORMANCE'93), LNCS 729*, pages 121–146. Springer, 1993.

[13] P. Harrison and B. Strulo. SPADES – a process algebra for discrete event simulation. *Journal of Logic Computation*, 10(1):3–42, 2000.

[14] J. Hillston. *A Compositional Approach to Performance Modelling*. Cambridge University Press, 1996.

[15] C. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.

[16] N. López and M. Núñez. A testing theory for generally distributed stochastic processes. In *CONCUR 2001, LNCS 2154*, pages 321–335. Springer, 2001.

[17] N. López, M. Núñez, and F. Rubio. An integrated framework for the analysis of asynchronous communicating stochastic processes, 2004. To appear in *Formal Aspects of Computing*.

[18] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[19] X. Nicollin and J. Sifakis. An overview and synthesis on timed process algebras. In *Computer Aided Verification'91, LNCS 575*, pages 376–398, 1991.

[20] M. Núñez. Algebraic theory of probabilistic processes. *Journal of Logic and Algebraic Programming*, 56(1–2):117–177, 2003.

[21] M. Núñez and D. de Frutos. Testing semantics for probabilistic LOTOS. In *Formal Description Techniques VIII*, pages 365–380. Chapman & Hall, 1995.

[22] M. Núñez, D. de Frutos, and L. Llana. Acceptance trees for probabilistic processes. In *CONCUR'95, LNCS 962*, pages 249–263. Springer, 1995.

[23] M. Núñez, I. Rodríguez, and F. Rubio. Towards the identification of living agents in complex computational environments. In *2nd IEEE Int. Conf. on Cognitive Informatics*, pages 151–160. IEEE Computer Society Press, 2003.

[24] F. L. Pelayo, F. Cuartero, V. Valero, and D. Cazorla. An example of performance evaluation by using the stochastic process algebra ROSA. In *7th Int. Conf. on Real-Time Systems and Applications*, pages 271–278. IEEE Computer Society Press, 2000.

[25] G. D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Computer Science Department, Aarhus University, 1981.

[26] G. Reed and A. Roscoe. A timed model for communicating sequential processes. *Theoretical Computer Science*, 58:249–261, 1988.

[27] Y. Wang. On cognitive informatics. In *1st IEEE Int. Conf. on Cognitive Informatics*, pages 34–42. IEEE Computer Society Press, 2002.

[28] Y. Wang. The Real Time Process Algebra (RTPA). *Annals of Software Engineering*, 14:235–274, 2002.

[29] Y. Wang. Using process algebra to describe human and software behaviors. *Brain and Mind*, 4:199–213, 2003.

[30] W. Yi. CCS+ Time = an interleaving model for real time systems. In *18th ICALP, LNCS 510*, pages 217–228. Springer, 1991.