SPADES—a process algebra for discrete event simulation *

Peter G. Harrison and Ben Strulo
JLC 98-30e,2nd revision 4/99

29 April 1999

Abstract

We present a process algebra, SPADES, based on Milner's CCS, which may be used to describe discrete event simulations with parallelism. It is able to describe the passing of time and probabilistic choice, either discrete, between a countable number of processes, or continuous, to choose a random amount of time to wait. Its operational semantics is presented as a labelled transition system and we discuss equivalences over this operational semantics that imply axioms that can be used to compare and transform processes formally. We discuss notions of equivalence over simulations and the meaning of non-determinism in the context of the specification of a simulation. The algebra is applied to describe quantitatively a range of communicating systems.

1 Introduction

Consider the implementation of a complex system with computerised components, such as a telephone network or parallel computer architecture. Typically work will start with some type of specification of the system as a whole. Such a specification may never be completely explicit but we certainly require some underlying understanding of the whole system. This will include parts of the system (we will call them modules) which are to be implemented and other parts (we will call them workloads) that represent the demands of the environment on the modules, such as customer calls in our telephone example. From this, the implementor will proceed to specifications of the actual modules, such as digital exchanges and their software. Finally executable programs (or pieces of hardware or a mixture) will be constructed for these modules.

Now consider the process of making some model of the performance of such a system. We will refer to such a model as a simulation. By this we mean an explicit algorithmic model which may be used either for analysis or for execution to determine performance experimentally. We therefore seek a formalism that is both high level and has a formal semantics to make the behaviour of the language explicit and rigorous. Existing languages with the facilities to describe simulations often have complex behaviour described in terms of things like priority queues and event lists. Inevitable ambiguities in such informal descriptions can lead to unexpected behaviour or worse, to simulations which are not true to their specification. Moreover, in addition, we hope to derive from a formal semantics, tools to analyze simulations. If we can assign a meaning to a simulation then we can say that two simulations are equal if their meanings are equal. Then we can perform mechanical simplification and abstraction or at least prove the validity of our manual simplifications and abstractions. This parallels the use of formal semantics in conventional programming languages, as discussed in say [21].

We also want our language to be high level. We could certainly consider Generalised Semi-Markov Processes (GSMPs) as a semantics for simulation. But a GSMP description is very far from any current notion of system specification. Given a system of any complexity it is very hard to see that a GSMP description is a correct model of that system. We need to find a formalism nearer to specification, more high level and thus more natural for describing real complex concurrent systems.

*Not to be confused with the more recent process algebra of the same name [6]

Surveying the literature suggests process algebras as one of the leading contenders [22, 27]. Many approaches to extending process algebras with actual time values [5, 1, 4, 10, 30, 36, 24] or with time and probabilities [9] have been discussed with a view to describing performance-related issues. We use a selection from these approaches together with new techniques to obtain a process algebra—SPADES—which may be used to describe discrete event simulations. We take a relatively high level language that has a formal semantics of parallelism, viz. CCS, and add constructs to represent a real-number time and various probabilistic features. We then extend the formal semantics to cover these additions. In this way we can derive semantically sound axioms relating sub-expressions of a program. Such axioms can form the basis of mechanisable program transformations which, for example, may allow simpler or more efficient or more mathematically tractable simulations to be derived from another with a clearer specification. Again this compares to program transformation in declarative languages [7].

Stochastic process algebras are proposed as a unified formalism for the quantitative and qualitative specification of systems. From its conception as basic temporal or probabilistic extensions of regular process algebras, the field of stochastic process algebras has been maturing as a field of research in its own right. The breadth of SPA's has increased, both towards those that are suitable for mathematical analysis, such as Markovian process algebras [14, 11] and those that are expressive enough to specify realistic systems [18, 20, 12]. The ultimate goal would be to develop an algebra that is both expressive and descriptive enough to describe a wide class of systems, yet have just enough formal restraint to permit efficient analysis.

The abilities to specify function and performance tend to be orthogonal, being catered for by the process algebra and the stochastic components of the SPA somewhat independently. Efforts to generalise the applicability of SPA's in the functional domain include such things as building upon more powerful process algebras (e.g. [28, 2]). Generalisations in the stochastic domain include explicit specification of probabilities or generalising the distributions of delays (e.g. [20, 29]) SPADES differs from the majority of other stochastic process algebras in that it is defined in terms of three almost disjoint transition systems, a labelled, an evolution and a probabilistic one. In this way, the notion of delay, probability and action are disjoint. Other SPAs tend to fuse the notion of action probability and delay into a single concept of an "event" which can occur only at a particular rate. SPADES's approach is arguably a more natural one for modelling purposes. With actions representing computation, probabilistic choice to model outcomes and evolution to model delays, it follows closely the discrete event simulation methodology of system specification with the added benefits of a rigorous foundation.

The extended versions of the specification language LOTOS are the forms of SPA closest related to SPADES. They have the benefit of being founded upon a popular and widely used system specification language but tend to be tailored to describing communication systems. SPADES has been designed to be as general and expressive as possible whilst maintaining a theoretical foundation which, for example, gives us axiomatisations and notions of equivalence that can be used to model the performance or behaviour of the system using analytical techniques [16].

SPADES is presented in the traditional way for process algebras, using a labelled transition system for its semantics and deriving strong equivalences, weak equivalences and congruences. However, the mathematical details are far more complex due to the integration of the stochastic components. To achieve full generality requires the measure-theoretic axiomatisation of probability theory and its inclusion in a conventional CCS-style semantics.

In the next section we discuss the SPADES language informally and present its syntax. In section 3 we present the operational semantics formally and show it has the properties we expect. We next introduce bisimulation equivalences over this operational semantics, strong in section 4 and weak, abstracting from internal behaviour, in section 5. Finally we show the power of the language by a series of examples.

2 Language

First we give the language syntax and informally present the intended meaning.

Our language is defined so that CCS is a sub-calculus in the sense that the operational semantics of the pure CCS terms in our language is the same as in CCS itself. As in CCS, communication and synchronisation are represented via an alphabet of atomic actions or labels which we will write Λ. Each action has a conjugate action represented by an overline and the conjugate to the conjugate is the original action. There is also a distinguished self-conjugate, invisible (or silent) action which represents internal activity of a process invisible to the outside. Thus we have visible actions in $\Delta = \Lambda \cup \overline{\Lambda}$ and a complete alphabet $\mathrm{Act} = \Delta \cup \{\tau\}$ including the names, the co-names and the invisible action.

Our language is presented as an algebra over two sorts. We use two sets of variables, PVar of sort *Process expression*, and TVar of sort *Time*. We use an algebra of time expressions giving a set of expressions TExp. This has the operator $+$ indicating addition and $\dot{-}$ indicating non-negative subtraction i.e.

$$x \dot{-} y = \begin{cases} x - y & x \geq y \\ 0 & \text{otherwise} \end{cases}$$

The other symbols are 0 and an arbitrary number of other constant symbols. We will assume that these constants are non-negative real numbers (we will write $\mathbb{R}^+$). Equality for this sort is defined in the usual way. We will use time expressions and their values interchangeably.

Our language is then standard (basic) CCS drawn directly from [22] but with this new sort of Time and some additional operators. We define the process expressions as

Definition 1 *The set of process expressions,* PExp*, is the least set including*

$$X \qquad \mathsf{nil} \qquad (t).P \qquad \alpha.P \qquad \alpha[s \leftarrow t].P \qquad \mathcal{R}[s \leftarrow f].P$$

$$\sum_{i \in I} [q_i]P_i \qquad P + Q \qquad P \mid Q \qquad P[S] \qquad P \backslash A \qquad \mathsf{rec}(X = P)$$

where $X \in \mathsf{PVar}$*,* $P, Q \in \mathsf{PExp}$*,* $s \in \mathsf{TVar}$*,* $t \in \mathsf{TExp}$*,* $\alpha \in \mathrm{Act}$*,* I *is a finite indexing set with the* $P_i \in \mathsf{PExp}$*,* S *is a relabelling function* $S : \mathrm{Act} \to \mathrm{Act}$*,* $A \subset \mathrm{Act}$ *is a set of actions with* $\tau \notin A$*,* q_i *are probabilities in* $[0, 1]$ *with* $\sum q_i = 1$*, and* f *is a probability density function over the* $\mathbb{R}^+$*.*

We will use $P \triangleq E$ to assign the process expression E a name P. We will use the convention that processes (and process variables) are represented by names with an initial capital and will represent actions by names with an initial small letter. We go through the syntactic constructs giving their informal meaning.

First, nil is a process constant representing a deadlocked or terminated process which can perform no action and must simply delay or idle from now on. $(t).P$ represents a process which must delay for t time units and then becomes process P while $\alpha.P$ represents a process which can perform an α action and then become P.

Thus a typical process using these operators is

$$P_1 \triangleq \mathit{start}.(2).\mathit{stop}.\mathsf{nil}$$

Here P_1 performs a *start* action, delays for two time units, performs a *stop* action and then terminates normally.

Next we have an action prefixing $\alpha[s \leftarrow t].P$ which also represents a process which can immediately perform an α action. If the α is performed the process becomes P but here the free variable s is replaced by the value t and so the process becomes $P\{t/s\}$. The essential difference is that in $\alpha.P$ the prefixing cannot delay (we say it is *impatient*). The process $\alpha[s \leftarrow t].P$ can delay for any time t' but records this delay and so becomes $\alpha[s \leftarrow t + t'].P$. This *patient* prefixing binds the variable s in P. Of course if s is not free in P then we need not concern ourselves with s or t, and for convenience we may write $\alpha[].P$ in this case (the brackets distinguishing patient from impatient prefixing).

Thus compare P_1 with

$$P_2 \triangleq start[].(2).stop.\mathsf{nil}$$

This process, once it has done a *start* action, will continue as for P_1. But instead it can delay its initial *start* action as long as necessary. Consider also

$$P_3 \triangleq start[s \leftarrow 0].(2+s).stop.\mathsf{nil}$$

This process will act like P_1 if it performs its *start* immediately. But it can delay for 3 units, say, and become

$$start[s \leftarrow 3].(2+s).stop.\mathsf{nil}$$

When this performs its *start* action it becomes $(2+3).stop.\mathsf{nil}$ and will hence have to delay 5 units before it can *stop*.

We view this delaying behaviour as analogous to that of the value-passing part of CCS. We imagine a global clock, broadcasting the passage of time, to which all processes waiting for a synchronisation listen. Then the performance of the action causes the time value variable to become instantiated to the passing of time it has heard from that clock. The process may act on that value later, perhaps by then delaying less if it has already been delayed.

$P + Q$ represents a process which waits until P or Q are ready to participate in some action. Then it will choose non-deterministically between them. Once we have shown that no ambiguity arises we will use $\sum_{i\in I} P_i$ to represent finite sums. Thus consider $short.P_1 + long.P_2$. This process offers a choice between two alternatives, controlled by its initial action.

$\sum_{i\in I}[q_i]P_i$ represents a process which will immediately and randomly become one of the P_i on the basis of the q_i probabilities. This is a straight-forward finite probabilistic choice not subject to any control. We will write this as $[p]P \dot{+} [q]Q$ in the binary case.

$\mathcal{R}[s \leftarrow f].P$ represents a process which can immediately sample a random variable with the specified density and become $P\{t/s\}$ for the t sampled from the random variable. Although this is a form of uncountable probabilistic branching, the resulting processes are highly restricted in that they are identical except in the values of certain time expressions and hence certain delays. We have formulated our algebra in terms of density functions because this gives a clearer definition of the probabilistic transition relation. It is also possible to present it more generally in terms of a probability distribution function. This is done by defining a measure over processes directly in terms of the measure of the random variable using structural induction over the form of the process. However, most random variables in common use do have density functions, especially if we allow δ-functions to represent discrete random variables[1].

The parallel construct $P \mid Q$ represents two processes placed in parallel and allowed to communicate and synchronise. We use the usual CCS rules for this. Similarly relabelling $P[S]$ and restriction $P \setminus A$ come directly from CCS. $P[S]$ behaves like P but with all its actions renamed by the function S. $P \setminus A$ behaves like P but is not allowed to perform any action in A.

We also wish to allow the power of recursion. Consider the process we will write

$$P_4 \triangleq start.(2).stop.P_4$$

This has a plain intuitive meaning: it performs as P_1 but then restarts immediately. We might like to define P_4 as some fix-point of this recursive equation (with $=$ instead of $\triangleq$). But we have not even defined equality, let alone proved that this fixpoint exists. We will not use equations to define processes. Instead we use this equation as a syntactic abbreviation for the process expression

$$\mathsf{rec}(X = start.(2).stop.X)$$

(called P_4) and define the behaviour of rec directly. The definition says, of course, that it behaves exactly like

$$start.(2).stop.\,\mathsf{rec}(X = start.(2).stop.X)$$

and hence can be unrolled as many times as necessary, giving the recursion we expected.

[1] These may also be represented by probabilistic sums if they have finite support

As a simple example to explain these constructs, we consider a situation where a sequence of tasks arrive randomly, forming a workload. Then some (we choose two) processors service the queue, first come first served.

First we define the Workload. The Workload chooses a random time (density function f_1) and then waits for that time. Then it restarts itself in parallel with a new Task.

$$Workload \triangleq \mathcal{R}[s \leftarrow f_1].(s).\,(Workload \mid Task)$$

The Task requests a processor, executes for a random time, and then releases the processor and terminates.

$$Task \triangleq \overline{procreq}[].\,\mathcal{R}[s \leftarrow f_2].(s).procrel.\mathsf{nil}$$

The Processor accepts a request, accepts a release and restarts.

$$Proc \triangleq procreq[].\overline{procrel}[].Proc$$

Then our system is

$$System \triangleq Workload \mid Proc \mid Proc$$

2.1 Algebraic details

We will include and omit brackets as usual to remove ambiguity or to improve readability. In particular we will use the convention that prefixing binds most tightly followed by unary suffixing operators $\backslash A$ and $[S]$. These are followed by $\sum\!\!\!\!\cdot$ and $\dot{+}$, then $\sum$ and $+$, and finally $|$.

We define free variables of either sort in the obvious way. Process variables can be used throughout for sub-processes and become bound by being used in $\mathsf{rec}(X = P)$ expressions (where the free process variables of P remain free except for X which is bound). Time variables are introduced in the time expressions used in prefixings while they are bound by being used in $\alpha[s \leftarrow t].P$ expressions. Here time variables in t are free while free time variables in P remain free except for s which is bound. Time variables are also bound by being used in $\mathcal{R}[s \leftarrow f].P$ expressions.

We assume syntactic identity (represented $\equiv$) is modulo change of name of bound variables of either sort (α-conversion in the λ-calculus). Thus $\alpha[s \leftarrow t].(s).\mathsf{nil} \equiv \alpha[s' \leftarrow t].(s').\mathsf{nil}$ and $\mathsf{rec}(X = P(X)) \equiv \mathsf{rec}(X' = P(X'))$ as long as there is no capture of free variables (i.e. X' is not free in $P(X)$, X is not free in $P(X')$, and the same for s and t).

We identify the set of *processes*, Proc to be the closed process expressions i.e. those with no free variables of either sort. We use $P\{t/s\}$ to mean P with t replacing the free occurrences of variable s i.e. standard syntactic substitution.

As the algebra stands, expressions like $\mathsf{rec}(X = X)$ are allowed. There seems to be no particular advantage to allowing such unguarded recursion. No transitions may be inferred for it and it is not our model for divergence. So following e.g. [26] we will simply disallow unguarded recursion. We require a recursive expression to have its free variable "guarded" by some action so that each recursive unwinding requires at least one event before it can be repeated. We say a process variable X is *guarded* in P iff all occurrences of X are in some subterm $\alpha.Q$ or $\alpha[s \leftarrow t].Q$. Then we say that $\mathsf{rec}(X = P)$ is well-formed only if X is guarded in P. This is straightforwardly decidable.

2.2 Language features

Our language is based on the timed CCS of [35]. It takes its impatient actions from Moller and Toft's temporal CCS in [23] though their calculus does not generalise directly to real number time. Our language is a parallel development to [4] except that we have only two options for the upper limit to the time of execution of an action. Our impatient prefixing is equivalent to Chen's $\alpha|_0^0$ while our idling prefixing is equivalent to his $\alpha(s)|_0^\infty$.

Our language has a similar purpose to TIPP proposed in [8]. The most important difference is in our treatment of non-determinism. The models of TIPP execute non-deterministic choices with uniform probability while in ours such choices must be explicitly resolved.

Our rules for the temporal behaviour of processes do not mean time behaves in exactly the way expected for the time required by a program to execute. In this it is similar to the calculi mentioned above. For example it is easy to write programs which perform an infinite amount of computation in finite time e.g. $\mathsf{rec}(X = \mathit{tick}.X)$. This process can never evolve since it must always *tick* impatiently. Thus it forces time to stop for the whole system. Consider also $(\mathit{tick}.\mathsf{nil}) \setminus \{\mathit{tick}\}$. Because the *tick* here is impatient this process cannot evolve. But because it is restricted it cannot perform *tick* either. Thus this process is simply deadlocked and similarly prevents time passing in the whole system.

We justify this by interpreting temporal evolution as representing simulated time rather than the time in which calculation takes place. If we run a simulation we do not expect the time the simulation takes to run to be related to anything in the system it is simulating. The simulation may deadlock or loop and take forever to simulate no time at all in the real system. Or it may run very fast and simulate the whole history of the real system in very little calculation time. Generally, one abstracts from this simulation execution time by ignoring it. The time evolution of our processes represents time passing in the simulated system, i.e. the numbers output by some time-advance procedure in our simulation. Thus the semantics we give is the semantics of a discrete event simulation.

Having defined our syntax and given some feeling for its intended behaviour we define its semantics formally.

3 Semantics

3.1 Transition system

The operational semantics of these processes are described in terms of three sorts of transitions between processes. We define probabilistic transitions so that they will be resolved first. The idea is that any probabilistic part may be assumed under any circumstances to have already been resolved before any other actions are considered. We will show that our transition system has this property (proposition 1).

Then we have the standard CCS type of *labelled* transitions which are labelled from Act and happen non-deterministically following the usual behaviour of a CCS-type language. Transitions labelled with τ must happen immediately i.e. before any evolution can take place. We show this property as proposition 4. Note that our impatient actions must also happen before any evolution. We will call impatient actions and τ actions *immediate*.

When no more immediate transitions are possible the system becomes *stable*. Then, all parts of the process synchronously undergo a timed *evolution* transition via their (t) actions. This represents the passing of time and continues until further probabilistic or immediate transitions are possible. To be sure of this we need to show that the process cannot evolve past the potential of performing a sampling or immediate transition. This is shown in proposition 3.

The assumption that τ actions take priority over evolution is sometimes called the Maximal Progress assumption and is discussed in [35]. In fact our approach is different to that described in [35] in that we have explicit actions that are not τ but that still have priority over evolution (are impatient). This makes it easy to write processes such as $\mathit{Clock} \triangleq \mathit{tick}.(1).\mathit{Clock}$ which ticks once a time unit. Here each tick cannot be made to delay and so the clock cannot be made to run slow.

The possible transitions are described via inference systems. We say the transitions are the least relation between processes that satisfies the given inference rules. We define the labelled transitions first as they are the simplest and most familiar. These have an interleaving semantics over parallel constructs, as in CCS, i.e. only one process can progress at a time. In contrast, evolution is truly concurrent, or synchronous, in that all parallel processes evolve together under a single global clock. Similarly, probabilistic transitions are truly concurrent.

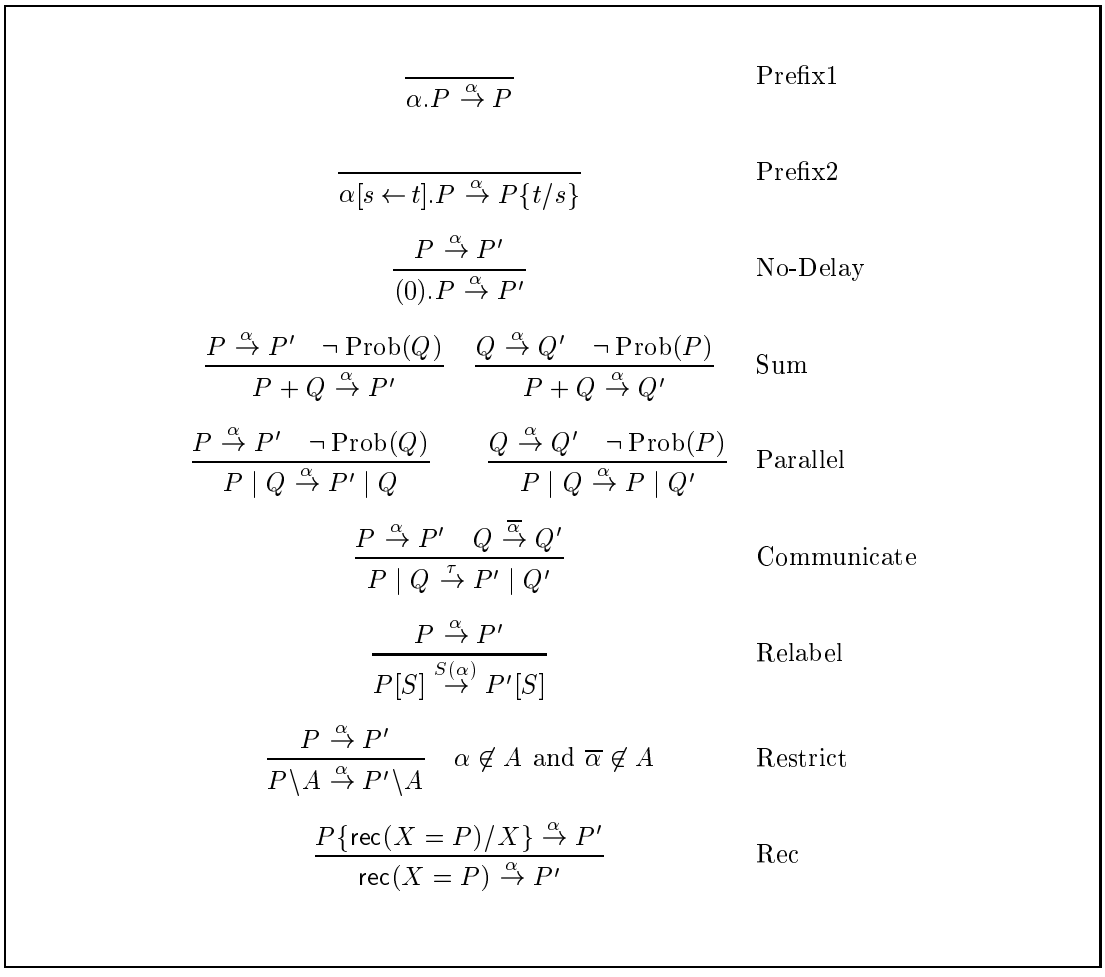

Figure 1: Labelled transitions

3.2 Labelled transitions

These are defined in Figure 1 and are very similar to the standard CCS rules. In the Sum and Parallel rules we use preconditions like $\neg \mathrm{Prob}(P)$. This predicate is not formally defined until we define our probabilistic transitions in the next section but informally it means that P cannot perform a probabilistic transition (is not Probabilistic). Thus these preconditions force all probabilistic choices to be resolved before any labelled transitions. The Prefix2 rule shows the way in which time variables are bound by the execution of an action. The No-Delay rule allows zero delays to be disregarded when any transition is to be inferred. We will thus later be able to deduce that $(0).P = P$.

3.3 Probabilistic transitions

We would like to define our probabilistic transitions in a similar way and deduce a probability distribution over transitions. But consider the process $[\frac{1}{2}]P \dot{+} [\frac{1}{2}]P$. A natural transition inference system will infer a transition with probability half to P for this process but by two possible inferences. We need to keep these two transitions separate so that each has a probability half but the total probability of a transition to P is 1. So we distinguish our transitions by a special index shown as a subscript. This index represents the probability space over which our distribution is defined. Choosing an index chooses the particular transition (and inference) from all the possible probabilistic transitions (and inferences) for the process.

We define the probabilistic transitions in Figure 2. We name the rules for use in proofs. Each transition has an associated probability measure, shown above it, and its associated index,

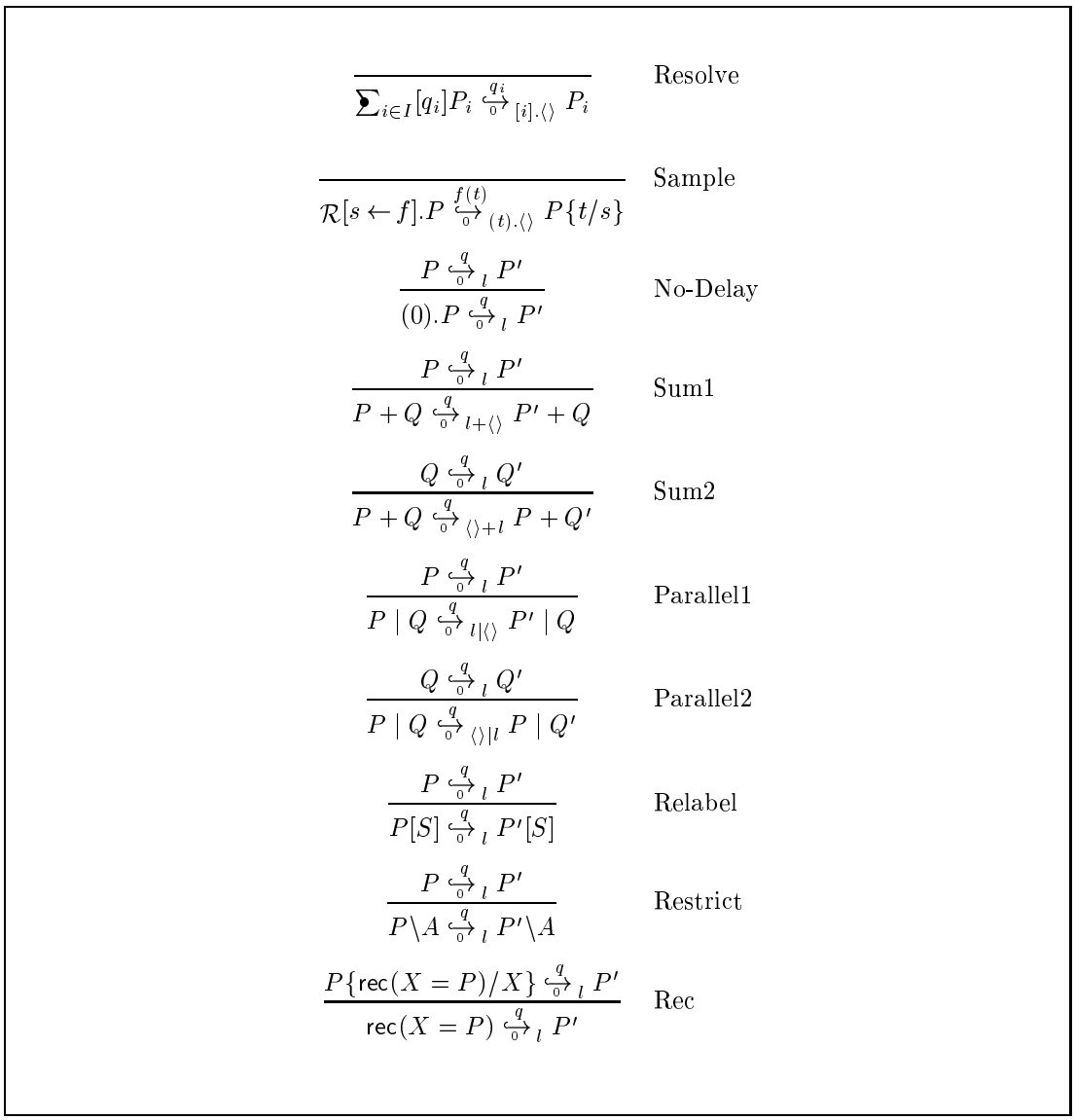

Figure 2: Single step probabilistic transitions

$$\frac{P \overset{p}{\underset{0}{\hookrightarrow}}_l P' \quad P' \overset{q}{\hookrightarrow}_m P''}{P \overset{p.q}{\hookrightarrow}_{\mathrm{lf}(l,m)} P''} \quad \text{Mult-Samples}$$

$$\frac{\neg\, \mathrm{Prob}(P)}{P \overset{1}{\hookrightarrow}_{\langle\rangle} P} \quad \text{No-Samples}$$

Figure 3: Probabilistic transitions

shown as a subscript. Indices come from the set Index_0 which we define:

Definition 2 *The set of* index atoms *is defined by*

$$\mathsf{Atom} = \{[i] : i \in \mathbb{N}\} \cup \{(t) : t \in \mathbb{R}^+\}$$

The set of indices, Index, *is the least set including*

$$\langle\rangle \qquad a.l \qquad l \mid l' \qquad l + l'$$

where $l, l' \in \mathsf{Index}$, $a \in \mathsf{Atom}$.

The set of single step indices, Index_0, *is the subset of indices containing at most one atom.*

Thus the transitions are a relation from $\mathsf{Proc} \times [0,1] \times \mathsf{Index}_0 \times \mathsf{Proc}$.

The index carries the part of some complex process that has given rise to this probabilistic transition. The atom, a, indicates how the probabilistic choice has been resolved while the rest of the index skeleton mirrors the rest of the process which has not resolved any probabilistic choices it may hold.

Having defined when a process can perform a probabilistic transition we are in a position to define the Prob predicate we used earlier:

Definition 3 *We say a process* P *is* probabilistic, $\mathrm{Prob}(P)$, *iff* $P \overset{p}{\underset{0}{\hookrightarrow}}_l P'$ *for some* p, l, P'.

This transition relation captures our fundamental idea of how we wish single probabilistic transitions to operate but it is not clear how it represents probabilities. It is even non-deterministic since a sum or parallel combination with two probabilistic sub-terms is given two possible transitions. To clarify the position we combine these multiple transitions into a relation from $\mathsf{Proc} \times [0,1] \times \mathsf{Index} \times \mathsf{Proc}$ with another inference system, Figure 3. Here we adopt the convention that processes with no probabilistic choices to make have the trivial transition $P \overset{1}{\hookrightarrow}_{\langle\rangle} P$. Note also that a simple induction shows that we have $P \overset{p}{\hookrightarrow}_l P'$ implies $\neg\, \mathrm{Prob}(P')$ since that is the case for both inference rules.

The indices now represent all the probabilistic choices that go to make up the complete transition. Here we use the lf function to extend the index as we generate the final transition.

Definition 4 *The* label extension function $\mathrm{lf} : \mathsf{Index}_0 \times \mathsf{Index} \to \mathsf{Index}$ *is defined by*

$$\begin{aligned}
\mathrm{lf}(\langle\rangle, m) &= m \\
\mathrm{lf}(a.\langle\rangle, m) &= a.m \\
\mathrm{lf}(l + l', m + m') &= \mathrm{lf}(l, m) + \mathrm{lf}(l', m') \\
\mathrm{lf}(l \mid l', m \mid m') &= \mathrm{lf}(l, m) \mid \mathrm{lf}(l', m')
\end{aligned}$$

Note that this definition covers sufficient cases since if $P \overset{p}{\underset{0}{\hookrightarrow}}_{l+l'} Q$ then Q is of the form $Q' + Q''$ and similarly for $|$.

By our guardedness restriction we know that for every process there is some maximum depth of inference both for $\overset{}{\underset{0}{\hookrightarrow}}$ and for $\hookrightarrow$. We will frequently perform induction on this maximum depth, first for $\hookrightarrow$ and within that for $\overset{}{\underset{0}{\hookrightarrow}}$.

This is the usual sort of presentation for operational semantics of process algebras. We must demonstrate the sense in which it provides a well defined probabilistic semantics. In other words we wish to show we have defined some sort of probability density over the transitions from P. Thus we define

Definition 5 *For a given process, P,*

$$\mathsf{Index}_P = \{m \in \mathsf{Index} : \exists P', q \quad P \overset{q}{\hookrightarrow}_m P'\}$$

$p_P : \mathsf{Index} \to [0,1]$ *is*

$$p_P(m) = \begin{cases} q & \textit{iff} \quad \exists P' \quad P \overset{q}{\hookrightarrow}_m P' & m \in \mathsf{Index}_P \\ 0 & & \textit{otherwise} \end{cases}$$

and $q_P : \mathsf{Index}_P \to \mathsf{Proc}$ *is*

$$q_P(m) = P' \qquad \textit{iff} \quad \exists q \quad P \overset{q}{\hookrightarrow}_m P'$$

Note that our convention for non-probabilistic processes means that if $\neg\,\mathrm{Prob}(P)$ then

$$\mathsf{Index}_P = \{\langle\rangle\}$$

$$p_P(m) = \begin{cases} 1 & m = \langle\rangle \\ 0 & \text{otherwise} \end{cases}$$

$$q_P(\langle\rangle) = P$$

We need to show that these definitions are consistent in that they really do define functions and so we prove

Lemma 1 *If $P \overset{q}{\underset{0}{\hookrightarrow}}_l Q$ and $P \overset{q'}{\underset{0}{\hookrightarrow}}_l Q'$ then $q = q'$ and $Q \equiv Q'$.*

Proof

By induction on depth of inference. □

and so we obtain

Lemma 2 *If $P \overset{q}{\hookrightarrow}_m Q$ and $P \overset{q'}{\hookrightarrow}_m Q'$ then $q = q'$ and $Q \equiv Q'$.*

Proof

By induction on depth of inference looking at cases of the structure of P.

- Trivial if $\neg\,\mathrm{Prob}(P)$ since $q = q' = 1$ while $Q \equiv Q' \equiv P$.
- $\mathcal{R}[s \leftarrow f].P$, $\sum[q_i]P_i$, trivial since e.g. m specifies the random choice and the result holds inductively for $P\{t/s\}, P_i$.
- $P_1 + P_2$ — We have $m = m_1 + m_2$ and $P_1 + P_2 \overset{p_1 \cdot p_2}{\hookrightarrow}_{m_1+m_2} P_1' + P_2'$ so $P_1 \overset{p_1}{\hookrightarrow}_{m_1} P_1'$ and $P_2 \overset{p_2}{\hookrightarrow}_{m_2} P_2'$ and so the result is true inductively.
- $P_1 \mid P_2$ similar.
- all other cases simpler inductions

□

This shows we have some sort of probability density function. But over what can we integrate it? We start by treating it as describing a distribution over our index space. The set of indices associated with all the transitions from the given process form the sample space, i.e. it is this index which distinguishes the separate transitions. First we must define how we can integrate over this space. We define:

Definition 6 *The σ-field of* measurable sets of indices, $\overline{C}$ *is the least class defined by*

- $\{\langle\rangle\} \in \overline{C}$.
- *If* $i \in \mathbb{N}$ *and* $C \in \overline{C}$ *then* $\{[i].l : l \in C\} \in \overline{C}$.
- *If* $R \subseteq \mathbb{R}^+$ *is measurable and* $C \in \overline{C}$ *then* $\{(t).l : t \in R \;\; l \in C\} \in \overline{C}$.
- *If* $C_1, C_2 \in \overline{C}$ *then* $\{l_1 + l_2 : l_1 \in C_1 \;\; l_2 \in C_2\} \in \overline{C}$.
- *If* $C_1, C_2 \in \overline{C}$ *then* $\{l_1 \mid l_2 : l_1 \in C_1 \;\; l_2 \in C_2\} \in \overline{C}$.
- *If* $C_i \in \overline{C}$ *for* $i \in \mathbb{N}$ *then* $\cup_i C_i \in \overline{C}$.
- *If* $C_i \in \overline{C}$ *for* $i \in \mathbb{N}$ *then* $\cap_i C_i \in \overline{C}$.

This is plainly a σ-field.

Then it is straightforward to define integration and the probability associated with any such set of indices by

Definition 7 *The* index probability measure, $\mu^* : \mathsf{Proc} \times \overline{C} \to [0,1]$ *is the Lebesgue integral*

$$\mu^*(P, I) = \int_{l \in I} p_P(l)$$

Now this definition will only make sense if p_P is a measurable function. To show that it is we proceed as follows.

Consider first a process expression, Q, with only a single free time variable, s, (and no free process variables). Plainly its probabilistic transitions do not depend on the value of s and so we write

$$Q(s) \xrightarrow{p}_m Q'(s) \quad \text{iff} \quad \forall t \in \mathbb{R}^+ \; Q\{t/s\} \xrightarrow{p}_m Q'\{t/s\}$$

Clearly we have $p_{Q\{t/s\}}(m) = p_{Q(s)}(m)\{t/s\}$ and similarly for q. Now we prove

Lemma 3 *p_P is measurable.*

Proof

By induction on the maximum depth of inference of P, looking at cases of the structure of P.

- If $\neg \operatorname{Prob}(P)$ then trivial since both $\{\}$ and $\{\langle\rangle\}$ are measurable.
- $(0).Q$ trivial since $p_{(0).Q} = p_Q$ and this is measurable by induction.
- $Q[S]$, $Q \backslash A$, $\mathsf{rec}(X = Q)$ all trivial in the same way.
- $P \equiv \mathcal{R}[s \leftarrow f].Q$ — Then

$$p_P(l) = \begin{cases} f(t).p_{Q\{t/s\}}(m) & \text{if } l = (t).m \\ 0 & \text{otherwise} \end{cases}$$

But

$$p_{Q\{t/s\}}(m) = p_{Q(s)}(m)\{t/s\}$$

and this is measurable by induction, giving the result for p_P.

- $P \equiv \sum_{i \in I} [q_i] Q_i$ —
$$p_P([i].m) = q_i.p_{Q_i}(m)$$
and so is measurable by induction.

- $Q_1 + Q_2$ — Then
$$p_{Q_1+Q_2}(l) = \begin{cases} p_{Q_1}(m).p_{Q_2}(m') & \text{if } l = m_1 + m_2 \\ 0 & \text{otherwise} \end{cases}$$
and so the result follows inductively.

- $Q_1 \mid Q_2$ similar.

□

We now wish to move this structure over to Proc. Using the usual definition of $q_P^{-1}(S) = \{l \in \mathsf{Index}_P : q_P(l) \in S\}$, we define:

Definition 8 *The* measurable sets w.r.t. P *are*
$$\overline{B}_P = \{S \subseteq \mathsf{Proc} : q_P^{-1}(S) \in \overline{C}\}$$

The σ-field of measurable sets *is $\overline{B}$ where*
$$\overline{B} = \bigcap_{P \in \mathsf{Proc}} \overline{B}_P$$

Thus a set of processes is measurable w.r.t. P if the set of indices which drive P into it by probabilistic transitions is measurable, while a set of processes is measurable if it is measurable w.r.t. to any process.

Now we may define our probability measure over this σ-field as:

Definition 9 *The* process probability measure, $\mu : \mathsf{Proc} \times \overline{B} \to [0,1]$ *is*
$$\mu(P, S) = \mu^*(P, q_P^{-1}(S))$$

Thus $\mu(P, S)$ is the total probability that P may become a member of S by a probabilistic transition. Finally, to prove that $\mu(P, .)$ is a probability measure we need

Lemma 4 $\mu(P, \mathsf{Proc}) = 1$

Proof

Again by induction on maximum depth of inference of P, we look at cases of the structure of P.

- If $\neg \mathrm{Prob}(P)$ then trivial.

- $(0).Q$, $Q[S]$, $Q \backslash A$, $\mathsf{rec}(X = Q)$ all trivial since e.g. $\mu((0).Q, \mathsf{Proc}) = \mu(Q, \mathsf{Proc})$ which is 1 by induction.

- $\mathcal{R}[s \leftarrow f].Q$ — Then
$$\begin{aligned} \mu(P, \mathsf{Proc}) &= \mu^*(P, \mathsf{Index}_P) \\ &= \int_{l \in \mathsf{Index}_P} p_P(l) \\ &= \int_{(t).m \in \mathsf{Index}_P} f(t).p_{Q\{t/s\}}(m) \end{aligned}$$

$$= \int_{t\in\mathbb{R}^+} f(t).\int_{m\in \mathsf{Index}_{Q\{t/s\}}} p_{Q(s)}\,(m)\,\{t/s\}$$

by Fubini's theorem (in e.g. [19]). So by induction

$$\begin{aligned}\mu(P,\mathsf{Proc}) &= \int_{t\in\mathbb{R}^+} f(t).1\\ &= 1\end{aligned}$$

as required.

- $\sum_{i\in I}[q_i]Q_i$: similarly.
- $Q_1 + Q_2$: if either is not probabilistic then trivial induction. Otherwise

$$\begin{aligned}\mu(P,\mathsf{Proc}) &= \mu^*(P,\mathsf{Index}_P)\\ &= \int_{l\in\mathsf{Index}_P} p_P(l)\\ &= \int_{m_1\in\mathsf{Index}_{Q_1}\ m_2\in\mathsf{Index}_{Q_2}} p_{Q_1}(m_1).p_{Q_2}(m_2)\\ &= \int_{m_1\in\mathsf{Index}_{Q_1}} p_{Q_1}(m_1).\int_{m_2\in\mathsf{Index}_{Q_2}} p_{Q_2}(m_2)\\ &= 1\end{aligned}$$

by induction.

- $Q_1 \mid Q_2$: similarly.

□

Thus, for each process P, these probabilistic transitions induce a probability measure $\mu(P,.)$ over the σ-field of sets of processes $\overline{B}$. We have given this derivation for the measure only in terms of density functions because this gives a clearer definition of the probabilistic transition relation. In can also be presented more generally by defining the measure directly in terms of the measure of the random variable using structural induction over the form of the process. However, most random variables in common use do have density functions, especially if we allow δ-functions to represent discrete random variables.

3.4 Evolution transitions

Eventually no more immediate or probabilistic transitions will be possible. The only thing the process can then do is to evolve in time.

As in [35] a central aim of this design is to give the Maximal Progress property that τ actions always happen before evolution. The problem here is that it is difficult to prevent a parallel construct from evolving past a potential τ operation. Thus we would like to have a rule

$$\frac{P \overset{t}{\rightsquigarrow} P' \quad Q \overset{t}{\rightsquigarrow} Q'}{P \mid Q \overset{t}{\rightsquigarrow} P' \mid Q'}$$

But consider $P \equiv \alpha[].\mathsf{nil}$, $Q \equiv \overline{\alpha}[].\mathsf{nil}$. Both of these can delay arbitrarily but in parallel we wish them to be forced to act immediately without delay. In [35] this is done by "Timed Sorts". We use a slightly different approach.

First we define a single step evolution relation $\overset{t}{\rightsquigarrow}_\circ$. Note that we will not distinguish between time expressions and time values. The definitions are in Figure 4.

These rules do not allow the inference of a zero time evolution (trivial induction). Also we have no rules to allow $\alpha.P$ to evolve; it is this that makes it "impatient".

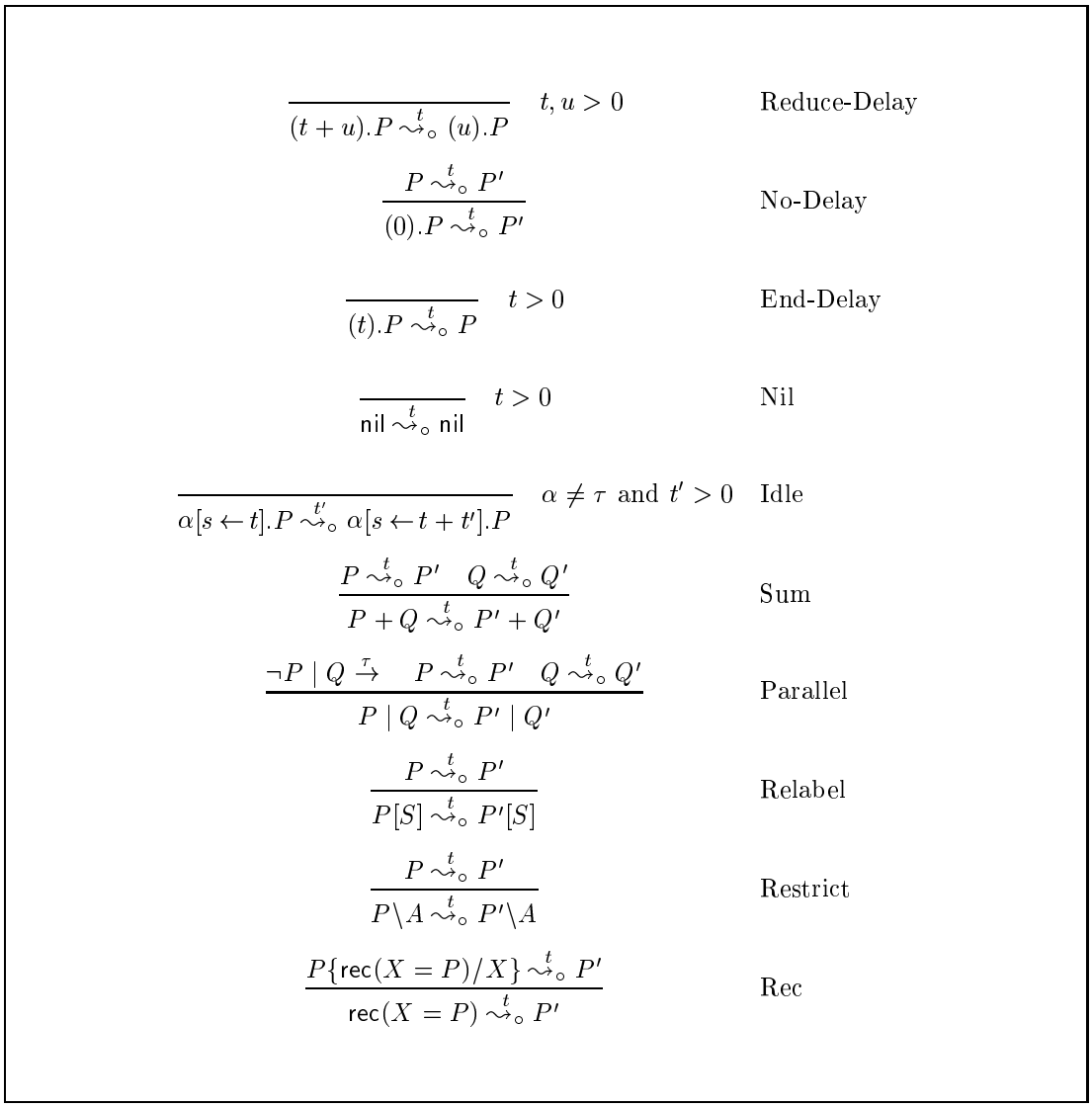

Figure 4: Single step evolution transitions

These rules capture our essential ideas about evolution. However they only allow evolutions between the simplest changes of syntactic form of processes. Thus for example we do have $(1).(1).P \stackrel{1}{\leadsto}_\circ (1).P \stackrel{1}{\leadsto}_\circ P$ but not $(1).(1).P \stackrel{2}{\leadsto}_\circ P$. This is necessary to ensure that parallel constructs are checked for immediate transitions at every stage at which they are possible. But we do wish to combine allowable transitions so we add a new level of transition. Our evolution transitions are thus defined in Figure 5. We now have $(1).P \stackrel{1}{\leadsto}_\circ P$ and so by Any-Delay we

$$\frac{P \stackrel{t}{\leadsto}_\circ P'}{P \stackrel{t}{\leadsto} P'} \quad \text{Any-Delay}$$

$$\frac{P \stackrel{t'}{\leadsto}_\circ P' \quad P' \stackrel{t}{\leadsto} P''}{P \stackrel{t+t'}{\leadsto} P''} \quad \text{Add-Delay}$$

Figure 5: Evolution transitions

have $(1).P \stackrel{1}{\leadsto} P$. But we also have $(1).(1).P \stackrel{1}{\leadsto}_\circ (1).P$ and so by Add-Delay we can infer $(1).(1).P \stackrel{2}{\leadsto} P$ as required.

4 Properties

We have given our processes a formal operational semantics so we must now show that this semantics has the properties we expect. Many of these have been discussed at length in the literature for temporal process algebras but we have the additional factor of probabilistic transitions. We have stated informally that if our processes are probabilistic they must do that first. We prove (by induction on the depth of inference for each possible syntactic construct of P):

Proposition 1 *If* $\quad \mathrm{Prob}(P) \quad$ *then* $\quad \neg P \stackrel{\alpha}{\to}$ *and* $\neg P \stackrel{t}{\leadsto}$.

Thus if probabilistic transitions are possible they are the only transitions possible and so happen before any others.

We list some other properties of our system and compare them to those noted by other authors for their temporal process algebra. In particular [25] gives a concise collection of important properties of various systems and we will adopt their terminology for these.

4.1 Time determinism

Most systems of timed process algebras have the property that time evolution introduces no non-determinism; this arises only from labelled actions. Our calculus has this property — we prove it in two steps, using straightforward inductions on the depth of inference:

Lemma 5 *If* $P \stackrel{t}{\leadsto}_\circ P'$ *and* $P \stackrel{t}{\leadsto}_\circ P''$ $\quad$ *then* $\quad P' \equiv P''$.

Proposition 2 *If* $P \stackrel{t}{\leadsto} P'$ *and* $P \stackrel{t}{\leadsto} P''$ $\quad$ *then* $\quad P' \equiv P''$.

Proof

Easy induction on depth of inference of $P \stackrel{t}{\leadsto} P'$ since determinacy is preserved at each step. □

4.2 Time additivity

Most other systems also require that if a process can delay for some period it can also delay for any smaller period ([36] — "time continuity"). Our calculus does have this property but our requirement that τ is immediate makes this rather more difficult than it might seem. We must show that a process that can evolve $P \stackrel{t+t'}{\rightsquigarrow} P'$ does not have the lurking capability of a $P \stackrel{t}{\rightsquigarrow} P'' \stackrel{\tau}{\rightarrow}$ which would imply $\neg\, P'' \stackrel{t'}{\rightsquigarrow} P'$.

We prove two intermediate lemmas, again by induction on depth of inference.

Lemma 6 *If* $P \stackrel{t}{\rightsquigarrow}_{\circ} P'$ *and* $P \stackrel{t'}{\rightsquigarrow}_{\circ} P''$ *with* $t' > t$ *and* $P' \stackrel{\alpha}{\rightarrow}$ *then* $P'' \stackrel{\alpha}{\rightarrow}$.

This lemma shows that although a process may be able to obtain new capabilities through evolution it can do so only at the end of its simple evolution period. Note however that this is not true for the extended evolution relation since e.g. $(1).\alpha[].P \stackrel{2}{\rightsquigarrow} \alpha[].P$.

Lemma 7 *If* $P \stackrel{t+t'}{\rightsquigarrow}_{\circ} P''$ *then* $\exists P'$ *with* $P \stackrel{t}{\rightsquigarrow}_{\circ} P'$ *and* $P' \stackrel{t'}{\rightsquigarrow}_{\circ} P''$.

The result we require now follows:

Proposition 3 $P \stackrel{t+t'}{\rightsquigarrow} P''$ *iff* $\exists P'$ *with* $P \stackrel{t}{\rightsquigarrow} P'$ *and* $P' \stackrel{t'}{\rightsquigarrow} P''$.

Thus evolution does behave as expected for a real number clock i.e. it reaches every intermediate time. This is also needed to ensure our naïve interpretation of the maximal progress assumption. We now know that an evolving process may evolve for a continuous interval either forever or to some maximum t. Once this t is reached it becomes a new process with either immediate or probabilistic transitions.

4.3 Deadlock-freeness

When a process terminates normally it becomes nil. Because this allows any evolution to continue this form of termination is used by a component when it has no further contribution to make to a simulation. But in our calculus we do have states (such as $\alpha.\mathsf{nil}$) which cannot evolve (because the prefixing is impatient). If such a process also cannot perform a labelled or probabilistic transition then it can do nothing. An example of such a process is, as already discussed, $(tick.\mathsf{nil})\backslash\{tick\}$. We consider it to represent an erroneous simulation which has reached a deadlock situation and cannot continue to reach its time advance procedure.

Thus our calculus does not have the property of "deadlock-freeness" from [25]. We will define a symbol to represent such programs

Definition 10 *The* time stop process *is*

$$\mathbf{0} \triangleq (\alpha.\mathsf{nil})\backslash\alpha$$

Alternatively we could have simply added $\mathbf{0}$ to our language with no transition rules.

4.4 Action urgency

As already discussed, we have both impatient and patient actions (the notion of "urgency" from [25]). However our calculus is unusual in that it also has the maximal progress ("minimal delay" or "tau-urgency") property.

Proposition 4 *If* $P \stackrel{\tau}{\rightarrow}$ *then* $\neg P \stackrel{t}{\rightsquigarrow}$.

This, together with proposition 1, means that expressions can be divided into three disjoint sets. Each state is either *probabilistic* or *immediate* or *evolving*.

4.5 Persistency

We do have the important and interesting "persistency" property that the passage of time cannot (alone) remove the capability of performing actions.

Proposition 5 *If* $P \overset{t}{\rightsquigarrow} Q$ *and* $P \overset{\alpha}{\rightarrow}$ *then* $Q \overset{\alpha}{\rightarrow}$.

Proof

We prove the result first for the single step relation by induction on the depth of inference of $P \overset{t}{\rightsquigarrow}_\circ P'$.

- Reduce-Delay: $P \equiv (t+u).Q$ and so $\neg P \overset{\alpha}{\rightarrow}$
- End-Delay & Nil: similarly.
- No-Delay: $P \equiv (0).Q$ and so $Q \overset{t}{\rightsquigarrow}_\circ P'$ and $P \overset{\alpha}{\rightarrow}$
  Thus $P' \overset{\alpha}{\rightarrow}$ by the inductive hypothesis, as required.
- Idle: $P \equiv \alpha[s \leftarrow t].Q$ and so the result is trivial.
- The other inferences are easy inductive steps.

The result is now obtained for the general relation by a straightforward Induction on the depth of inference of $P \overset{t}{\rightsquigarrow} Q$. □

Thus time evolution does not remove the capability to perform actions. It can, however, uncover new capabilities as in

$$a[].P + (1).b[].Q \overset{1}{\rightsquigarrow} a[].P + b[].Q$$

Capabilities disappear when some action (e.g. a τ action) makes the process change state. This is the mechanism for time-outs as in

$$a[].P + (1).\tau.Q \overset{1}{\rightsquigarrow} a[].P + \tau.Q \overset{\tau}{\rightarrow} Q$$

This process can perform the patient a action throughout the one time unit evolution period. At that time the impatient τ becomes available and must take place before the process can continue to evolve. Thus the availability of a is "timed-out" and the process becomes Q.

4.6 Finite-variability

In our calculus we can easily define processes that perform infinite computation in finite time. For example we may define the standard divergent process:

Definition 11

$$\Omega \triangleq \mathsf{rec}(X = \tau.X)$$

This performs an infinite sequence of τ and so will never be able to evolve. Hence it performs infinite calculation in zero simulated time. Thus we do not have the "finite-variability" ("non-Zenoness, well-timedness") property and also not the "bounded control" property. Like **0** this represents a simulation which is failing to correctly advance time. The difference is that Ω is continuing to loop uselessly instead of deadlocking.

4.7 Free variables

Finally we have an obligation to show that free variables are not going to cause a problem for our interpretation of processes as programs. We show that if we start with a closed process expression it will remain closed.

Lemma 8

If P is a process (i.e. a closed process expression) and $P \xrightarrow{\alpha} P'$ then P' is a (closed) process.

If P is a process and $P \hookrightarrow_l P'$ then P' is a (closed) process.

If P is a process and $P \overset{t}{\rightsquigarrow} P'$ then P' is a (closed) process.

4.8 The meaning of processes

We have given processes an operational semantics as transition systems. Thus they are an (uncountable) set of states, each state with one of three kinds of transitions from it. Either they are immediate states with a finite set of immediate transitions to other states, or they are probabilistic states which sample probabilistic distributions to choose their next state, or they are evolving states which evolve deterministically on the basis of a set of clocks (exposed (t) expressions) to some new state.

To relate this to discrete event simulation we consider the immediate transitions as representing internal computation and synchronization of the simulation. So a process (simulation) makes probabilistic choices and does internal computations until all its component agents become stable (i.e. unable to act immediately). Then all such agents bid a time-advance real number. The smallest of these is chosen and simulated time is advanced by that amount. Then further computations take place. Note that this simple interpretation of behaviour assumes that all visible actions are impatient — we will return to this point as we write examples later.

5 Strong bisimulation

We now wish to define notions of equivalence over these process expressions. We can follow Milner [22] and look at the standard ideas of bisimulation for the immediate and evolution transitions but it is not immediately clear how such a notion generalises to the probabilistic transitions.

We wish to make processes equivalent only if they define equal probability measures over sets of their descendants. We would like to say $P = Q$ iff $\mu(P, S) = \mu(Q, S)$ for all sets of processes S. But we do not wish to distinguish processes if their measures differ only on sets which separate equivalent descendants. Thus we do not wish to distinguish $\sum_{i \in I}[q_i]P_i$ from $\sum_{i \in I}[q_i]Q_i$ if we are considering P_i and Q_i equivalent. Thus we must disallow comparison of the measure for sets such as $S = \{P_i\}$ since $\mu(Q, S) = 0$ here whereas $\mu(P, S) = 1$. S would be a valid set for comparison if it also included all processes equivalent to any of the P_i. So we define the sub σ-field over which we require the measures to be equal:

Definition 12 *For an equivalence relation $\mathcal{R} \subset \mathsf{Proc} \times \mathsf{Proc}$ we define $\overline{B}(\mathcal{R}) \subset \overline{B}$ as*

$$\{S \in \overline{B} : S \text{ is a union of } \mathcal{R}\text{-equivalence classes } \}$$

In other words $\overline{B}(\mathcal{R})$ are all the measurable sets which don't separate $\mathcal{R}$-equivalent processes.

5.1 Definition

We proceed to define our fundamental equivalence following Milner. Our notion of equivalence is to be recursive; we would like to say two processes, P and Q, are equivalent if anything

P can do and become P' is matched by something Q can do and become Q', with P' and Q' equivalent. But this definition is circular; we actually want the greatest fix point of this definition. So we first give a condition over general equivalence relations which tests if they are "satisfactory" in the processes they relate.

Definition 13 *An equivalence relation $\mathcal{R} \subset \mathsf{Proc} \times \mathsf{Proc}$ is a* strong bisimulation *iff $P\ \mathcal{R}\ Q$ implies for all $\alpha \in \mathrm{Act}$, $t \in \mathbb{R}^+$, $S \in \overline{B}(\mathcal{R})$:*

1. *if $P \xrightarrow{\alpha} P'$, then $\exists Q'$ such that $Q \xrightarrow{\alpha} Q'$ and $P'\ \mathcal{R}\ Q'$.*
2. *if $Q \xrightarrow{\alpha} Q'$, then $\exists P'$ such that $P \xrightarrow{\alpha} P'$ and $P'\ \mathcal{R}\ Q'$.*
3. *if $P \overset{t}{\rightsquigarrow} P'$, then $\exists Q'$ such that $Q \overset{t}{\rightsquigarrow} Q'$ and $P'\ \mathcal{R}\ Q'$.*
4. *if $Q \overset{t}{\rightsquigarrow} Q'$, then $\exists P'$ such that $P \overset{t}{\rightsquigarrow} P'$ and $P'\ \mathcal{R}\ Q'$.*
5. *if* $\mathrm{Prob}(P)$, *then* $\mathrm{Prob}(Q)$ *and* $\mu(P, S) = \mu(Q, S)$.
6. *if* $\mathrm{Prob}(Q)$, *then* $\mathrm{Prob}(P)$ *and* $\mu(P, S) = \mu(Q, S)$.

Note that these last two conditions are equivalent to

5. $\mu(P, S) = \mu(Q, S)$.
6. $\mathrm{Prob}(P)$ iff $\mathrm{Prob}(Q)$

We have used the original form of the comparison to emphasise the bisimulation nature.

Many equivalence relations (including both the empty and identity relations) are strong bisimulations. But say P and Q are related by some strong bisimulation relation. Then they have the equivalence property we require since their descendants are still related by that same equivalence relation. So we define strong bisimulation between processes by:

Definition 14 *We say two processes P and Q are in strong bisimulation, denoted by $P \sim Q$, iff there is any strong bisimulation with $P\ \mathcal{R}\ Q$.*

Thus

$$\sim = \cup\{\mathcal{R} : \mathcal{R} \text{ is a strong bisimulation}\}$$

We regularly need to convert an arbitrary relation into an equivalence relation. Thus we write $\mathcal{R}^*$ to indicate the reflexive, symmetric, transitive closure of $\mathcal{R}$.

Proposition 6 *$\sim$ is the largest strong bisimulation*

Proof

It is plainly symmetric and reflexive. For transitivity, assume $P \sim Q \sim R$. Thus, for some strong bisimulations $\mathcal{R}$ and $\mathcal{S}$, we have $P\ \mathcal{R}\ Q\ \mathcal{S}\ R$. Now consider $(\mathcal{R} \cup \mathcal{S})^*$, the transitive closure of $\mathcal{R} \cup \mathcal{S}$. This is plainly an equivalence relation and so we check the other conditions. We prove by induction over length of transitive closure. Conditions 1–4 are straightforward. The last conditions are trivial once we observe that any set $S \in \overline{B}((\mathcal{R} \cup \mathcal{S})^*)$ is also in $\overline{B}(\mathcal{R})$ (and $\overline{B}(\mathcal{S})$) since $\mathcal{R}$ (and $\mathcal{S}$) are $\subseteq (\mathcal{R} \cup \mathcal{S})^*$. Thus $\sim$ is transitive.

Thus it is an equivalence relation.

Now assume $P \sim Q$. Then $P\ \mathcal{R}\ Q$ for some $\mathcal{R}$. Thus, again, conditions 1–4 are trivial. Also, as before, $S \in \overline{B}(\sim)$ implies $S \in \overline{B}(\mathcal{R})$ since $\mathcal{R} \subseteq \sim$. Thus $\mu(P, S) = \mu(Q, S)$ and so $\sim$ is a strong bisimulation. □

Since we only ever look at the value of time expressions to deduce transitions we know that $t = t'$ implies $P\{t/s\} \sim P\{t'/s\}$.

We have defined the equivalence we wish to investigate. But we first prove some lemmas to deal with common situations we will encounter. To claim two process are strongly bisimilar we must exhibit a complete strong bisimulation relating them. But we might like to supply a smaller relation which omits to relate other processes we already know to be bisimilar. To do this we define *strong bisimulation up to* $\sim$

Definition 15 *For a relation* $\mathcal{R} \subset \mathsf{Proc} \times \mathsf{Proc}$ *(not necessarily an equivalence) define* $\mathcal{S} = (\sim \cup \mathcal{R})^*$. *Then* $\mathcal{R}$ *is a strong bisimulation up to* $\sim$ *iff*

$P\ \mathcal{R}\ Q$ *implies for all* $\alpha \in \mathrm{Act}$, $t \in \mathbb{R}^+$, $S \in \overline{B}(\mathcal{S})$

1. *if* $P \xrightarrow{\alpha} P'$, *then* $\exists Q'$ *such that* $Q \xrightarrow{\alpha} Q'$ *and* $P'\ \mathcal{S}\ Q'$.
2. *if* $Q \xrightarrow{\alpha} Q'$, *then* $\exists P'$ *such that* $P \xrightarrow{\alpha} P'$ *and* $P'\ \mathcal{S}\ Q'$.
3. *if* $P \overset{t}{\rightsquigarrow} P'$, *then* $\exists Q'$ *such that* $Q \overset{t}{\rightsquigarrow} Q'$ *and* $P'\ \mathcal{S}\ Q'$.
4. *if* $Q \overset{t}{\rightsquigarrow} Q'$, *then* $\exists P'$ *such that* $P \overset{t}{\rightsquigarrow} P'$ *and* $P'\ \mathcal{S}\ Q'$.
5. $\mu(P, S) = \mu(Q, S)$.

Then we have

Lemma 9 *If* $\mathcal{R}$ *is a strong bisimulation up to* $\sim$ *then* $(\sim \cup \mathcal{R})^*$ *is a strong bisimulation and so* $\mathcal{R} \subseteq \sim$.

Proof

Define $\mathcal{S} = (\sim \cup \mathcal{R})^*$. This is plainly an equivalence relation. We show $\mathcal{S}$ a strong bisimulation by induction over the structure of $\mathcal{S}$. Assume $P\ \mathcal{S}\ Q$. Either

- $P \sim Q$. Trivial.
- $P\ \mathcal{R}\ Q$. Then our bisimulation conditions are straightforwardly true, since $\mathcal{R}$ is a bisimulation up to $\sim$.
- $P\ \mathcal{S}\ Q$ because $Q\ \mathcal{S}\ P$. Trivial.
- $P\ \mathcal{S}\ Q$ because $P \equiv Q$. Trivial.
- $P\ \mathcal{S}\ Q$ because $P\ \mathcal{S}\ R\ \mathcal{S}\ Q$.
  - If $P \xrightarrow{\alpha} P'$, then $\exists R'$ such that $R \xrightarrow{\alpha} R'$ and $P'\ \mathcal{S}\ R'$ since by induction we have the conditions for $P\ \mathcal{S}\ R$. Similarly if $R \xrightarrow{\alpha} R'$, then $\exists Q'$ such that $Q \xrightarrow{\alpha} Q'$ and $R'\ \mathcal{S}\ Q'$. Thus $P'\ \mathcal{S}\ Q'$ as required.
  - The next three conditions are similar.
  - If $S \in \overline{B}(\mathcal{S})$ then $\mu(P, S) = \mu(R, S) = \mu(Q, S)$ as required.

Thus $\mathcal{S}$ is a strong bisimulation. But $\mathcal{R} \subseteq \mathcal{S}$ so $\mathcal{R} \subseteq \sim$ as required. □

In the fifth condition (relating to probabilities) we must check sets S in $\overline{B}(\mathcal{S})$. But since both $\sim$ and $\mathcal{R} \subseteq \mathcal{S}$ it is clear that it is sufficient to establish the equality for all sets in $\overline{B}(\sim)$ or $\overline{B}(\mathcal{R}^*)$.

We also require a lemma to help us deal with the probabilistic transitions. Frequently the transitions of P and Q related by $\mathcal{R}$ are plainly "isomorphic" in some sense. We show the sense in which this makes their measures equal.

Lemma 10 *Assume* $g_1, g_2 : \mathsf{Proc} \to \mathsf{Proc}$ *and* $h : \mathsf{Index}_P \to \mathsf{Index}$ *have*

- $P \stackrel{p}{\hookrightarrow}_l P'$ *iff* $g_1(P) \stackrel{p}{\hookrightarrow}_{h(l)} g_2(P')$
- $\int_{l \in L} l = \int_{l \in L} h(l) \quad$ *for all* $L \in \overline{C}$

Then $\mu(P, S) = \mu(g_1(P), g_2(S))$. [2]

Proof

Plainly

$$q_P^{-1}(S) = h^{-1}(q_{g_1(P)}^{-1}(g_2(S))) \qquad p_P(l) = p_{g_1(P)}(h(l))$$

Thus

$$\begin{aligned}
\mu(P,S) &= \int_{l \in q_P^{-1}(S)} p_P(l) \\
&= \int_{l \in q_P^{-1}(S)} p_{g_1(P)}(h(l)) \\
&= \int_{l \in h^{-1}(q_{g_1(P)}^{-1}(g_2(S)))} p_{g_1(P)}(h(l)) \\
&= \int_{l \in q_{g_1(P)}^{-1}(g_2(S))} p_{g_1(P)}(l) \\
&= \mu(g_1(P), g_2(S))
\end{aligned}$$

as required. □

5.2 Algebraic properties

We have defined an equivalence over processes which preserves the transition behaviour of those processes. We now look at the algebraic behaviour of this equivalence by finding various sound axioms for $\sim$. We find that $+$ and $|$ are commutative and associative, with nil as a unit, and that $\backslash A$ and $[S]$ distribute through the other operators. The proofs are by exhibition of the appropriate bisimulations, [32].

Theorem 1

1. $P + \mathsf{nil} \sim P$
2. $P + Q \sim Q + P$
3. $P + (Q + R) \sim (P + Q) + R$
4. $P \mid \mathsf{nil} \sim P$
5. $P \mid Q \sim Q \mid P$

[2] The domain of g_2 is extended to sets of processes in the natural way, i.e. by a 'map'

6. $P \mid (Q \mid R) \sim (P \mid Q) \mid R$
7. $\mathsf{nil}\backslash A \sim \mathsf{nil}$
8. $(\alpha.P)\backslash A \sim \alpha.(P\backslash A) \quad$ *if* $\alpha \notin A$ *and* $\overline{\alpha} \notin A$
9. $(\alpha.P)\backslash A \sim \mathbf{0} \quad$ *if* $\alpha \in A$ *or* $\overline{\alpha} \in A$
10. $(\alpha[s \leftarrow t].P)\backslash A \sim \alpha[s \leftarrow t].(P\backslash A) \quad$ *if* $\alpha \notin A$ *and* $\overline{\alpha} \notin A$
11. $(\alpha[s \leftarrow t].P)\backslash A \sim \mathsf{nil} \quad$ *if* $\alpha \in A$ *or* $\overline{\alpha} \in A$
12. $(P + Q)\backslash A \sim (P\backslash A) + (Q\backslash A)$
13. $(t).P\backslash A \sim (t).(P\backslash A)$
14. $\mathcal{R}[s \leftarrow f].P\backslash A \sim \mathcal{R}[s \leftarrow f].(P\backslash A)$
15. $(\sum[p_i]P_i)\backslash A \sim \sum[p_i](P_i\backslash A)$
16. $\mathsf{nil}[S] \sim \mathsf{nil}$
17. $(\alpha.P)[S] \sim S(\alpha).(P[S])$
18. $(\alpha[s \leftarrow t].P)[S] \sim S(\alpha)[s \leftarrow t].(P[S])$
19. $(P + Q)[S] \sim P[S] + Q[S])$
20. $(t).P[S] \sim (t).(P[S])$
21. $\mathcal{R}[s \leftarrow f].P[S] \sim \mathcal{R}[s \leftarrow f].(P[S])$
22. $(\sum[p_i]P_i)[S] \sim \sum[p_i](P_i[S])$
23. $\mathsf{rec}(X = P) \sim P\{\mathsf{rec}(X = P)/X\}$

Since we are only interested in processes modulo bisimulation and for this equivalence we have associativity (and commutativity) for $+$ and $\mid$ we will eliminate brackets and abbreviate multiple sums by $\sum_{i\in I} P_i$ and multiple parallels by $\prod_{i\in I} P_i$ (for I finite). We will similarly drop the brackets from indices in Index.

We might have expected $P + P \sim P$ but this is not true for probabilistic P where the two P on the LHS can choose independently (without resolving the $+$) and become different. It is of course true for P which must act before a probabilistic transition.

The more interesting laws are the following

Theorem 2

1. $\alpha.P + \alpha.P \sim \alpha.P$
2. $\alpha[s \leftarrow t].P + \alpha[s \leftarrow t].P \sim \alpha[s \leftarrow t].P$
3. $(t).\mathsf{nil} \sim \mathsf{nil}$
4. $(0).P \sim P$
5. $(t).(t').P \sim (t + t').P$
6. $(t).(P + Q) \sim (t).P + (t).Q$
7. $(t).(P \mid Q) \sim (t).P \mid (t).Q$
8. $\tau[s \leftarrow t].P \sim \tau.P\{t/s\}$
9. $\alpha[s \leftarrow t].P \sim \alpha[s \leftarrow 0].P\{s + t/s\}$
10. $\tau[s \leftarrow 0].P + (t).Q \sim \tau[s \leftarrow 0].P \quad$ *if* $t > 0$

11. $\alpha.P + (t).Q \sim \alpha.P \quad$ *if* $t > 0$

12. $P + \sum_{i\in I}[q_i]P_i \sim \sum_{i\in I}[q_i]\,(P + P_i)$

13. $P \mid \sum_{i\in I}[q_i]P_i \sim \sum_{i\in I}[q_i]\,(P \mid P_i)$

14. $\sum_{i\in I}[p_i]P_i + \sum_{j\in J}[q_j]Q_j \sim \sum_{i\in I, j\in J}[p_i.q_j]\,(P_i + Q_j)$

15. $P + \mathcal{R}[s \leftarrow f].Q \sim \mathcal{R}[s \leftarrow f].(P + Q) \quad$ *if* s *not free in* P

16. $P \mid \mathcal{R}[s \leftarrow f].Q \sim \mathcal{R}[s \leftarrow f].(P \mid Q) \quad$ *if* s *not free in* P

17. $\alpha[s \leftarrow 0].P + (t).\,\alpha[s \leftarrow t].P \sim \alpha[s \leftarrow 0].P$

The most important result however is the expansion theorem. In CCS it is well known that $\alpha.\mathsf{nil} \mid \beta.\mathsf{nil} \sim \alpha.\beta.\mathsf{nil} + \beta.\alpha.\mathsf{nil}$. This is true in our language of course but we must also consider the other forms of prefixing. We do not have that

$$\alpha[].\mathsf{nil} \mid (2).\beta.\mathsf{nil} \sim \alpha[].(2).\beta.\mathsf{nil} + (2).\beta.\alpha[].\mathsf{nil}$$

The problem here is that the initial α may be delayed, say by one time unit. Then the delay between its execution and the subsequent β needs to be reduced by that one time unit. In fact we have

$$\alpha[].\mathsf{nil} \mid (2).\beta.\mathsf{nil} \sim \alpha[s \leftarrow 0].(2 \dot{-} s).\beta.\mathsf{nil} + (2).\beta.\alpha[].\mathsf{nil}$$

This simple equivalence demonstrates why we have extended CCS with time variables — we need them to record and then act upon the amount of time an idling action has delayed.

Theorem 3 *If* $P = \left(\sum_{i\in I}(t_i).\,\alpha_i[s \leftarrow 0].P_i\right)$, $Q = \left(\sum_{j\in J}(t'_j).\,\beta_j[s \leftarrow 0].Q_j\right)$ *then*

$$\begin{aligned} P \mid Q \quad \sim \quad & \sum_{i\in I}(t_i).\,\alpha_i[s \leftarrow 0].\left(P_i \mid \sum_{j\in J}(t'_j \dot{-} t_i \dot{-} s).\,\beta_j[s \leftarrow (t_i + s) \dot{-} t'_j].Q_j\right) \\ + \quad & \sum_{j\in J}(t'_j).\,\beta_j[s \leftarrow 0].\left(Q_j \mid \sum_{i\in I}(t_i \dot{-} t'_j \dot{-} s).\,\alpha_i[s \leftarrow (t'_j + s) \dot{-} t_i].P_i\right) \\ + \quad & \sum_{\alpha_k = \overline{\beta_l}}((t_k \dot{-} t'_l) + t'_l).\tau.\,(P_k\{t'_l \dot{-} t_k/s\} \mid Q_l\{t_k \dot{-} t'_l/s\}) \end{aligned}$$

If $P = \left(\sum_{i\in I}(t_i).\,\alpha_i[s \leftarrow 0].P_i\right)$, $Q = \left(\sum_{j\in J}(t'_j).\beta_j.Q_j\right)$ *then*

$$\begin{aligned} P \mid Q \quad \sim \quad & \sum_{i\in I}(t_i).\,\alpha_i[s \leftarrow 0].\left(P_i \mid \sum_{j\in J}(t'_j \dot{-} t_i \dot{-} s).\beta_j.Q_j\right) \\ + \quad & \sum_{j\in J}(t'_j).\beta_j.\left(Q_j \mid \sum_{i\in I}(t_i \dot{-} t'_j).\,\alpha_i[s \leftarrow (t'_j + s) \dot{-} t_i].P_i\right) \\ + \quad & \sum_{\alpha_k = \overline{\beta_l}}(t'_l).\tau.\,(P_k\{t'_l \dot{-} t_k/s\} \mid Q_l) \end{aligned}$$

If $P = \left(\sum_{i\in I}(t_i).\alpha_i.P_i\right)$, $Q = \left(\sum_{j\in J}(t'_j).\beta_j.Q_j\right)$ *then*

$$\begin{aligned} P \mid Q \quad \sim \quad & \sum_{i\in I}(t_i).\alpha_i.\left(P_i \mid \sum_{j\in J}(t'_j \dot{-} t_i).\beta_j.Q_j\right) \\ + \quad & \sum_{j\in J}(t'_j).\beta_j.\left(Q_j \mid \sum_{i\in I}(t_i \dot{-} t'_j).\alpha_i.P_i\right) \end{aligned}$$

$$+ \sum_{\alpha_k=\overline{\beta_l}, t_k=t'_l} (t_k).\tau.\,(P_k \mid Q_l)$$

Note that in these expansions, the impatient action term with the least leading delay will pre-empt any of the other terms with a larger initial (t).

Proof

We prove the first only since the other two are obvious from the first. Proof is by bisimulation. Define

$$\begin{aligned}
P(d) &= \sum_{i\in I}(t_i \dot{-} d).\,\alpha_i[s \leftarrow d \dot{-} t_i].P_i \\
Q(d) &= \sum_{j\in J}(t'_j \dot{-} d).\,\beta_j[s \leftarrow d \dot{-} t'_j].Q_j \\
R_P(d) &= \sum_{i\in I}(t_i \dot{-} d).\,\alpha_i[s \leftarrow d \dot{-} t_i].\,(P_i \mid Q(t_i + s)) \\
R_Q(d) &= \sum_{j\in J}(t'_j \dot{-} d).\,\beta_j[s \leftarrow 0].\,\big(Q_i \mid P(t'_j + s)\big) \\
R_\tau(d) &= \sum_{\alpha_k=\overline{\beta_l}} (\max(t_k, t'_l) \dot{-} d).\tau.\,(P_k\{t'_l \dot{-} t_k/s\} \mid Q_l\{t_k \dot{-} t'_l/s\})
\end{aligned}$$

Then the expansion theorem reads that

$$P(0) \mid Q(0) \sim R_P(0) + R_Q(0) + R_\tau(0)$$

We prove it by defining

$$\mathcal{S} = \{(P(d) \mid Q(d), R_P(d) + R_Q(d) + R_\tau(d)) : d \in [0, t_\tau]\}$$

where t_τ is the first time at which the process may perform a τ i.e.

$$t_\tau = \min\{\{t_i : \alpha_i = \tau\} \cup \{t'_j : \beta_j = \tau\} \cup \{\max(t_k, t'_l) : \alpha_k = \overline{\beta_l}\}\}$$

or ∞ if it may not perform one (i.e. if this set is empty). Then $\mathcal{S}$ is a strong bisimulation up to $\sim$, which we prove by establishing each of the conditions of definition 15 in turn:

1. If $P(d) \mid Q(d) \xrightarrow{\alpha_i} P'$ because term i in $P(d)$ acts, then we have P' is $P_i\{d \dot{-} t_i/s\} \mid Q(d)$ and $t_i \le d$. But the RHS can $\xrightarrow{\alpha_i}$ to

   $$P_i\{d \dot{-} t_i/s\} \mid Q(t_i + (d \dot{-} t_i))$$

   and since $t_i \le d$ then $t_i + (d \dot{-} t_i) = d$ as required. If the LHS performs β_j the result is similar. Finally if it performs τ because $\alpha_i = \overline{\beta_j}$ then we know $d = \max(t_i, t'_j)$ and the LHS becomes

   $$P_i\{d \dot{-} t_i/s\} \mid Q_j\{d \dot{-} t'_j/s\}$$

   which is

   $$P_i\{t'_j \dot{-} t_i/s\} \mid Q_j\{t_i \dot{-} t'_j/s\}$$

   while the RHS can perform τ and become

   $$P_i\{t'_j \dot{-} t_i/s\} \mid Q_j\{t_i \dot{-} t'_j/s\}$$

   which is the same as required.

2. Inspection of the RHS shows we have accounted for every way it can act.

3. Assume the LHS $\overset{t}{\rightsquigarrow}$. Then it becomes

$$\begin{aligned} & \sum_{i\in I}(t_i \dot{-} (d+t)).\, \alpha_i[s \leftarrow (d+t) \dot{-} t_i].P_i \\ | \quad & \sum_{j\in J}(t'_j \dot{-} (d+t)).\, \beta_j[s \leftarrow (d+t) \dot{-} t'_j].Q_j \end{aligned}$$

which is $P(d+t) \mid Q(d+t)$ while the RHS becomes

$$\begin{aligned} & \sum_{i\in I}(t_i \dot{-} (d+t)).\, \alpha_i[s \leftarrow (d+t) \dot{-} t_i].\, (P_i \mid Q\,(t_i+s)) \\ + \quad & \sum_{j\in J}(t'_j \dot{-} (d+t)).\, \beta_j[s \leftarrow (d+t) \dot{-} t'_j].\, \big(Q_i \mid P(t'_j+s)\big) \\ + \quad & \sum_{\alpha_k=\overline{\beta_l}} (\max(t_k, t'_l) \dot{-} d \dot{-} t).\tau.\, (P_k\{t'_l \dot{-} t_k/s\} \mid Q_l\{t_k \dot{-} t'_l/s\}) \end{aligned}$$

which is $R_P(d+t) + R_Q(d+t) + R_\tau(d+t)$ as required. Further the maximum time they can evolve (before they do a τ) is the same.

4. Already shown in 3.

5. Neither can perform a probabilistic transition and so the result is trivial.

□

We extend bisimulation to process expressions in the usual way.

Definition 16 *For process expressions with free variables included in the process variables $\tilde{X}$ and the time variables $\tilde{s}$, $P \sim Q$ iff $P\{\tilde{R}/\tilde{X}\}\{\tilde{t}/\tilde{s}\} \sim Q\{\tilde{R}/\tilde{X}\}\{\tilde{t}/\tilde{s}\}$ for all (closed) processes $\tilde{R}$ and all closed time expressions $\tilde{t}$.*

We now show that this $\sim$ relation is a congruence i.e. that it is preserved by all the operations of our algebra. Thus for example we wish to know that $P \sim Q$ implies $\alpha.P \sim \alpha.Q$. Knowing this for each operation from our algebra implies that

$$P \sim Q \quad \text{implies} \quad C[P] \sim C[Q] \text{ for any context } C[-]$$

i.e. that $\sim$ is preserved by substitution into any context. This is essential because we wish to use these algebraic laws to reason compositionally about complex processes. That $\sim$ is a congruence allows us to use substitution to simplify and work with expressions. The required result, proved in [32], is:

Proposition 7 *For all $P, Q \in \mathsf{PExp}$, $t \in \mathsf{TExp}$, $P \sim Q$ implies*

- $\alpha.P \sim \alpha.Q$
- $(t).P \sim (t).Q$
- $\mathcal{R}[s \leftarrow f].P \sim \mathcal{R}[s \leftarrow f].Q$
- $P + R \sim Q + R$
- *If* $\forall i \in I\ P_i \sim Q_i$ *then* $\sum_{i\in I}[q_i]P_i \sim \sum_{i\in I}[q_i]Q_i$
- $P \mid R \sim Q \mid R$
- $P[S] \sim Q[S]$
- $P\backslash A \sim Q\backslash A$
- $\mathsf{rec}(X = P) \sim \mathsf{rec}(X = Q)$

5.3 Remarks

The nature of our definition of bisimulation requires evolution time sequences to match exactly in bisimilar processes. Since we have no notion of convergence to a limit by a sequence of processes, this effectively implies that time is discrete. Similarly, probabilistic choice is finite and so is also defined on a discrete space. However, the time delays are random variables which are *arbitrary* real numbers. Hence the discrete set of time delays in any SPADES simulation can also be arbitrary and so we only lack the notion of limits *per se*, not the availability of real time values and axioms relating processes containing them.

We have derived a significant set of axioms that facilitates reasoning about processes defined in our algebra. However, the prospects for a complete axiomatisation would seem remote. This is because of the possibility of evolution by a process. Evolution is determined by time random variables which have previously been sampled in a process. Thus, at any time, two processes can only be deemed bisimilar if information about their history (i.e. sampled values) is known and matches. This would lead (at best) to a highly complex axiomatisation with conditional axioms, which would likely be intractable. Note, however, that such problems do not arise with finite probabilistic choice (any more than with non-deterministic choice) and so this feature would not preclude a complete axiomatisation, as in standard process algebras like CCS, for example.

However, in circumstances where all delays are random and known to be memoryless, e.g. have negative exponential distribution, this issue can be resolved. Sample actions and recording of the passage of time become redundant in such an algebra where delay actions need now only be decorated directly (with a rate). In such a situation the SPADES process algebra becomes a Markovian one with the extension of immediate actions [3]. It has been shown that a sound and complete axiomatisation exists for similar systems ([13]).

6 Weak bisimulation

As for standard CCS we wish to abstract from the τ action by moving to a weak version of bisimulation. We have already said that we wish to consider the τ actions as representing the invisible internal behaviour of a process. Thus we would certainly like $\alpha.\tau.\mathsf{nil}$ to be considered equal to $\alpha.\mathsf{nil}$ since their only observable behaviour is to perform α and terminate. The former has an internal state change before it terminates but we wish to consider that invisible.

We start by defining a new action relation ignoring τ actions.

Definition 17 *For $a \in$ Act we write*

$$P \stackrel{a}{\Rightarrow} Q \quad \textit{iff} \quad \exists m, n \geq 0 . P \stackrel{\tau^m a \tau^n}{\rightarrow} Q$$

$$P \stackrel{\varepsilon}{\Rightarrow} Q \quad \textit{iff} \quad \exists n \geq 0 . P \stackrel{\tau^n}{\rightarrow} Q$$

Here ε represents an internal invisible transition. Note that $P \stackrel{\varepsilon}{\Rightarrow} P$. The definition means that $P \stackrel{\tau}{\Rightarrow}$ implies that P can perform at least one τ.

We also wish to make arbitrary τ actions in the midst of a long evolution transition invisible. So we define a new evolution relation.

Definition 18

$$P \stackrel{t}{\Rightarrow\!\!\!\!\!\sim} Q \quad \textit{iff}$$

$$\exists n > 0, \textit{ and times } t_i > 0 \quad \textit{with } P \stackrel{\varepsilon}{\Rightarrow} \stackrel{t_1}{\sim} \stackrel{\varepsilon}{\Rightarrow} \ldots \stackrel{t_n}{\sim} \stackrel{\varepsilon}{\Rightarrow} Q$$

$$\textit{and } t = t_0 + t_1 + \cdots + t_n$$

It is not clear that we can generalize probabilistic transitions by allowing τ transitions in the middle. It certainly seems difficult to interpret non-deterministic choices within probabilistic

[3] The probabilistic choice extension is clearly not problematic

transitions. For example consider $\mathcal{R}[s \leftarrow f].\,(\tau.\,\mathcal{R}[s' \leftarrow f_1].P(s') + \tau.\,\mathcal{R}[s' \leftarrow f_2].P(s'))$. It is not clear that we can define a meaningful probability for this process to reach $\{P(s') : s' \in [0,1]\}$ say since we do not know whether to choose f_1 or f_2. So we make no changes to the probabilistic relation but allow an initial $\stackrel{\varepsilon}{\Rightarrow}$.

We define our fundamental equivalence as before using these modified relations:

Definition 19 *An equivalence relation $\mathcal{R} \subset \mathsf{Proc} \times \mathsf{Proc}$ is a* weak bisimulation *iff*
$P\ \mathcal{R}\ Q$ implies for all $\delta \in \Lambda \cup \{\varepsilon\}$, $t \in \mathbb{R}^+$, $S \in \overline{B}(\mathcal{R})$:

1. *if $P \stackrel{\delta}{\Rightarrow} P'$, then $\exists Q'$ such that $Q \stackrel{\delta}{\Rightarrow} Q'$ and $P'\ \mathcal{R}\ Q'$.*
2. *if $Q \stackrel{\delta}{\Rightarrow} Q'$, then $\exists P'$ such that $P \stackrel{\delta}{\Rightarrow} P'$ and $P'\ \mathcal{R}\ Q'$.*
3. *if $P \stackrel{t}{\rightrightarrows} P'$, then $\exists Q'$ such that $Q \stackrel{t}{\rightrightarrows} Q'$ and $P'\ \mathcal{R}\ Q'$.*
4. *if $Q \stackrel{t}{\rightrightarrows} Q'$, then $\exists P'$ such that $P \stackrel{t}{\rightrightarrows} P'$ and $P'\ \mathcal{R}\ Q'$*
5. *if $\mathrm{Prob}(P)$ then $\exists Q'$ such that $Q \stackrel{\varepsilon}{\Rightarrow} Q'$ with $\mathrm{Prob}(Q')$ and $\mu(P,S) = \mu(Q',S)$*
6. *if $\mathrm{Prob}(Q)$ then $\exists P'$ such that $P \stackrel{\varepsilon}{\Rightarrow} P'$ with $\mathrm{Prob}(P')$ and $\mu(P',S) = \mu(Q,S)$.*

$$\approx\ = \cup\{\mathcal{R} : \mathcal{R} \text{ is a weak bisimulation}\}$$

Proposition 8 *$\approx$ is the largest weak bisimulation*

Proof

The proof is the same as the one for strong bisimulation. □

To simplify our proofs we also define *weak bisimulation up to $\approx$* as for $\sim$ in the previous section.

Definition 20 *For a relation $\mathcal{R} \subset \mathsf{Proc} \times \mathsf{Proc}$ (not necessarily an equivalence) define $\mathcal{S} = (\approx \cup\, \mathcal{R})^*$. Then $\mathcal{R}$ is a* weak bisimulation up to $\approx$ *iff*
$P\ \mathcal{R}\ Q$ implies for all $\delta \in \Lambda \cup \{\varepsilon\}$, $t \in \mathbb{R}^+$, $S \in \overline{B}(\mathcal{S})$:

1. *if $P \stackrel{\delta}{\Rightarrow} P'$, then $\exists Q'$ such that $Q \stackrel{\delta}{\Rightarrow} Q'$ and $P'\ \mathcal{S}\ Q'$.*
2. *if $Q \stackrel{\delta}{\Rightarrow} Q'$, then $\exists P'$ such that $P \stackrel{\delta}{\Rightarrow} P'$ and $P'\ \mathcal{S}\ Q'$.*
3. *if $P \stackrel{t}{\rightrightarrows} P'$, then $\exists Q'$ such that $Q \stackrel{t}{\rightrightarrows} Q'$ and $P'\ \mathcal{S}\ Q'$.*
4. *if $Q \stackrel{t}{\rightrightarrows} Q'$, then $\exists P'$ such that $P \stackrel{t}{\rightrightarrows} P'$ and $P'\ \mathcal{S}\ Q'$.*
5. *if $\mathrm{Prob}(P)$ then $\exists Q'$ such that $Q \stackrel{\varepsilon}{\Rightarrow} Q'$ with $\mathrm{Prob}(Q')$ and $\mu(P,S) = \mu(Q',S)$*
6. *if $\mathrm{Prob}(Q)$ then $\exists P'$ such that $P \stackrel{\varepsilon}{\Rightarrow} P'$ with $\mathrm{Prob}(P')$ and $\mu(P',S) = \mu(Q,S)$.*

Then we have straightforwardly that

Lemma 11 *If $\mathcal{R}$ is a weak bisimulation up to $\approx$ then $(\approx \cup\, \mathcal{R})^*$ is a weak bisimulation and so $\mathcal{R} \subseteq \approx$.*

Thus to show two processes are $\approx$ we need only show that a relation including them is a weak bisimulation up to $\approx$.

We now show that $\sim$ is a refinement of $\approx$. We would certainly expect this since we wish to consider $\sim$ processes as identical and hence not distinguishable by any other meaningful equivalence.

Lemma 12

$$\sim \subseteq \approx$$

Proof

We show that $\sim$ is a weak bisimulation. If $P \sim Q$ then by induction on the number of strong transitions in e.g. $P \stackrel{\delta}{\Rightarrow} P'$ we can match each one with one in $Q \stackrel{\delta}{\Rightarrow} Q'$ preserving strong bisimulation at each step. The fifth and sixth conditions are plain since if P is probabilistic then Q has no τ transitions and *vice versa* while if both are probabilistic the conditions are the same trivially. □

Thus all the algebraic laws we established for $\sim$ are sound for $\approx$. We also have the important new law that $\tau.P \approx P$. Unfortunately this law makes $\approx$ fail to be a congruence. In particular the initial τ on the LHS can resolve a + context in a way which the RHS cannot match. Thus we have for example

$$\tau.\alpha.\mathsf{nil} \approx \alpha.\mathsf{nil} \quad \text{but} \quad \tau.\alpha.\mathsf{nil} + \beta.\mathsf{nil} \not\approx \alpha.\mathsf{nil} + \beta.\mathsf{nil}$$

since in the latter equation the left hand side may $\stackrel{\varepsilon}{\Rightarrow} \alpha.\mathsf{nil}$ and thus become (silently) unable to perform a β while the right hand side may not move internally (i.e. via τ actions) to any bisimilar term and thus will always be able to perform a β. This is the usual example of the problem caused by initial τ actions but we also have an interesting new problem. We know

$$\tau.\,\mathcal{R}[s \leftarrow f].R(s) \approx \mathcal{R}[s \leftarrow f].R(s)$$

but if we write

$$P \triangleq [\tfrac{1}{2}]\tau.\,\mathcal{R}[s \leftarrow f].R(s) \dot{+} [\tfrac{1}{2}]\,\mathcal{R}[s \leftarrow f].R(s)$$

$$Q \triangleq [\tfrac{1}{2}]\,\mathcal{R}[s \leftarrow f].R(s) \dot{+} [\tfrac{1}{2}]\,\mathcal{R}[s \leftarrow f].R(s)$$

we have $P \not\approx Q$ since writing $S = \{R(t) : t \in \mathbb{R}^+\}$ we have $\mu(P, S) = \frac{1}{2}$ while $\mu(Q, S) = 1$.

This means we cannot use $\approx$ for algebraic reasoning. We must move to a congruence. Following Milner we would like to define the largest congruence smaller than $\approx$ as our notion of equality i.e. the congruence which equates as many weakly bisimilar processes as possible.

Definition 21 *$P = Q$ iff $\forall$ contexts $C[-]$ $\quad C[P] \approx C[Q]$.*

This says that two processes are $=$ if no context can show they are not weakly bisimilar. We have chosen weak bisimulation as saying two processes are observation equivalent. But we allow our observers to check this weak bisimulation in any context and thus distinguish processes such as $\tau.P$ from P.

It is difficult to immediately prove properties of this $=$ relation and so we start with an alternative three stage characterization of equality. We define $P =_1 Q$ if any single action makes them weakly bisimilar, $P =_2 Q$ if any evolution makes them $=_1$, and $P =_3 Q$ if their probabilistic transitions respect $=_2$. Thus formally:

Definition 22 *$P =_1 Q$ iff*

1. *$P \approx Q$*
2. *for all $\alpha \in$ Act, if $P \stackrel{\alpha}{\rightarrow} P'$, then $\exists Q'$ such that $Q \stackrel{\alpha}{\Rightarrow} Q'$ and $P' \approx Q'$.*
3. *for all $\alpha \in$ Act, if $Q \stackrel{\alpha}{\rightarrow} Q'$, then $\exists P'$ such that $P \stackrel{\alpha}{\Rightarrow} P'$ and $P' \approx Q'$.*

$P =_2 Q$ iff

1. *$P =_1 Q$*

2. *for all* $t \in \mathbb{R}^+$, *if* $P \overset{t}{\rightsquigarrow} P'$, *then* $\exists Q'$ *such that* $Q \overset{t}{\Rightarrow} Q'$ *and* $P' =_1 Q'$.

3. *for all* $t \in \mathbb{R}^+$, *if* $Q \overset{t}{\rightsquigarrow} Q'$, *then* $\exists P'$ *such that* $P \overset{t}{\Rightarrow} P'$ *and* $P' =_1 Q'$.

$P =_3 Q$ *iff*

1. $P =_2 Q$

2. *if* $\mathrm{Prob}(P)$ *then* $\exists Q'$ *such that* $Q \overset{\varepsilon}{\Rightarrow} Q'$ *with* $\mathrm{Prob}(Q')$ *and for all* $S \in \overline{B}(=_2)$, $\mu(P, S) = \mu(Q', S)$.

3. *if* $\mathrm{Prob}(Q)$ *then* $\exists P'$ *such that* $P \overset{\varepsilon}{\Rightarrow} P'$ *with* $\mathrm{Prob}(P')$ *and for all* $S \in \overline{B}(=_2)$, $\mu(P', S) = \mu(Q, S)$.

Note that we check $\approx$ only in part 1 after a labelled transition (i.e. not an evolution or probabilistic choice). This is because, just as we do not wish to claim $\tau.P = P$, we must also disallow $(1).\tau.P = (1).P$ and $\mathcal{R}[s \leftarrow f].\tau.P = \mathcal{R}[s \leftarrow f].P$.

Plainly $P =_3 Q$ implies $P \approx Q$ and it is now straightforward but tedious to show that $=_3$ is the congruence associated with $\approx$, the relation we have chosen as $=$, [32].

Next we move to an intermediate characterization of equality. We have seen that $\approx$ is not preserved by $+$. We show that we need only look at $+$ contexts to make $\approx$ the same as $=_3$.

Lemma 13 $P =_3 Q$ *iff* $\forall R\ P + R \approx Q + R$

Proof

For the forward direction, we define

$$\mathcal{S} = \{(P + R, Q + R) : \forall P, Q, R \text{ with } P =_3 Q\}$$

and show that this is a weak bisimulation up to $\approx$. For the converse, we assume that $P \neq_3 Q$ and show that $P + R \not\approx Q + R$. □

We now have a rigorous notion of equality:

Lemma 14 $P =_3 Q$ *iff* $P = Q$.

Proof

- Forward. Assume $P =_3 Q$.
  Since $=_3$ is a congruence, $C[P] =_3 C[Q]$ for all contexts $C[-]$. Thus $C[P] \approx C[Q]$ and we have $P = Q$.
- Backward. Plainly $P = Q$ implies $\forall R\ P + R \approx Q + R$ and so $P =_3 Q$.

□

We are now in the position of standard CCS with $\sim \subset = \subset \approx$ where $\sim$ and $=$ are congruences. In particular, we have the following rules analogous to those of CCS.

Theorem 4

1. $\alpha.(t).\tau.P = \alpha.(t).P$

2. $\tau.P + P = \tau.P \quad$ *if* $\neg\, \mathrm{Prob}(P)$

3. $\alpha.(P + \tau.Q) = \alpha.(P + \tau.Q) + \alpha.Q \quad$ *if* $\neg \mathrm{Prob}(P)$

Proof

In each case we use the definition of $=_3$ directly.

- They are clearly $\approx$ and then they are $=_1$ since both sides can only $\xrightarrow{\alpha}$ and $(t).\tau.P \approx (t).P$. Since they have the same evolution and probabilistic transitions, they are $=_3$.
- Plainly $\tau.P + P \approx \tau.P$ If $\tau.P + P \xrightarrow{\tau} P$ then $\tau.P \xrightarrow{\tau} P$. Alternatively if $\tau.P + P \xrightarrow{\alpha} P'$ because $P \xrightarrow{\alpha} P'$ then $\tau.P \stackrel{\alpha}{\Rightarrow} P'$ as required.

  If $\tau.P \xrightarrow{\alpha} P'$ then $\alpha = \tau$ and $P' = P$ and we have $\tau.P + P \xrightarrow{\tau} P$ as required. Thus $\tau.P + P =_1 \tau.P$

  Neither side can evolve or make a probabilistic transition and so they are $=_3$ as required.
- $\alpha.(P + \tau.Q) \approx \alpha.(P + \tau.Q) + \alpha.Q$ since if the RHS $\xrightarrow{\alpha} Q$ the LHS can $\xrightarrow{\alpha\tau} Q$. The same consideration shows that they are $=_1$.

  If $\alpha \neq \tau$ then both sides can idle and since neither can make a probabilistic transition they are $=_3$ as required.

□

We also have the rules from elementary probability theory for amalgamating nested probabilistic sums. For example

$$[\tfrac{1}{2}]P \mathbin{\dot{+}} [\tfrac{1}{2}]\left([\tfrac{1}{2}]Q \mathbin{\dot{+}} [\tfrac{1}{2}]R\right) = [\tfrac{1}{2}]P \mathbin{\dot{+}} [\tfrac{1}{4}]Q \mathbin{\dot{+}} [\tfrac{1}{4}]R$$

Similarly we can combine probabilistic sums with a final $\mathcal{R}[s \leftarrow f].P$ for example

$$\sum [q_i]\, \mathcal{R}[s \leftarrow f_i].P = \mathcal{R}[s \leftarrow f].P$$

for the appropriate density function f. Taking these together with the axioms listed in theorems 1,2,3,4 we have moved from operational semantics and a formal semantic definition of equality to a set of purely syntactic algebraic equations. These include the expansion theorem and (as shown in section 6) enable us to move from simulations written in a specificational style to other equivalent simulations.

7 Examples

7.1 Generalised Semi-Markov Processes (GSMP)

The GSMP as discussed in Whitt [34] is one of the most general yet tractable stochastic processes for modelling discrete event simulations (see e.g. the detailed exposition in Shedler [31] or Iglehart and Shedler [15]). We show we can represent systems modelled by a GSMP in our formalism.

Heuristically a GSMP has some countable set of states A through which it moves at times determined by a finite set of clocks C. Each state has associated with it some subset of the clocks, $E(a)$, specified by the function $E : A \to 2^C$. These are the clocks which will race to trigger a transition from that state. Once a state has been entered the clocks run down synchronously until one, say $c \in C$, has reached zero. Then a state transition occurs. The new state is chosen randomly by the probability function $p : A \times C \to (A \to [0,1])$ which assigns a probability to each possible choice of next state based on the current state and whichever clock ran down first. The new state has a new set of associated clocks. Where these include clocks already set for the previous state their settings remain unchanged. Where the clocks were not associated to the previous state they need to be set. This is done by choosing a time with a probability density function depending on both the previous and current states and events $f : A \times C \times A \times C \to (\mathbb{R}^+ \to [0,1])$ (we write $f(a, c, a', c')$).

Finally we need to specify an initial distribution $q : A \to [0,1]$ to choose our initial state and an initial clock setting probability $f_0 : A \times C \to (\mathbb{R}^+ \to [0,1])$ (i.e. for each state $a \in A$ and clock $c \in E(a)$ we have a distribution $f_0(a,c)$).

We now build a process which performs these steps and hence models the behaviour of the GSMP. We call the process $GSMP$. Here $\prod$ is an abbreviation for finite indexed parallelism while $\bigodot$ abbreviates finite indexed multiple prefixing (where the order does not matter).

Our representation of the GSMP selects an initial state via the distribution q.

$$GSMP \triangleq \sum_{a \in A} [q(a)] St0_a$$

In parallel with this state it starts all clocks associated with the events in $E(a)$.

$$St0_a \triangleq St_a \mid \prod_{c \in E(a)} Clock0_{ac}$$

These initial clocks have a special distribution f_0. Here the to_c action says clock c has timed out, while the can_c action is used to cancel the clock.

$$Clock0_{ac} \triangleq \mathcal{R}[s \leftarrow f_0(a,c)].(s).to_c + \overline{can_c}[]$$

Each state waits for any clock from $E(a)$ to time out and cause its state to change:

$$St_a \triangleq \sum_{c \in E(a)} \overline{to_c}.StCl_{ac}$$

Note that this behaviour is potentially non-deterministic. However a condition on the distributions of our clock settings can ensure that the probability of two clocks timing out at the same instant and hence a non-deterministic choice having to be made is zero.

The state having received a time out chooses its succeeding state a' via,

$$StCl_{ac} \triangleq \sum_{a' \in A} [p(a,c)(a')] StClSt_{aca'}$$

then it cancels any outstanding and no longer required clocks.

$$StClSt_{aca'} \triangleq (\bigodot_{c' \in E(a) \setminus \{c\} \setminus E(a')} can_{c'}).StClSt1_{aca'}$$

Finally it starts the new state and any new clocks required.

$$StClSt1_{aca'} \triangleq St_{a'} \mid \prod_{c' \in E(a') \setminus (E(a) \setminus \{c\})} Clock_{aca'c'}$$

These new clocks use their standard distribution.

$$Clock_{aca'c'} \triangleq \mathcal{R}[s \leftarrow f(a,c,a',c')].(s).to_{c'} + \overline{can_{c'}}$$

Note that strictly speaking this approach only works if we have a finite number of states since we have defined our language so that all processes must be represented by finite expressions. This does not mean that they are finite-state since the size of the expression is not bounded. It does mean that we must be able to give a finite representation of an infinite state system. In a subsequent section we demonstrate this with a finite representation of an infinite queue. The difference here is that standard approaches to stochastic processes do not address issues of computability. It is possible to describe a GSMP where the relationship between states and transitions is not computable. Our language is implementable and hence cannot represent such processes. Of course they cannot be simulated either.

7.2 Two processor queue

We return to the simple example presented in the informal description of our language. We consider a situation where a sequence of tasks arrive randomly, forming a workload. Then some (we choose two as an example) processors service the queue, first come first served.

First we define the Workload. The Workload waits a random time and then restarts itself along with a new Task.

$$Workload \triangleq \mathcal{R}[s \leftarrow f_1].(s).\,(Workload \mid Task)$$

The Task requests a processor, executes for a random time, and then releases the processor and terminates.

$$Task \triangleq \overline{procreq}[].\,\mathcal{R}[s \leftarrow f_2].(s).procrel$$

The Processor accepts a request, accepts a release and restarts.

$$Proc_1 \triangleq procreq[].\overline{procrel}[].Proc_1$$

Then our initial presentation of the system will be

$$System_1 \triangleq (Workload \mid Proc_1 \mid Proc_1) \setminus \{procreq, procrel\}$$

However if we examine the behaviour of $System_1$ we find it can do nothing (except $\overset{t}{\Rightarrow}$) and hence it is $=$ to nil. This is because we have deliberately made all its internal behaviour invisible. We need to decide what part of that behaviour we wish to monitor. So we add probe actions to the processor which amounts to a decision to evaluate processor utilisation. If our probe actions distinguish the two uses of $Proc$ then our resulting process will be non-deterministic since we have not specified which of the two instances of $Proc$ will be acquired by a *procreq* action. Instead we say the processes are indistinguishable and write them:

$$Proc = procreq[].probe_1.\overline{procrel}[].probe_2.Proc$$

Here $probe_1$ indicates to the observer the acquisition of a processor while $probe_2$ indicates its release. Plainly we could write a system with N processors with no difficulty. So our system is

$$System \triangleq (Workload \mid Proc \mid Proc) \setminus \{procreq, procrel\}$$

Given our two processor system we find we can rewrite $System$, using the expansion theorem repeatedly, to derive:

$$System = \mathcal{R}[s \leftarrow f_1].Pr0(s)$$

where

$$\begin{aligned}
Pr0(s) &\triangleq (s).\,\mathcal{R}[s' \leftarrow f_1].\,\mathcal{R}[s'' \leftarrow f_2].probe_1.Pr1(s', s'') \\
Pr1(s, s') &\triangleq (s).\tau.\,\mathcal{R}[s'' \leftarrow f_1].\,\mathcal{R}[s''' \leftarrow f_2].probe_1.Pr2(s'', s' - s, s''') \\
&\quad +(s').probe_2.Pr0(s - s') \\
Pr2(s, s', s'') &\triangleq (s).\tau.\,\mathcal{R}[s''' \leftarrow f_1].Prq(s''', s' - s, s'' - s, 1) \\
&\quad +(s').probe_2.Pr1(s - s', s'' - s') \\
&\quad +(s'').probe_2.Pr1(s - s'', s' - s'') \\
Prq(s, s', s'', 1) &\triangleq (s).\tau.\,\mathcal{R}[s''' \leftarrow f_1].Prq(s''', s' - s, s'' - s, 2) \\
&\quad +(s').probe_2.\,\mathcal{R}[s''' \leftarrow f_2].probe_1.Pr2(s - s', s'' - s', s''') \\
&\quad +(s'').probe_2.\,\mathcal{R}[s''' \leftarrow f_2].probe_1.Pr2(s - s'', s' - s'', s''') \\
Prq(s, s', s'', n) &\triangleq (s).\tau.\,\mathcal{R}[s''' \leftarrow f_1].Prq(s''', s' - s, s'' - s, n + 1) \\
&\quad +(s').probe_2.\,\mathcal{R}[s''' \leftarrow f_2].probe_1.Prq(s - s', s'' - s', s''', n - 1) \\
&\quad +(s'').probe_2.\,\mathcal{R}[s''' \leftarrow f_2].probe_1.Prq(s - s'', s' - s'', s''', n - 1)
\end{aligned}$$

for $n > 1$.

The relationship between this version and a Generalized Semi-Markov Process representation should be clear (see section 7.1). Each named process represents a state and its parameters represent clocks. At each state transition either a clock is passed on after having run down or is reset by a random choice. Thus we have taken a high-level description of a simulation and rewritten it using formal rules into a low-level GSMP style description.

7.3 Random Walk and its Equivalence

We have claimed that our semantics allows us to assess when two simulations are equal. This is certainly true for obvious syntactic manipulations of our processes. But what happens when we compare a more complex example. We consider a random walk on the integers which is generally considered "equal" to a single server queue (where distributions etc. are chosen appropriately). Surprisingly we will show that this notion of equivalence is not that represented by our weak bisimulation equivalence. This is essentially because the two examples make their random choices at different times.

We will assume all our distributions are negative exponential and represent them by their rates r_1, r_2, r_3. We first describe the equations for a simple random walk.

$$\begin{array}{rcll} W_0 & \triangleq & \mathcal{R}[s \leftarrow r_1].\epsilon_0.(s).\,([p]W_0 \dot{+} [q]W_1) & \\ W_n & \triangleq & \mathcal{R}[s \leftarrow r_1].\epsilon_n.(s).\,([p]W_{n-1} \dot{+} [q]W_{n+1}) & n > 0 \end{array}$$

The ϵ_n actions here are probes showing us by an impatient action when we reach a state in which the system is just starting a wait at position n.

Then we present our model of a single server, single class queue:

$$\begin{array}{rcll} B_0 & \triangleq & \epsilon_0.\overline{\alpha}[].B_1 & \\ B_n & \triangleq & \epsilon_n.\,(\overline{\alpha}[].B_{n+1} + \gamma[].B_{n+1}) & n > 0 \\ S & \triangleq & \mathcal{R}[s \leftarrow r_2].(s).\overline{\gamma}[].S & \\ P & \triangleq & \mathcal{R}[s \leftarrow r_3].(s).\,(\alpha[] \mid P) & \end{array}$$

Here α represents an arrival while γ represents a departure or service. The ϵ_n actions this time show when the queue has just obtained n items. We then define

$$Q_n \triangleq (B_n \mid S \mid P) \setminus \{\alpha, \beta, \gamma\}$$

and would like to show $W_n = Q_n$.

$$\begin{array}{l} Q_n = (B_n \mid S \mid P) \setminus \{\alpha, \beta, \gamma\} \\ \quad = (\epsilon_n.\overline{\alpha}[].B_{n+1} + \gamma[].B_{n-1} \mid \mathcal{R}[s \leftarrow r_2].(s).\overline{\gamma}[].S \mid \mathcal{R}[s \leftarrow r_3].(s).\,(\alpha[] \mid P)) \setminus \{\alpha, \beta, \gamma\} \end{array}$$

by the definitions of B_n, S and P.

$$= \mathcal{R}[s \leftarrow r_2].\,\mathcal{R}[s' \leftarrow r_3].\epsilon_n.\,(\overline{\alpha}[].B_{n+1} + \gamma[].B_{n-1} \mid (s).\overline{\gamma}[].S \mid (s').\,(\alpha[] \mid P)) \setminus \{\alpha, \beta, \gamma\}$$

by axiom 16, theorem 2 and the expansion theorem.

$$\begin{array}{l} = \mathcal{R}[s \leftarrow r_2].\,\mathcal{R}[s' \leftarrow r_3].\epsilon_n.\,(\quad (s).\tau.\,(B_{n-1} \mid S \mid (s' \dot{-} s).\,(\alpha[] \mid P)) \\ \qquad\qquad\qquad\qquad + (s').\tau.\,(B_{n+1} \mid (s \dot{-} s').\overline{\gamma}[].S \mid P)) \setminus \{\alpha, \beta, \gamma\} \end{array}$$

by the expansion theorem again.

Now we may use properties of the exponential distribution to rewrite this. Consideration of the μ measure of both sides shows that, given r_2,r_3 the rates of exponential distributions and for any process $P(s, s')$ with two time free variables:

$$\begin{array}{l} \mathcal{R}[s \leftarrow r_2].\,\mathcal{R}[s' \leftarrow r_3].P(s, s') = \mathcal{R}[s \leftarrow r_2 + r_3].\,\Big(\quad [\frac{r_2}{r_2+r_3}]\,\mathcal{R}[s' \leftarrow r_3].P(s, s + s') \\ \qquad\qquad\qquad\qquad\qquad\qquad \dot{+} [\frac{r_3}{r_2+r_3}]\,\mathcal{R}[s' \leftarrow r_2].P(s + s', s)\Big) \end{array}$$

We may then apply this general result:

$$Q_n = \mathcal{R}[s \leftarrow r_2 + r_3].\Big([\tfrac{r_2}{r_2+r_3}]\,\mathcal{R}[s' \leftarrow r_3].\epsilon_n.(s).\tau.\,(B_{n-1} \mid S \mid (s').\,(\alpha[] \mid P))$$

$$\dot{+}[\tfrac{r_3}{r_2+r_3}]\,\mathcal{R}[s' \leftarrow r_2].\epsilon_n.(s).\tau.\,(B_{n+1} \mid (s').\overline{\gamma}[].S \mid P)\Big) \setminus \{\alpha,\beta,\gamma\}$$

We may use theorem 4 axiom 1 to drop the τ (since it is protected by ϵ_n).

However to complete the transformation it is plain we require a coarser equivalence under which we would have the axiom

$$\alpha.\sum [q_i]Q_i = \sum [q_i]\alpha.Q_i$$

This axiom seems intuitively obvious at first. It states that whether you make a random choice before or after an action does not matter if the action is unaffected by the choice. It implies:

$$\begin{aligned}
Q_n = {} & \mathcal{R}[s \leftarrow r_2 + r_3].\epsilon_n.(s).\Big([\tfrac{r_2}{r_2+r_3}]\,(B_{n-1} \mid S \mid \mathcal{R}[s' \leftarrow r_3](s').\,(\alpha[] \mid P)) \\
& \dot{+}[\tfrac{r_3}{r_2+r_3}]\,(B_{n+1} \mid \mathcal{R}[s' \leftarrow r_2].(s').\overline{\gamma}[].S \mid P)\Big) \setminus \{\alpha,\beta,\gamma\} \\
= {} & \mathcal{R}[s \leftarrow r_2 + r_3].\epsilon_n.(s).\Big([\tfrac{r_2}{r_2+r_3}]\,(B_{n-1} \mid S \mid \mathcal{R}[s' \leftarrow r_3].(s').\,(\alpha[] \mid P)) \setminus \{\alpha,\beta,\gamma\} \\
& \dot{+}[\tfrac{r_3}{r_2+r_3}]\,(B_{n+1} \mid \mathcal{R}[s' \leftarrow r_2].(s').\overline{\gamma}[].S \mid P) \setminus \{\alpha,\beta,\gamma\}\Big) \\
= {} & \mathcal{R}[s \leftarrow r_1].\epsilon_n.(s).\,([p]Q_{n-1} \dot{+} [q]Q_{n+1})
\end{aligned}$$

as required where we take $p = \frac{r_2}{r_2+r_3}$, $q = \frac{r_3}{r_2+r_3}$, and $r_1 = r_2 + r_3$.

However, the axiom is not sound under bisimulation. This is because the LHS has a state in which it has performed α but not yet made the choice while the RHS has no such state. In fact it is only sound in a "linear" equivalence while bisimulation is a "branching" equivalence (see [27]). Moreover, there are problems with interactions in a non-deterministic context. If we define $P_1 = \alpha.\,\big([\frac{1}{2}]\beta.\mathsf{nil} \dot{+} [\frac{1}{2}]\gamma.\mathsf{nil}\big)$ and $P_2 = [\frac{1}{2}]\alpha.\beta.\mathsf{nil} \dot{+} [\frac{1}{2}]\alpha.\gamma.\mathsf{nil}$ we would have $P_1 = P_2$. But now consider $Q = \overline{\alpha}.\overline{\beta}.\delta.\mathsf{nil} + \overline{\alpha}.\overline{\gamma}.\delta.\mathsf{nil}$ which reacts with non-determinism to an α action, giving

$$\begin{aligned}
& (P_1 \mid Q) \setminus \{\alpha,\beta,\gamma\} \\
& \quad = \quad \tau.\,\big([\tfrac{1}{2}]\tau.\delta.\mathsf{nil} \dot{+} [\tfrac{1}{2}]\mathbf{0}\big) + \tau.\,\big([\tfrac{1}{2}]\mathbf{0} \dot{+} [\tfrac{1}{2}]\tau.\delta.\mathsf{nil}\big) \\
& \quad = \quad \tau.\,\big([\tfrac{1}{2}]\tau.\delta.\mathsf{nil} \dot{+} [\tfrac{1}{2}]\mathbf{0}\big)
\end{aligned}$$

while

$$\begin{aligned}
& (P_2 \mid Q) \setminus \{\alpha,\beta,\gamma\} \\
& \quad = \quad \big([\tfrac{1}{2}](\tau.\delta.\mathsf{nil} + \tau.\mathbf{0}) \dot{+} [\tfrac{1}{2}](\tau.\delta.\mathsf{nil} + \tau.\mathbf{0})\big) \\
& \quad = \quad [1](\tau.\delta.\mathsf{nil} + \tau.\mathbf{0})
\end{aligned}$$

The resulting processes differ non-trivially in that the second is non-deterministic whereas the first is not. This is a consequence of our retaining non-determinism in the language to have reasoning power in under-specified systems. Nevertheless, the axiom can be treated as a congruence in linear contexts without non-determinism.

8 Conclusion

We have formally presented a language powerful enough to write both arbitrary parallel programs (since it includes CCS) and arbitrary discrete event simulations. We do not rule out non-determinism, indeed we adopt the CCS approach of explaining parallelism via non-determinism. Our approach is to say that a process which exhibits non-determinism is not a

valid simulation in the sense that our chosen performance measures still have some dependence on the resolution of this non-determinism. In other words we have not adequately specified the behaviour of some parts of the system to derive its performance. Such a situation arises, for example, where some schedule for the behaviour of some part of the system is unspecified (e.g. it depends on operating system behaviour).

We have presented a formal semantics for the language and this allows us to prove properties of programs written in the language at a semantic level. We have also derived algebraic tools which allow us to transform programs at a syntactic level and for example find out if they do exhibit non-determinism. We can use these tools to transform a program written in a clear high-level style into one that is closer to well understood models of simulation such as GSMPs or is perhaps more efficient or simpler to understand. We might then hope to derive analytical properties of the simulation such as recurrence.

The language is relatively high-level and is related to languages in use to specify parallel systems and even to real programming languages such as Occam. However our examples show that manipulations can be complex to do by hand and in practice we would expect machine support to be important. We have also presented an example showing that our semantics distinguishes certain simulations that are generally accepted as being equal.

It is interesting that [17, 6] introduce a SPA coincidentally named SPADES (but more commonly known by the spades symbol ♠), several years after our own SPA was first published [32, 33]. This process algebra mimics ours in that the notion of action and delay are now separated into two distinct specification constructs. It uses *stochastic automata* as a semantic model. The authors show the (close) relationship between stochastic automata and GSMPs and come up with a simulation algorithm to extract performance metrics from their models. Notions of probability have to be modelled using the interaction between clocks. Were it not for the fact that the synchronisation paradigm in their approach is a broadcast based one, their models could be viewed as a straightforward subclass of ours. [2] used a class of *extended generalised semi-Markov processes* as the underlying performance model for their SPA. This SPA captures the notion of weights and priorities associated with actions. Unfortunately the approach maintains the idea that actions and durations are modelled by a single specification construct. This leads to the necessity of somewhat obfuscating the syntactic and semantic presentation of the SPA with constructs that are used to deal with the fact that, in a generalised framework, action durations are no longer *memoryless*.

References

[1] J. C. M. Baeten and J. A. Bergstra. Real Time Process Algebra. *Formal Aspects of Computing*, 3, 142–188, 1991.

[2] M. Bernardo and R. Gorrieri Extended Markovian Process Algebra. In *Proceedings of CONCUR '96*, LNCS 1119, Springer-Verlag, 1996.

[3] M. Bernardo, M. Bravetti and R. Gorrieri. Towards Performance Evaluation with General Distributions in Process Algebras. In *Proceedings of CONCUR '98*, LNCS 1466, 405–422, Springer-Verlag, 1998.

[4] L. Chen. An interleaving model for real–time systems. In Anil Nerode and Mikhail Taitslin, eds, *Proceedings of Logical Foundations of Computer Science (Tver '92)*, LNCS 620, 81–92, Springer-Verlag, 1992.

[5] Liang Chen. Timed Processes: Models, Axioms and Decidability. LFCS report ECS-LFCS-93-271 (also CST-101-93), University of Edinburgh, 1993.

[6] P. D'Argenio, J.-P. Katoen, and E. Brinksma. General Purpose Discrete Event Simulation using ♠. In *Proceedings of the 6th Workshop on Process Algebra and Performance Modelling, PAPM '98*, 85–102, 1998.

[7] Anthony J. Field and Peter G. Harrison. *Functional Programming.* Addison Wesley, 1988.

[8] N. Götz, U. Herzog, and M. Rettelbach. TIPP — a language for timed processes and performance evaluation. Technical Report 4/92 IMMD7, University of Erlangen-Nürnberg, November 1992.

[9] Hans A. Hansson. *Time and Probability in Formal Design of Distributed Systems.* Elsevier, 1994.

[10] Matthew Hennessy and Tim Regan. A process algebra for timed systems. *Information and Computation*, 117(2), 221–239, March 1995.

[11] H. Hermanns, M. Rettlebach. Syntax, Semantics, Equivalences and Axioms for MTIPP In *Proceedings of the 2nd Workshop on Process Algebra and Performance Modelling, PAPM '94*, 1994.

[12] Holger Hermanns and Michael Rettlebach. Towards a superset of basic lotos for performance prediction. In *Proceedings of the 4th Workshop on Process Algebra and Performance Modelling, PAPM '96*, 1996.

[13] H. Hermanns, M. Rettelbach and T. Weiss. Formal characterisation of immediate actions in SPA with nondeterministic branching. *The Computer Journal*, 38(7), 530–541, 1995.

[14] J. Hillston. *A Compositional Approach to Performance Modelling.* PhD thesis, University of Edinburgh, 1994.

[15] Donald L. Iglehart and Gerald S. Shedler. Simulation of non-Markovian systems. *IBM Journal of Research and Development*, 27(5), 472–479, September 1983.

[16] K. Kanani. *A Unified Framework for Systematic Quantitative and Qualitative Analysis of Communicating Systems.* PhD thesis, Imperial College of Science, Technology and Medicine. University of London, August 1998.

[17] J.-P. Katoen, P.R. D'Argenio and E. Brinksma. A stochastic automata model and its algebraic approach. Ctit technical report, process algebra and performance modelling, fifth international workshop, University of Twente, June 1997.

[18] Luc Léonard and Guy Leduc. An introduction to et-lotos for the description of time-sensitive systems. *Computer Networks and ISDN Systems*, 29, 271–292, February 1997.

[19] M. Loève. *Probability Theory.* Van Nostrand, 1963.

[20] C. Miguel, A. Fernández, and L. Vidaller. A LOTOS based performance tool. *Computer Networks and ISDN Systems*, (25), 791–814, 1993.

[21] R.E. Milne and C. Strachey. *A Theory of Programming Language Semantics.* Chapman and Hall, 1976.

[22] Robin Milner. *Communication and Concurrency.* Prentice Hall, 1989.

[23] Faron Moller and Chris Tofts. A temporal calculus of communicating systems. In J. C. M. Baeten and J. W. Klop, eds, *Proceedings of CONCUR '90: Theories of Concurrency: Unification and Extension*, LNCS 458, 401-415, Springer-Verlag, 1990.

[24] Faron Moller and Chris Tofts. Behavioural abstraction in TCCS. In *Proceedings of the 19th Colloquium on Automata Languages and Programming*, LNCS 623, 579–570, Springer-Verlag, 1992.

[25] Xavier Nicollin and Joseph Sifakis. An overview and synthesis on timed process algebras. In Kim G. Larsen and Arne Skou, editors, *Proceedings of CAV 91*, LNCS 575, 376–398, Springer-Verlag, 1991.

[26] I. Phillips. Refusal testing. *Theoretical Computer Science*, 50, 241–284, 1987.

[27] A. Pnueli. Linear and branching structures in the semantics and logics of reactive systems. In *Proceedings of the Twelfth Colloquium on Automata Languages and Programming*, LNCS 194, Springer-Verlag, 1985.

[28] Corrado Priami. Stochastic π-calculus. *The Computer Journal*, 38(7), 1995.

[29] Corrado Priami. Stochastic π-calculus with General Distributions. In *Proceedings of the 4th Workshop on Process Algebra and Performance Modelling, PAPM '96*, 1996.

[30] Steve Schneider. An operational semantics for timed CSP. *Information and Computation*, 116(2), 193–213, 1995.

[31] Gerald S. Shedler. *Regenerative Stochastic Simulation.* Academic Press, 1993.

[32] B. Strulo. *Process algebra for discrete event simulation.* PhD thesis, Imperial College of Science, Technology and Medicine. University of London, October 1993.

[33] B. Strulo and P. Harrison. Stochastic process algebra for discrete event simulation. In *Quantitative Methods in Parallel Systems*, Esprit Basic Research Series, 18–37, Springer-Verlag, 1995.

[34] Ward Whitt. Continuity of Generalized Semi-Markov Processes. *Mathematics of Operations Research*, 5, 14–23, 1980.

[35] Wang Yi. *A Calculus of Real Time Systems.* PhD thesis, Chalmers University of Technology, S-412 96 Göteborg, Sweden, 1991.

[36] Wang Yi. CCS + Time = an interleaving model for real time systems. In *Proceedings of the Eighteenth Colloquium on Automata Languages and Programming*, LNCS 510, 217–228, Springer-Verlag, 1991.