Process Algebra for Discrete Event Simulation

P.G. Harrison* **B. Strulo†**
Department of Computing
Imperial College
LONDON SW7 2BZ

Abstract

We present a process algebra or programming language, based on CCS, which may be used to describe discrete event simulations with parallelism. It has extensions 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. It has a clear operational semantics and we give approaches to denotational semantics given in terms of an algebra of equivalences over processes. It raises questions about when two simulations are equivalent and what we mean by non-determinism in the context of the specification of a simulation. It also exemplifies some current approaches to adding time and probability to process algebras.

1 Introduction

Imagine we wish to simulate the behaviour of a complex system with computerised components, such as a telephone network. First, let us look at the implementation of such a complex system. When it is implemented, 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.

Two directions are apparent in the evolution of these implementation processes. Firstly the development of appropriate high level languages, both for execution and specification, has been essential to manage this complexity. Secondly much work has gone into understanding the semantics of such languages formally with the intention of reasoning mathematically about the implementations.

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

*Supported by the CEC as part of the ESPRIT-BRA QMIPS project 7269
†Supported by a grant from the SERC

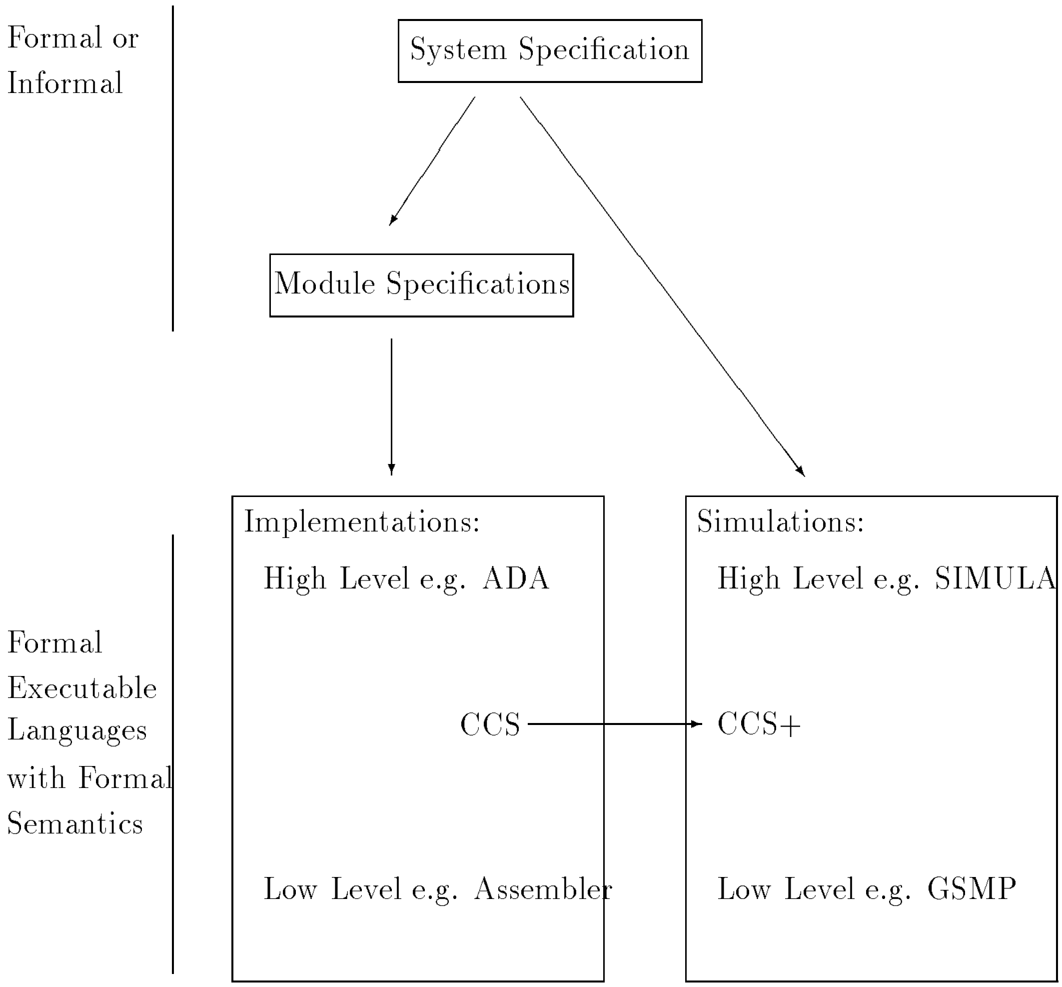

Figure 1: Relationship between implementation and simulation

model which may be used either for analysis or for execution to determine performance experimentally. It seems that even if our intention is abstract analysis we must have in mind some underlying simulation in this sense. So we look for formalisms that are both high level and have a formal semantics. In figure 1 we show schematically the kind of comparison we are considering. Here CCS+ represents the language we will describe.

We require a formal semantics because such a semantics makes 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. However, 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

[7].

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. But we must still have a rigorous semantics and thus make our formalism a bridge between informal descriptions of a system and mathematical models of that system.

So by making the language high level we aim to reduce the chance of the original simulation being unfaithful to the specification and by making its semantics formal we aim to reduce the chance of any simplifications being unfaithful to the original.

What is the difference between an implementation and a simulation? Firstly, in a simulation, we need to be able to model our workloads while for an implementation they are supplied by the outside world. Further we need to model them in some detail so as to represent the demands we expect them to impose on our implementation. Similarly when we abstract from detail in our simulation by regarding sub-modules as black boxes we need to describe their behaviour in more detail than may interest an implementor since we want to represent their performance in addition to their abstract behaviour. Thus in a simulation we need to be able to describe both probabilistic behaviour with explicit distributions and also explicit behaviour in time.

Secondly, in a simulation, we are often much more concerned directly with parallelism. Single modules in an implementation are generally sequential and so the implementor may be able to use sequential languages both for implementation and for module specification. Essentially he may be able to move consideration of parallelism up to the level of the system specification. In simulation we are modelling the behaviour of the whole system. Our simulation has to be faithful to the whole system specification and so we will inevitably need to be able to describe parallelism explicitly if this is how the system is to operate.

Work on formal semantics has very clearly elucidated the behaviour of sequential systems. Formal semantics for parallelism have proved more controversial and it is only recently that some consensus about the main approaches has been reached. Surveying the literature has suggested process algebras as one of the leading contenders [8, 12]. Many approaches to extending process algebras with actual time values [3, 1, 2, 6, 13, 16, 10] or with time and probabilities [5] have also been discussed.

We use a selection from these approaches with some new techniques to obtain a process algebra which may be used to describe discrete event simulations. So we take a relatively high level language that has a formal semantics of parallelism (CCS from [8]) and add constructs to represent a real-number time and various probabilistic features. We then extend its 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 [4].

We present a process algebra or programming language, based on CCS, which may be used to describe concurrent processes. It has extensions 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. It thus has sufficient facilities to specify discrete event simulations. It has a clear operational semantics and we discuss equivalences over these operational semantics that imply equations on processes that can be used to work formally with processes.

Because it allows both non-determinism and probabilistic choice, it can be used to show the difference between these concepts and their relevance to simulation. It also exemplifies some current approaches to adding time and probability to process algebras.

2 Language

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

The language is based on the timed CCS of [15] with some features of the temporal CCS of [9] though their calculus does not generalise directly to real number time and they have more power in that they have a weak + operator that we do not. It ends up very close to [2] except 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 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. Our rules for evolution (as in the mentioned calculi) does not behave in exactly the way expected for the time required to compute. Thus for example it is easy to write programs which perform an infinite amount of computation in finite time and thus force time to stop for the whole system. Other processes simply deadlock in time and also prevent time passing in the whole system. We justify this by interpreting time evolution as representing simulated time rather than the time in which real calculation takes place.

As in CCS, communication and synchronisation are represented via an alphabet of atomic actions or labels. 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 an alphabet Act including names, co-names and the invisible action τ.

We use two sorts and 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 with this new sort of Time. We then distinguish four forms of prefixing, one which describes a prefix of a timed delay, represented (t). This uses the value of the time expression t which may have free variables. We use two separate forms of action prefixing. Simple prefixing $\alpha.P$ is distinguished from $\alpha[s \leftarrow t]$ where the action prefix binds the time variable s to the time of occurrence of the action

α. Then the final form binds time variables to the sampling of a random variable. This is represented $\mathcal{R}[s \leftarrow f]$.

Finally we add a probabilistic choice, represented by $\sum_{i\in I}[q_i]P_i$.

So 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 distribution function taking values in* $\mathbb{R}^+$*.*

Here nil is a process constant representing a deadlocked or terminated process which can perform no action and must simply idle from now on. $(t).P$ represents a process which must delay for t time units and then becomes process P.

$\alpha.P$ and $\alpha[s \leftarrow t].P$ both represent processes which can immediately perform an α action. If they do they become P but in the latter case the free variable s is replaced by the value t and so becomes $P\{t/s\}$. The essential difference is that the former prefixing is "impatient" and cannot delay. The latter expression can delay for any time t' but records this delay and becomes $\alpha[s \leftarrow t+t'].P$. This latter 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 it from impatient prefixing).

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. Then the process may act on that value later, perhaps by delaying less later if it has already been delayed.

$P+Q$ represents a process which can evolve until P or Q are ready to participate in a labelled transition. 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.

$\sum_{i\in I}[q_i]P_i$ 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.

$\mathcal{R}[s \leftarrow f].P$ represents a process which can immediately sample a random variable with the specified distribution and become $P\{t/s\}$ for some t. Although this is a form of uncountable probabilistic branching the resulting processes are highly restricted. They are identical except in the values of certain time expressions and hence certain delays.

The other constructs have the same meaning as basic CCS. We define free variables of either sort in the standard way.

3 Semantics

3.1 Transition system

The operational semantics of these processes are described in terms of three sorts of transitions between processes. Thus we envisage every state in the system as represented by a process and the transitions representing state changes. We define probabilistic transitions which we assume to 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.

This assumption that immediate transitions take priority over evolution is sometimes called the Maximal Progress assumption and is discussed in [15]. In fact our language is more expressive than his since we can also write impatient actions that are not τ *e.g.* in processes such as $Clock = tick.(1).Clock$ which ticks once a time unit. Here each tick cannot be made to idle and so the clock cannot be made to run slow.

While this is similar to some ideas of timed calculi we envisage it rather with two separate notions of time. We might imagine that immediate transitions must take some real time to process. But this is not the simulated time that passes during evolution transitions. Thus we imagine the system performing calculations to decide its next step and resolving any exposed probabilistic choices. It calculates until eventually stability is reached. Then all processes co-operatively allow simulated time to pass until at least one process is ready to perform more calculations.

The possible transitions are described via inference systems in the usual way. We say the transitions are the least relation between processes that satisfies the given inference rules.

3.2 Probabilistic transitions

We start with the probabilistic transitions in Figure 2. We name the rules for use in proofs. Each transition has an associated probability measure, shown above it. It also has an associated index, shown as a subscript. We give a syntax for indices as

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 basic indices, Index_0*, is the least set including*

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

where $l, l' \in \mathsf{Index}_0$, $a \in \mathsf{Atom}$.

Note the way there is a single a value carried throughout the inference and on top of it an index "skeleton" full of $\langle\rangle$. Note also the strong way we wish probabilistic transitions to happen first - the Delay rule. Only a labelled prefix can hide a probabilistic transition. However this is just enough to prevent the inference reaching down to a recursion and an infinite chain of inference.

This relation captures our fundamental idea of how we wish 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 our actual relation with another inference system, figure 3. Here we use a syntax for indices as

Definition 3 *The set of general indices,* Index, *is the least set including*

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

where $l, l' \in \mathsf{Index}$, $a \in \mathsf{Atom}$, $i \in \mathbb{N}$, $t \in \mathbb{R}^+$.

Note that $\mathsf{Index}_0 \subset \mathsf{Index}$.

Here we use the lf function to generate 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 $\mid$.

By our guardedness restriction we know that for every process there is some maximum length of inference both for $\underset{0}{\hookrightarrow}$ and for $\hookrightarrow$. We will call these numbers $maxp_0(P)$ and $maxp(P)$. We will also regularly perform induction by the lexicographic ordering on $(maxp(P), maxp_0(P))$. We will call this "induction on $maxp$".

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 distribution over the transitions from P.

Our first problem is that this transition relation gives derivations which are prefixes of each other. For example we have both

$$\mathcal{R}[s \leftarrow f].\,\mathcal{R}[s' \leftarrow f'].P \overset{f(t)}{\hookrightarrow}_{[t]} \mathcal{R}[s' \leftarrow f'].P\{t/s\}$$

$$\mathcal{R}[s \leftarrow f].\,\mathcal{R}[s' \leftarrow f'].P \overset{f(t).f(t')}{\hookrightarrow}_{[t].[t']} P\{t/s\}\{t'/s'\}$$

We cannot easily interpret these as probabilities since here for example the probability of transiting to the set $\{\mathcal{R}[s' \leftarrow f'].P\{t/s\} : t \in \mathbb{R}^+\} \cup \{P\{t/s\}\{t'/s'\} : t, t' \in \mathbb{R}^+\}$ could

$$\frac{}{\sum_{i\in I}[q_i]P_i \overset{q_i}{\underset{0}{\hookrightarrow}}_{[i].\langle\rangle} P_i} \quad \text{Resolve}$$

$$\frac{}{\mathcal{R}[s\leftarrow f].P \overset{f(t)}{\underset{0}{\hookrightarrow}}_{(t).\langle\rangle} P\{t/s\}} \quad \text{Sample}$$

$$\frac{P \overset{q}{\underset{0}{\hookrightarrow}}_l P'}{(t).P \overset{q}{\underset{0}{\hookrightarrow}}_l (t).P'} \quad \text{Delay}$$

$$\frac{P \overset{q}{\underset{0}{\hookrightarrow}}_l P'}{P+Q \overset{q}{\underset{0}{\hookrightarrow}}_{(l+\langle\rangle,a)} P'+Q} \quad \text{Sum1}$$

$$\frac{Q \overset{q}{\underset{0}{\hookrightarrow}}_l Q'}{P+Q \overset{q}{\underset{0}{\hookrightarrow}}_{\langle\rangle+l} P+Q'} \quad \text{Sum2}$$

$$\frac{P \overset{q}{\underset{0}{\hookrightarrow}}_l P'}{P \mid Q \overset{q}{\underset{0}{\hookrightarrow}}_{l|\langle\rangle} P' \mid Q} \quad \text{Parallel1}$$

$$\frac{Q \overset{q}{\underset{0}{\hookrightarrow}}_l Q'}{P \mid Q \overset{q}{\underset{0}{\hookrightarrow}}_{\langle\rangle|l} P \mid Q'} \quad \text{Parallel2}$$

$$\frac{P \overset{q}{\underset{0}{\hookrightarrow}}_l P'}{P[S] \overset{q}{\underset{0}{\hookrightarrow}}_l P'[S]} \quad \text{Relabel}$$

$$\frac{P \overset{q}{\underset{0}{\hookrightarrow}}_l P'}{P\backslash A \overset{q}{\underset{0}{\hookrightarrow}}_l P'\backslash A} \quad \text{Restrict}$$

$$\frac{P\{\mathsf{rec}(X=P)/X\} \overset{q}{\underset{0}{\hookrightarrow}}_l P'}{\mathsf{rec}(X=P) \overset{q}{\underset{0}{\hookrightarrow}}_l P'} \quad \text{Fix}$$

Figure 2: Basic probabilistic transitions

$$\frac{P \overset{p}{\underset{0}{\hookrightarrow}}_l P'}{P \overset{p}{\hookrightarrow}_l P'} \quad \text{Any-Samples}$$

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

Figure 3: Actual probabilistic transitions

naïvely be evaluated to 2. The simplest way round this is to only look at descendants which cannot $\hookrightarrow$.

Thus we define

Definition 5 *For a given process, P,*

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

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

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

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

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

We need to show these definitions are consistent so we prove

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

This shows we have some sort of probability "distribution" function. But over what can we integrate this distribution? We start by treating it as 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.

Now we prove

Lemma 2 *p_P is measurable.*

Indices are trees of real and natural numbers and so it is straightforward to define measurable sets. We define:

Definition 6 Borel *is the σ-field of measurable sets from* Index.

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

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

$$\mu^*(P, I) = \int_I p_P(l)\, dl$$

We then wish to move this structure over to Proc. But what is the appropriate σ-field of measurable sets on 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*

$$\overline{B}_P = q_P^{-1}(\mathsf{Borel}) = \{S : \exists I \in \mathsf{Borel} \quad S = q_P^{-1}(I)\}$$

The σ-field of measurable sets is $\overline{B}$ where

$$\overline{B} = \bigcap_{P \in \mathsf{Proc}} \overline{B}_P$$

$$\frac{}{\alpha.P \overset{\alpha}{\to} P} \qquad \text{Prefix1}$$

$$\frac{}{\alpha[s \leftarrow t].P \overset{\alpha}{\to} P\{t/s\}} \qquad \text{Prefix2}$$

$$\frac{P \overset{\alpha}{\to} P'}{(0).P \overset{\alpha}{\to} P'} \qquad \text{No-Delay}$$

$$\frac{P \overset{\alpha}{\to} P' \quad \neg P \hookrightarrow \quad \neg Q \hookrightarrow}{P + Q \overset{\alpha}{\to} P'} \qquad \frac{Q \overset{\alpha}{\to} Q' \quad \neg P \hookrightarrow \quad \neg Q \hookrightarrow}{P + Q \overset{\alpha}{\to} Q'} \qquad \text{Sum}$$

$$\frac{P \overset{\alpha}{\to} P' \quad \neg Q \overset{q}{\hookrightarrow}_l}{P \mid Q \overset{\alpha}{\to} P' \mid Q} \qquad \frac{Q \overset{\alpha}{\to} Q' \quad \neg P \overset{q}{\hookrightarrow}_l}{P \mid Q \overset{\alpha}{\to} P \mid Q'} \qquad \text{Parallel}$$

$$\frac{P \overset{\alpha}{\to} P' \quad Q \overset{\overline{\alpha}}{\to} Q' \quad \neg P \mid Q \overset{q}{\hookrightarrow}_l}{P \mid Q \overset{\tau}{\to} P' \mid Q'} \qquad \text{Communicate}$$

$$\frac{P \overset{\alpha}{\to} P'}{P[S] \overset{S(\alpha)}{\to} P'[S]} \qquad \text{Relabel}$$

$$\frac{P \overset{\alpha}{\to} P'}{P \backslash A \overset{\alpha}{\to} P' \backslash A} \quad \alpha \notin A \text{ and } \overline{\alpha} \notin A \qquad \text{Restrict}$$

$$\frac{P\{\mathsf{rec}(X = P)/X\} \overset{\alpha}{\to} P'}{\mathsf{rec}(X = P) \overset{\alpha}{\to} P'} \qquad \text{Fix}$$

Figure 4: Labelled transitions

Now we may define our probability distribution over this σ-field in the usual way 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 sampling transition. Finally, to prove that $\mu(P, .)$ is a probability we need

Lemma 3 $\mu(P, \mathsf{Proc}) = 1$ *for all* P *with* $P \hookrightarrow$.

Thus, for each process P, these probabilistic transitions induce a measure $\mu(P, .)$ over the σ-field of sets of processes $\overline{B}$.

3.3 Labelled transitions

These are defined in Figure 4. These are very similar to the standard CCS rules. There are preconditions to force all probabilistic choices to be resolved first. 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 $(0).P = P$.

3.4 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 [15] 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

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

But consider $P \equiv \alpha$, $Q \equiv \overline{\alpha}$. Both of these can delay arbitrarily but in parallel we wish them to be forced to act immmediately without delay. In [15] this is done by "Timed Sorts". We use a slightly different approach.

First we define a simple evolution relation $\overset{t}{\leadsto}_\circ$. Note that we will not distinguish between time expressions and time values. The definitions are in Figure 5.

Note that 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.

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 $(1).(1).P \overset{1}{\leadsto}_\circ (1).P \overset{1}{\leadsto}_\circ P$ but not $(1).(1).P \overset{2}{\leadsto}_\circ P$. 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 actual evolution transitions are thus defined in Figure 6. Thus we now have $(1).(1).P \overset{1}{\leadsto}_\circ (1).P$ and so by Any-Delay we have $(1).(1).P \overset{1}{\leadsto} (1).P$. But we also have $(1).P \overset{1}{\leadsto}_\circ P$ and so by Add-Delay we can infer $(1).(1).P \overset{2}{\leadsto} P$ as required.

3.5 Properties

We have stated informally that if our processes can sample they must do so first. We prove:

Proposition 1 *If* $P \overset{q}{\hookrightarrow}_l$ *then* $\neg P \overset{\alpha}{\rightarrow}$ $\quad \neg P \overset{t}{\leadsto}$.

Thus if probabilistic transitions are possible they are the only transitions possible and 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 [11] gives a concise collection of important properties of various systems and we will adopt their terminology for these.

3.5.1 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:

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

$$\frac{\neg P \hookrightarrow}{(t+u).P \overset{t}{\rightsquigarrow}_{\circ} (u).P} \quad t > 0 \qquad \text{Reduce-Delay}$$

$$\frac{P \overset{t}{\rightsquigarrow}_{\circ} P'}{(0).P \overset{t}{\rightsquigarrow}_{\circ} P'} \qquad \text{No-Delay}$$

$$\frac{\neg P \hookrightarrow}{(t).P \overset{t}{\rightsquigarrow}_{\circ} P} \quad t > 0 \qquad \text{End-Delay}$$

$$\frac{}{\mathsf{nil} \overset{t}{\rightsquigarrow}_{\circ} \mathsf{nil}} \quad t > 0 \qquad \text{Nil}$$

$$\frac{}{\alpha[s \leftarrow t].P \overset{t'}{\rightsquigarrow}_{\circ} \alpha[s \leftarrow t+t'].P} \quad \alpha \neq \tau \text{ and } t' > 0 \qquad \text{Idle}$$

$$\frac{P \overset{t}{\rightsquigarrow}_{\circ} P' \quad Q \overset{t}{\rightsquigarrow}_{\circ} Q'}{P + Q \overset{t}{\rightsquigarrow}_{\circ} P' + Q'} \qquad \text{Sum}$$

$$\frac{\neg P \mid Q \xrightarrow{\tau} \quad P \overset{t}{\rightsquigarrow}_{\circ} P' \quad Q \overset{t}{\rightsquigarrow}_{\circ} Q'}{P \mid Q \overset{t}{\rightsquigarrow}_{\circ} P' \mid Q'} \qquad \text{Parallel}$$

$$\frac{P \overset{t}{\rightsquigarrow}_{\circ} P'}{P[S] \overset{t}{\rightsquigarrow}_{\circ} P'[S]} \qquad \text{Relabel}$$

$$\frac{P \overset{t}{\rightsquigarrow}_{\circ} P'}{P\backslash A \overset{t}{\rightsquigarrow}_{\circ} P'\backslash A} \qquad \text{Restrict}$$

$$\frac{P\{\mathsf{rec}(X = P)/X\} \overset{t}{\rightsquigarrow}_{\circ} P'}{\mathsf{rec}(X = P) \overset{t}{\rightsquigarrow}_{\circ} P'} \qquad \text{Fix}$$

Figure 5: Simple Evolution transitions

$$\frac{P \overset{t}{\rightsquigarrow}_{\circ} P'}{P \overset{t}{\rightsquigarrow} P'} \qquad \text{Any-Delay}$$

$$\frac{P \overset{t}{\rightsquigarrow} P' \quad P' \overset{t'}{\rightsquigarrow}_{\circ} P''}{P \overset{t+t'}{\rightsquigarrow} P''} \qquad \text{Add-Delay}$$

Figure 6: Extended Evolution transitions

3.5.2 Time additivity

Most other systems also require that if a process can idle for some period it can also idle for any smaller period ([16] - "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 $P \overset{t+t'}{\leadsto} P'$ does not have the lurking capability of a $P \overset{t}{\leadsto} P'' \overset{\tau}{\rightarrow}$ which would imply $\neg\, P'' \overset{t'}{\leadsto} P'$.

We prove two intermediate lemmas.

Lemma 4 *If $P \overset{t}{\leadsto}_\circ P'$ and $P \overset{t'}{\leadsto}_\circ P''$ with $t' > t$ and $P' \overset{\alpha}{\rightarrow}$ then $P \overset{\alpha}{\rightarrow}$.*

Proof
Induction on length of inference of $P \overset{t'}{\leadsto}_\circ$.

- Reduce-Delay: $P \equiv (t'+u).Q$ and so $P' \equiv (t'-t+u).Q$ and so $\neg P' \overset{\alpha}{\rightarrow}$
- Nil & End-Delay similar
- No-Delay: $P \equiv (0).Q$
  So $Q \overset{t}{\leadsto}_\circ Q'$ and $Q \overset{t'}{\leadsto}_\circ$ and $Q' \overset{\alpha}{\rightarrow}$
  So $Q \overset{\alpha}{\rightarrow}$ by ind. hyp. and so $P \overset{\alpha}{\rightarrow}$ as required.
- Idle: $P \equiv \alpha[s \leftarrow t].Q$ and so $P \equiv P'$
- Sum: $P \equiv P_1 + P_2$
  So $P_1 \overset{t'}{\leadsto}_\circ$ and $P_1 \overset{t}{\leadsto}_\circ P_i'$ and similarly for P_2.
  Also w.l.o.g. $P_1' \overset{\alpha}{\rightarrow}$. But since $P_1 \overset{t'}{\leadsto}_\circ$ is a shorter inference $P_1 \overset{\alpha}{\rightarrow}$ and hence $P \overset{\alpha}{\rightarrow}$ as required.
- Parallel: $P \equiv R \mid Q$
  So $R \overset{t'}{\rightarrow}$ and $Q \overset{t'}{\rightarrow}$ and $R \overset{t}{\rightarrow} R'$ and $Q \overset{t}{\rightarrow} Q'$
  Also $R' \mid Q' \overset{\alpha}{\rightarrow}$ so either
  - $R' \overset{\alpha}{\rightarrow}$ and so $R \overset{\alpha}{\rightarrow}$ by ind. hyp. and so $P \overset{\alpha}{\rightarrow}$ as required.
  - $Q' \overset{\alpha}{\rightarrow}$ similarly.
  - $R' \overset{\beta}{\rightarrow}$ and $Q' \overset{\overline{\beta}}{\rightarrow}$. Then by ind. hyp $R \mid Q \overset{\tau}{\rightarrow}$ which contradicts $P \overset{t'}{\leadsto}_\circ$
- Other inferences trivial inductive steps.

□

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.* $(1).\alpha[].P \overset{2}{\leadsto} \alpha[].P$.

Lemma 5 *If $P \overset{t+t'}{\leadsto}_\circ P''$ then $\exists P'$ with $P \overset{t}{\leadsto}_\circ P'$ and $P' \overset{t'}{\leadsto}_\circ P''$.*

Proof
Induction on length of inference of $P \overset{t+t'}{\leadsto}_\circ P'$. The Parallel step requires the use of lemma 4 to show that $\neg P' \overset{\tau}{\rightarrow}$. □

Proposition 3 *If* $P \overset{t+t'}{\rightsquigarrow} P''$ *then* $\exists P'$ *with* $P \overset{t}{\rightsquigarrow} P'$ *and* $P' \overset{t'}{\rightsquigarrow} P''$.

Proof
Induction on length of inference of $P \overset{t+t'}{\rightsquigarrow} P'$.

- Any-Delay: $P \overset{t+t'}{\rightsquigarrow}_{\circ} P''$ and so $\exists P'$ with $P \overset{t}{\rightsquigarrow}_{\circ} P'$ and $P' \overset{t'}{\rightsquigarrow}_{\circ} P''$. Thus $\exists P'$ with $P \overset{t}{\rightsquigarrow} P'$ and $P' \overset{t'}{\rightsquigarrow} P''$ as required.
- Add-Delay: We have $P \overset{t+t'}{\rightsquigarrow} P''$ because $P \overset{u}{\rightsquigarrow} Q$ and $Q \overset{u'}{\rightsquigarrow}_{\circ} P''$ with $u + u' = t + t'$. If $t = u$ then the result is trivial. Otherwise, if $t < u$ then we have the result by considering $P \overset{t+(u-t)}{\rightsquigarrow} Q$ for which the result is true by the inductive hypothesis. Alternatively, if $u < t$ then we may consider $Q \overset{(t-u)+(u'-(t-u))}{\rightsquigarrow}_{\circ} P''$ for which we know the result to be true.

$\square$

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.

3.5.3 Deadlock-freeness

In our calculus we do have states (such as $(\alpha.\mathsf{nil})\backslash\alpha$) which can do nothing - not even an evolution transition - since their impatient action prevents them from evolving. Thus our calculus does not have the property of "deadlock-freeness" from [11]. Such time-stop processes are intuitively difficult to understand. However time evolution in our calculus is intended to represent simulated time passing and so a process which prevents this passing is simply part of a simulation which fails to reach its time-advance step and so is not a well formed simulation but is a understandable program. We will define a symbol to represent such programs

Definition 10 *The time stop program is*

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

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

3.5.4 Action urgency

As already discussed, we have both impatient and patient actions (the notion of "urgency" from [11]). However our calculus is unusual in that it also has the maximal progress ("minimal delay" or "tau-urgency") property.

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

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

3.5.5 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* $P \overset{t}{\leadsto} Q$ *and* $P \xrightarrow{\alpha}$ *then* $Q \xrightarrow{\alpha}$.

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

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

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

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

3.5.6 Finite-variability

In our calculus we can easily define processes that perform infinite computation in finite time. Thus we do not have the "finite-variability" ("non-Zenoness, well-timedness") property and also not the "bounded control" property. Questions of whether this is realistic or satisfactory are less relevant in this simulation context.

3.5.7 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.

Lemma 6 *If* P *is a process and* $P \xrightarrow{\alpha} P'$ *then* P' *is a process.*
If P *is a process and* $P \overset{q}{\hookrightarrow}_l P'$ *then* P' *is a process.*
If P *is a process and* $P \overset{t}{\leadsto} P'$ *then* P' *is a process.*

3.6 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 them. Either they are *immediate* states with a countable set of immediate transitions to other states, or they are *sampling* 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 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 number. Then further computations take place.

4 Strong bisimulation

We now wish to define an equivalence over these process expressions. For the immediate and evolution transitions the standard notion of bisimulation seems appropriate while for the probabilistic transitions the notion is similar but slightly more complex.

We wish to make processes equivalent only if they define equal measures over their descendants. 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 including P_i and not Q_i for example. So we define the sub σ-field over which we require the measures to be equal.

Definition 11 *For an equivalence relation $\mathcal{R} \subset \mathsf{Proc} \times \mathsf{Proc}$ we define $\overline{B}(\mathcal{R}) \subset \overline{B}$ as*

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

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

Then we may define our fundamental equivalence:

Definition 12 *An equivalence relation $\mathcal{R} \subset \mathsf{Proc} \times \mathsf{Proc}$ is a* strong bisimulation *iff $P \mathrel{\mathcal{R}} Q$ 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 $P' \mathrel{\mathcal{R}} Q'$.*
2. *if $Q \xrightarrow{\alpha} Q'$, then $\exists P'$ such that $P \xrightarrow{\alpha} P'$ and $P' \mathrel{\mathcal{R}} Q'$.*
3. *if $P \overset{t}{\leadsto} P'$, then $\exists Q'$ such that $Q \overset{t}{\leadsto} Q'$ and $P' \mathrel{\mathcal{R}} Q'$.*
4. *if $Q \overset{t}{\leadsto} Q'$, then $\exists P'$ such that $P \overset{t}{\leadsto} P'$ and $P' \mathrel{\mathcal{R}} Q'$.*
5. *If P and Q are probabilistic then $\mu(P, S) = \mu(Q, S)$.*

Part 5 of this definition states that the equivalence is "good" if the total probability of moving from P to any set of states which isn't "too fine" is the same for any related process Q.

Definition 13 *We say two processes P and Q are in strong bisimulation, denoted by $P \sim Q$, iff there is any strong bisimulation with $P \mathrel{\mathcal{R}} Q$.*

Thus

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

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*

We show $+$ and $|$ are commutative and associative and have nil as a unit. We show $\backslash A$ and $[S]$ pass through the other operators and rec satisifes the usual law.

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. $P \mid Q \sim Q \mid P$
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 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 in the form $\sum(t_i).\,\alpha_i[s_i \leftarrow t_i].P_i$.

These last are not very remarkable. The more interesting laws are those that follow:

Theorem 2

- $(t).\mathsf{nil} \sim \mathsf{nil}$
- $(0).P \sim P$
- $(t)(t').P \sim (t+t').P$
- $(t).(P+Q) \sim (t).P + (t).Q$
- $(t).(P \mid Q) \sim (t).P \mid (t).Q$
- $\tau[s \leftarrow t].P \sim \tau.P\{t/s\}$
- $\alpha[s \leftarrow t].P \sim \alpha[s \leftarrow 0].P\{s+t/s\}$
- $\tau[s \leftarrow 0].P + (t).P' \sim \tau[s \leftarrow 0].P \quad \textit{if } t > 0$
- $\alpha.P + (t).P' \sim \alpha.P \quad \textit{if } t > 0$
- $P + \sum_{i \in I}[q_i]P_i \sim \sum_{i \in I}[q_i]\,(P + P_i)$
- $P \mid \sum_{i \in I}[q_i]P_i \sim \sum_{i \in I}[q_i]\,(P \mid P_i)$
- $\sum_{i \in I}[p_i]P_i + \sum_{j \in J}[q_j]Q_j \sim \sum_{i \in I, j \in J}[p_i.q_j]\,(P_i + Q_j)$
- $\mathcal{R}[s \leftarrow f].(t).P \sim (t).\,\mathcal{R}[s \leftarrow f].P$
- $P + \mathcal{R}[s \leftarrow f].Q \sim \mathcal{R}[s \leftarrow f].(P+Q)$
- $P \mid \mathcal{R}[s \leftarrow f].Q \sim \mathcal{R}[s \leftarrow f].(P \mid Q)$
- $\alpha[s \leftarrow 0].P + (t).\,\alpha[s \leftarrow t].P \sim \alpha[s \leftarrow 0].P$

The most important result however is the expansion theorem.

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*

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

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*

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

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*

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

Note that in the second two expansions, various terms with the least leading delay will pre-empt any of the other terms with a larger initial (t).

We extend bisimulation to process expressions in the usual way.

Definition 14 *For process expressions with free process variables* $\tilde{X}$ *and time variables* $\tilde{s}$ *at most,* $P \sim Q$ *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* $\tilde{t}$.

Then we obtain

Theorem 4 $\sim$ *is a congruence i.e.*

- $(t).P \sim (t).Q$
- $\alpha.P \sim \alpha.Q$
- $\alpha[s \leftarrow t].P \sim \alpha[s \leftarrow t].Q$
- $\mathcal{R}[s \leftarrow f].P \sim \mathcal{R}[s \leftarrow f].Q$
- $P + R \sim Q + R$
- *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 Weak bisimulation

As for standard CCS we wish to abstract from the τ action by moving to a weak version of bisimulation.

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

Definition 15 *For $a \in \text{Act}$ we write*

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

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

Here ε represents the empty word. Note that $P \overset{\varepsilon}{\Rightarrow} P$.

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

Definition 16

$$\begin{aligned} &P \overset{t}{\rightrightarrows} Q \quad \textit{iff} \\ &\quad \exists n > 0, m_i \geq 0, \ \textit{and times } t_i > 0 \quad \textit{with } P \overset{\tau^{m_1}}{\rightarrow}\overset{t_1}{\rightsquigarrow}\overset{\tau^{m_2}}{\rightarrow} \ldots \overset{\tau^{m_n}}{\rightarrow}\overset{t_n}{\rightsquigarrow} Q \\ &\quad \qquad \qquad \qquad \qquad \qquad \qquad \qquad \textit{and } t = t_1 + t_2 + \cdots + t_n \end{aligned}$$

We do wish to generalise probabilistic transitions but it turns out to be difficult to allow τ transitions in the middle. It certainly seems difficult to interpret non-deterministic choices within probabilistic 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 consider f_1 or f_2. So we make no changes to the probabilistic relations.

We define our fundamental equivalence as before using these modified relations

Definition 17 *An equivalence relation $\mathcal{R} \subset \mathsf{Proc} \times \mathsf{Proc}$ is a* weak bisimulation *iff $P \,\mathcal{R}\, Q$ implies for all $\delta \in \text{Act} - \{\tau\} \cup \{\varepsilon\}$, $t \in \mathbb{R}^+$, $S \in \overline{B}(\mathcal{R})$:*

1. *if $P \overset{\delta}{\Rightarrow} P'$, then $\exists Q'$ such that $Q \overset{\delta}{\Rightarrow} Q'$ and $P' \,\mathcal{R}\, Q'$.*
2. *if $Q \overset{\delta}{\Rightarrow} Q'$, then $\exists P'$ such that $P \overset{\delta}{\Rightarrow} P'$ and $P' \,\mathcal{R}\, Q'$.*
3. *if $P \overset{t}{\rightrightarrows} P'$, then $\exists Q'$ such that $Q \overset{t}{\rightrightarrows} Q'$ and $P' \,\mathcal{R}\, Q'$.*
4. *if $Q \overset{t}{\rightrightarrows} Q'$, then $\exists P'$ such that $P \overset{t}{\rightrightarrows} P'$ and $P' \,\mathcal{R}\, Q'$.*
5. *If P and Q are probabilistic then $\mu(P,S) = \mu(Q,S)$.*

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

Note that previously we had that any probabilistic process could not act and vice versa so that only probabilistic processes could be bisimilar to probabilistic processes. Now we can have $P \hookrightarrow$ and $P \overset{\varepsilon}{\Rightarrow}$ and thus such a process can be equated to one which can perform a τ (and not sample).

Proposition 7 $\approx$ *is the largest weak bisimulation*

Lemma 7

$$\sim \subseteq \approx$$

Now *approx* is not a congruence with respect to $+$ since *e.g.*

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

since in the latter equation the left hand side may $\stackrel{\varepsilon}{\Rightarrow} \beta.\mathsf{nil}$ while the right hand side may not move internally to any bisimilar term. The solution is to follow Milner and move to the associated congruence as our notion of equality.

Definition 18 *$P = Q$ iff $\forall$ contexts $C[-]$ $\quad C[P] \approx C[Q]$.*

In fact this takes us to the position of standard CCS with $\sim \subset = \subset \approx$ and $\sim$ and $=$ congruences. In particular, we have weak rules similar to [15].

Theorem 5

- $\tau.P + P = \tau.P$.
- $\mu.(P + \tau.Q) = \mu.(P + \tau.Q) + \mu.Q$.
- $\mu.\sum_i(t_i).\mu_i.P_i = \mu.\left(\sum(t_i).\mu_i.P_i + (t).\tau.\sum(t_i \dot{-} t).\mu_i.P_i\right)$

We also have the obvious rules for amalgamating nested probabilistic sums. We can also combine probabilistic sums with a final $\mathcal{R}[s \leftarrow f].P$ (*e.g.* $\sum[q_i]\,\mathcal{R}[s \leftarrow f_i].P = \mathcal{R}[s \leftarrow f].P$ for the appropriate distribution f.

But note that $\approx$ will not equate processes that can delay with processes that are probabilistic *e.g.* $\mathcal{R}[s \leftarrow f].\mathsf{nil}$ and nil.

6 Examples and speculations

6.1 GSMP

We show we can represent a GSMP in this formalism. We do not attempt to model variable clock rates.

The GSMP has states A, with an initial distribution $q : A \to [0,1]$. The clocks are in C and each state has associated clocks $E : A \to 2^C$. The initial clocks have a special distribution $F_0 : A \times C \to (\mathbb{R}^+ \to [0,1])$ while the general clocks have a distribution depending on both the previous and current states and events $F : A \times C \times A \times C \to (\mathbb{R}^+ \to [0,1])$. The choice of next state is controlled by the distribution $p : A \times C \to (A \to [0,1])$.

We represent this by 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).

It selects an initial state *via* the distribution q.

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

Along with this state it starts all associated clocks $E(a)$.

$$St0_a = St_a \mid \prod_{e \in E(a)} Clock0_{ae}$$

These inital clocks have a special distribution F_0.

$$Clock0_{ae} = \mathcal{R}[s \leftarrow F_0(a,e)].(s).\alpha_e + \overline{\beta_e}$$

Each state waits for any associated event,

$$St_a = \sum_{e \in E(a)} \overline{\alpha_e}.StEv_{ae}$$

then chooses its succeeding state a' via,

$$StEv_{ae} = \sum\!\!\!\!\bullet_{a' \in A} [p(a,e)(a')] StEvSt_{aea'}$$

then it cancels any outstanding and no longer required clocks.

$$StEvSt_{aea'} = (\bigodot_{e' \in E(a) \backslash \{e\} \backslash E(a')} \beta_{e'}).StEvSt1_{aea'}$$

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

$$StEvSt1_{aea'} = St_{a'} \mid \prod_{e' \in E(a') \backslash (E(a) \backslash \{e\})} Clock_{aea'e'}$$

These new clocks use their standard distribution.

$$Clock_{aea'e'} = \mathcal{R}[s \leftarrow F(a,e,a',e')].(s).\alpha_{e'} + \overline{\beta_{e'}}$$

Note that there are no fundamental problems with having a countably infinite number of states (though we could not then explicitly write all the expressions written above).

6.2 Repairman example

We model a machine shop by N machines each with a time to failure with a distribution $F(n)$. These may be modelled by:

$$M_n = \mathcal{R}[s \leftarrow F(n)].(s).\alpha_n.\overline{\beta_n}.M_n$$

We imagine R repairmen each of whom chooses (non-deterministically) to repair a defective machine. He takes a time $G(r,n)$ to do it. We model the repairmen by:

$$R_r = \sum_n \overline{\alpha_n}.\mathcal{R}[s \leftarrow G(r,n)].(s).\beta_n.R_r$$

We can then model the machine shop as a whole by:

$$Shop = \prod_n M_n \mid \prod_r R_r$$

This process exhibits non-determinism. It will certainly not be the same simulation regardless of the execution schedule chosen. For example, if the schedule preferentially services some machines then the performances of different machines will differ. But some measures are insensitive to the schedule (total machine availability, repairman workload *etc.*). We could detect this by inserting a probe action at the appropriate point (say in every machine) and then restricting over all other actions. We would expect the resulting simulations to be identical under all schedules.

6.3 Queueing

Consider a set of agents A_p indexed by some number p. We constrain the agents to say they are ready to start with an α_p and then to wait for a β_p (from the server) before they do.

$$A'_p = \alpha_p.\overline{\beta_p}.A_p$$

Then we queue the agents with a queue specified like this:

$$B_p = \gamma_p.\sum_p \overline{\alpha_p}.B_p$$

$$B_\infty = \sum_p \overline{\alpha_p}.(B_\infty[\delta/\gamma] \mid B_p[\delta/\alpha]) \setminus \delta$$

Note that here we use α *etc.* to represent α_p for all p.

Then the server accepts an instruction γ_p from the queue and issues a β_p to tell that agent it is ready.

$$S = \sum_p \overline{\gamma_p}.\beta_p.S(p)$$

We assume $S(p)$ becomes S again after dealing with the agent.

Placing these in parallel and restricting over $\{\alpha, \beta\}$ causes the agents to queue for the server. The queue is unbounded.

6.4 Periodic availability

Imagine a hard disc in which the requested sector is ready to be read once per revolution. We might try to model this as a server (simplifying our previous example) by

$$S = (1).(\overline{\gamma}.\beta.S + \tau.S)$$

Every time unit this server will either accept a request (and continue) or simply continue another revolution. The problem is that if a request is pending the server can non-deterministically ignore it and continue (since it becomes $\tau.\beta.S + \tau.S$ which may fail to generate a β). The point is that our model is physically unrealistic: the server is asked to instaneously assess whether a request is pending as the sector comes by, rather than during the period in which the sector approaches.

So our model should be:

$$S = \overline{\gamma}[s \leftarrow 0].(1 \dot{-} s).\beta.S + (1).\tau.S$$

If the γ action becomes available before the time unit is up the server becomes committed to act upon it. Otherwise it will time out and start again. It will still act non-deterministically if the γ arrives exactly at one time unit. This is rather similar to the case of GSMPs where it is usually disallowed by a condition on the distributions.

We may now construct a general model where the queueing agents are requests for specific sectors and the disc presents them in order.

$$S_n = \overline{\gamma_n}[s \leftarrow 0].(1 \dot{-} s).\beta_n.S_{n+1} + (1).\tau.S_{n+1}$$

($n+1$ is here taken modulo the number of sectors in the disc.)

$$A_n = \alpha_n.\overline{\beta_n}$$

$$R = \mathcal{R}[s \leftarrow f].(s).\sum[p_i]A_i.R$$

Here R generates requests randomly and we may use the queue B_∞ to queue them. So our system is $R \mid B_\infty \mid S_1$.

6.5 Equivalence

We consider a random walk on the integers:

$$W_0 = \mathcal{R}[s \leftarrow f].\epsilon_0.(s).\left([\tfrac{1}{2}]W_{n-1} \dot{+} [\tfrac{1}{2}]W_{n+1}\right)$$

and a single server queue (simplified) represented by:

$$B_0 = \epsilon_0.\overline{\alpha}[].B_1$$

$$B_n = \epsilon_n.(\overline{\alpha}[].B_{n+1} + \gamma[].B_{n+1}) \qquad n > 0$$

$$S = \mathcal{R}[s \leftarrow f].(s).\overline{\gamma}[].S$$

$$P = \mathcal{R}[s \leftarrow f].(s).(\alpha[] \mid P)$$

Here the ϵ_n 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 or a state in which the queue has just obtained n items. We then define

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

and would like to show $W_n = Q_n$. We assume f is a negative exponential distribution as usual.

$$\begin{aligned}
Q_n &= (B_n \mid S \mid P)\setminus\{\alpha,\beta,\gamma\}\\
&= (\epsilon_n.\overline{\alpha}[].B_{n+1} + \gamma[].B_{n-1} \mid \mathcal{R}[s \leftarrow f].(s).\overline{\gamma}[].S \mid \mathcal{R}[s \leftarrow f].(s).(\alpha[] \mid P))\setminus\{\alpha,\beta,\gamma\}\\
&= \mathcal{R}[s \leftarrow f].\mathcal{R}[s' \leftarrow f].\epsilon_n.(\overline{\alpha}[].B_{n+1} + \gamma[].B_{n-1} \mid (s).\overline{\gamma}[].S \mid (s').(\alpha[] \mid P))\setminus\{\alpha,\beta,\gamma\}\\
&= \mathcal{R}[s \leftarrow f].\mathcal{R}[s' \leftarrow f].\epsilon_n.((s).\tau.(B_{n-1} \mid S \mid (s \dot{-} s').(\alpha[] \mid P))\\
&+ \ (s').\tau.(B_{n+1} \mid (s' \dot{-} s).\overline{\gamma}[].S \mid P))\setminus\{\alpha,\beta,\gamma\}
\end{aligned}$$

Now we may use properties of the exponential distribution to rewrite this as:

$$\begin{aligned}
Q_n &= \mathcal{R}[s \leftarrow f].\left([\tfrac{1}{2}]\,\mathcal{R}[s' \leftarrow f].\epsilon_n.(s).\tau.(B_{n-1} \mid S \mid (s').(\alpha[] \mid P))\right.\\
&\dot{+}\ \left.[\tfrac{1}{2}]\,\mathcal{R}[s' \leftarrow f].\epsilon_n.(s).\tau.(B_{n+1} \mid (s').\overline{\gamma}[].S \mid P)\right)\setminus\{\alpha,\beta,\gamma\}
\end{aligned}$$

We may 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

$$\alpha.\sum^{\bullet}[q_i]Q_i = \sum^{\bullet}[q_i]\alpha.Q_i$$

This is not sound under bisimulation. We will assume instead that we are working under some equivalence where this axiom holds. Then we would have

$$\begin{array}{rcl}
Q_n & = & \mathcal{R}[s \leftarrow f].\epsilon_n.(s).\big([\tfrac{1}{2}]\,(B_{n-1} \mid S \mid \mathcal{R}[s' \leftarrow f](s').\,(\alpha[] \mid P)) \\
& \dot{+} & [\tfrac{1}{2}]\,(B_{n+1} \mid \mathcal{R}[s' \leftarrow f].(s').\overline{\gamma}[].S \mid P)\big) \setminus \{\alpha, \beta, \gamma\} \\
& = & \mathcal{R}[s \leftarrow f].\epsilon_n.(s).\big([\tfrac{1}{2}]\,(B_{n-1} \mid S \mid \mathcal{R}[s' \leftarrow f].(s').\,(\alpha[] \mid P)) \setminus \{\alpha, \beta, \gamma\} \\
& \dot{+} & [\tfrac{1}{2}]\,(B_{n+1} \mid \mathcal{R}[s' \leftarrow f].(s').\overline{\gamma}[].S \mid P) \setminus \{\alpha, \beta, \gamma\}\big) \\
& = & \mathcal{R}[s \leftarrow f].\epsilon_n.(s).\big([\tfrac{1}{2}]Q_{n-1} \dot{+} [\tfrac{1}{2}]Q_{n+1}\big)
\end{array}$$

as required.

To consider our missing axiom further, note that by repeated application we can generalise it to arbitrary length sequences of actions. Plainly it could never be sound for any recursive, bisimulation type of equivalence. In fact this axiom will only be sound in what is called a "linear" equivalence while bisimulation is a "branching" equivalence (see [12]). Work is proceeding on appropriate linear equivalences but their semantics are considerably more complex.

6.6 The mandatory example

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.

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

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

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

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

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

Then our initial presentation of the system is

$$System_1 = (Workload \mid Proc \mid Proc) backslash \{procreq, procrel\}$$

However if we examine the behaviour of $System_1$ we find it can do nothing (except $\overset{t}{\rightrightarrows}$) 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

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

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

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

for $n > 1$.

The relationship between this version and a Generalized Semi-Markov Process representation should be clear (see section 6.1). 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 Conclusions

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. However we have also presented an example showing that our semantics distinguishes certain simulations that are generally accepted as being equal.

A more complete description is in [14] where the complete language is presented with proofs of all results. Also a different semantics is presented in terms of Hennessy and de Nicola testing which overcomes the problems just noted.

References

[1] J.C.M. Baeten and J.A. Bergstra. Real time process algebra. Technical Report CS-R9053, Centrum voor Wiskunde en Informatica, October 1990.

[2] Liang Chen. An interleaving model for real-time systems. Technical Report ECS-LFCS-91-184, LFCS, University of Edinburgh, November 1991.

[3] Liang Chen, Stuart Anderson, and Faron Moller. A Timed Calculus of Communicating Systems. Technical report, University of Edinburgh, 1990.

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

[5] Hans A. Hansson. *Time and Probability in Formal Design of Distributed Systems*. PhD thesis, Uppsala University, P.O. Box 520, S-751 20 Uppsala, Sweden, 1991.

[6] Matthew Hennessy and Tim Regan. A process algebra for timed systems. Technical report, Sussex University, April 1991.

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

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

[9] Faron Moller and Chris Tofts. A Temporal Calculus of Communicating Systems. In *Concur '90*, number 458 in LNCS, 1990.

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

[11] Xavier Nicollin and Joseph Sifakis. An overview and synthesis on timed process algebras. In Kim G. Larsen and Arne Skou, editors, *Proc. CAV 91*, number 575 in LNCS, pages 376–398, 1991.

[12] 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*, volume 194 of *LNCS*, 1985.

[13] Steve Schneider. An operational semantics for timed CSP. Technical report, Programming Research Group, Oxford University, February 1991.

[14] Ben Strulo. *Process Algebra for Discrete Event Simulation.* PhD thesis, to appear, 1993.

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

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