Formal Aspects of Computing (1993) 3: 1–000

Processes with Probabilities, Priority and Time.

Chris Tofts,

Dept. of Computer Science,
University College Swansea,
Swansea, SA2 8PP.
Email:c.m.n.Tofts@swansea.ac.uk

Abstract. A probabilistic calculus, derived from Milner's SCCS, WSCCS is presented. We define a notion of bisimulation for probabilistic processes and show that it is a congruence. A simple equational characterisation is shown to be both sound and complete for finite processes. We present many examples including some extended ones. The larger examples show both the expressive power of WSCCS and the availability of simple proof methods for some complex systems.

1. Introduction.

Process algebras [Mil83, BBK86, Mil90] depend upon concepts of non-determinism, or choice; however the biases in the resolution of such choices are usually left unspecified. It is natural to consider extending such systems by adding a probabilistic quantification for non-determinism. Recently there has been much interest in the properties and representation of concurrent probabilistic systems [CJS90, LS89, GSST90, SST89, Jon90, JS90, Tof90a, SS90, Chr90] and in probabilistic sequential systems [Wel83, Rab76]. One reason for this interest is that probabilistic methods of computation often offer more efficient and more fault tolerant solutions than conventional deterministic methods. A probabilistic system will often cope with the presence of faults simply as a consequence of its underlying

Correspondence and offprint requests to: Dept. of Computer Science,
University College Swansea,
Swansea, SA2 8PP.
Email:c.m.n.Tofts@swansea.ac.uk

algorithm [Tof90b, Tof91], avoiding the need for modification to handle fault generating situations.

In this paper we shall give an account of a calculus relying upon a notion of *relative frequency*, as this is formally simpler than the direct description of probability. In particular we will be looking at the choice operator of our calculus. Conventionally, we write a choice as

$$A \ + \ B$$

and interpret it as a process that has the capabilities of both the processes A and B. This choice is underspecified. In an unbiased environment - one which lets the process decide which choice to take - we could see behaviours ranging from infinite selection of one process over the other to some fixed ratio of choice between the processes.

The theory of Markov [Kei] and stochastic systems is rich, and includes many general results. As a consequence of our processes being probabilistic in nature, they can be realised within these frameworks and existing theories exploited to prove properties of our processes. Indeed the later examples of this paper exploit some simple Markov theories, which enable us to prove properties of some highly complex (in terms of their number of states) systems.

Our calculus is synchronous, since we shall be quantifying the relative frequency of free simultaneous choice. In an asynchronous system choices may be resolved at arbitrary times, thus giving a choice which may not be between equally free objects. This calculus simplifies the probabilistic system of [SST89,GSST90] and is based upon Milner's [Mil83] *synchronous calculus of communicating systems.* We choose not to quantify our choices directly with probabilities but to use *weights* which we can interpret as probabilistic specification via the concept of relative frequency; our notion of equality will allow us to make that interpretation. Our intention is that processes such as the following will be identified:

$$1:P \ + \ 2:Q \text{ and } 3:P \ + \ 6:Q.$$

That is to say that we shall see 1 occurrence of P for every 2 of Q in the left hand process, and shall see 3 occurrences of P to every 6 of Q in the right hand process. Since the relative frequency of P and Q is the same in both systems we would like to equate these process descriptions.

Whilst a calculus built from natural weights may at first seem unnatural as a way of expressing probabilities, we shall consider the calculus as describing rational probabilities, which we have forced into naturals by multiplication by a suitable factor. We shall see later that multiplication of weights by a constant factor is conserved by our interpretation of WSCCS processes.

We exploit some special weights, which will be denoted by the symbols,

$$\omega, \omega^2, \ldots, \omega^n, \ldots$$

to express a notion of priority. We shall introduce an operator, similar to that of [BBK86], which will extract the highest priority part of a process. However, unlike the earlier work, our priorities can be dynamic. The relative priority of two actions is given by the syntax of the process and thus does not have to remain fixed over the execution of the process. The use of powers of priority gives us a richer notion of priority than that given in [Tof90a, SS90], in particular we are able to address the question of *maximal progress*, that is to form our processes in such as way that they will compute whenever their environment will permit them to. Whilst the work of [Cam89] has a dynamic notion of priority (within CCS);

it requires that the operational semantics is parameterised by environments, our semantics avoids this problem.

Weights are operationally more convenient than probabilities since we do not have to ensure that they normalise to any particular value. This normalisation problem is the cause of the complexity of the derivation law for permission in probabilistic systems. Using weights the permission rule is literally the pruning of sub-processes that will never execute as none of their actions are permitted.

Since our calculus is synchronous we have a simple notion of time. Whilst this is not as rich as many of the timed systems now available [RR86, Tof89, MT90, HR90, Yi90, CAM90 and many others], it is sufficiently expressive to allows us to observe simple temporal properties of our processes, much as we can with SCCS [Mil83].

As in [SST89, GSST90, Tof89, Tof90a] we shall work with a multi-coloured transition system; that is to say we have more than one notion of evolution. We intend to use two different mechanisms of state change, either by a weighted choice, or by the performance of a named event. This description method is used as it makes the underlying mechanism of state change clear, and we obtain a rich system which is capable of expressing most of the processes we wish to study. Consequently our model is stratified [GSST90] which is suspected to be the most abstract possible for such systems.

We proceed by defining our calculus WSCCS, *the weighted synchronous calculus of communicating systems.* Its syntax and some motivating examples of processes are presented, followed by the language's operational semantics. We define a notion of process equivalence based on matching the **absolute** frequency with which equivalent states can occur, and demonstrate that it is a congruence. Subsequently, it is straightforward to define an equivalence based on the **relative** frequency of occurrence of equivalent states, which is indeed a notion of probabilistic equivalence.

From our notions of process equivalence we have a simple equational characterisation of WSCCS processes and this is presented. We show that it is both sound and complete for finite processes. We conclude the technical introduction to the language WSCCS with a description of some uses of priority.

Subsequently we present four extended examples, intended to demonstrate the usefulness of the calculus and its simplicity, these are an eager buffer, a first come first served resource allocation system, a self-synchronising system and a probabilistic task allocation system.

The eager buffer is a buffer in which the data is moved as far in the direction of its destination as it is possible to do so, using empty cells. Thus mimicing a physical buffers usual data availability performance. We demonstrate that such a buffer can be constructed from simple cells within our language.

The first come first served allocation system, is one in which there is a single resource, which may be claimed by many competing users. The resource should be allocated to the user that has been waiting the longest. We demonstrate that by using our dynamic priorities such an allocating strategy can be coded *without* auxiliary processes or data structures.

The autosynchronisation algorithm is a method for synchronising a large number of distributed systems. The system is uniform, that is all of its components are identical. We demonstrate that such a synchronisation methodology is tolerant to component failure, as well as local clock errors. We exploit a powerful general proof method to demonstrate the properties of our synchronisation system. This example exploits our ability to code broadcast actions within WSCCS.

Finally, we present a probabilistic task system, which also exploits the Markov chain decomposition theorem to demonstrate its correctness. Our parallel probabilistic task allocation system is shown to be both dynamic, that is changing allocations in response to work load changes, and tolerant of failures of two different kinds. These last two examples are substantive and have been used to explain some Biological behavioural problems. Far more extensive discussions of the properties of these two systems can be found in [Tof90b] and [Tof91] respectively.

2. The Language WSCCS.

Our language WSCCS is an extension of Milner's SCCS [Mil83], a language for describing synchronous concurrent systems. To define our language we presuppose an abelian group *Act* of atomic action symbols with identity $\surd$ and the inverse of a being $\overline{a}$. As in SCCS, the complements a and $\overline{a}$ form the basis of communication. Normally multiplication of actions will be represented by juxtaposition, but when clarity demands we shall use the symbol # to represent action composition. We also take a set of weights $\mathcal{W}$, denoted by w_i, which are of the form $n\omega^k$ with n from the positive natural numbers $\mathcal{P}$ and the ω^k (with $k \geq 0$) a set of infinite objects, with the following multiplication and addition rules assuming $k > k'$:

$$n\omega^k + m\omega^{k'} = n\omega^k = m\omega^{k'} + n\omega^k \qquad n\omega^k + m\omega^k = (n+m)\omega^k = m\omega^k + n\omega^k$$
$$n\omega^k * m\omega^{k'} = (nm)\omega^{k+k'} = m\omega^{k'} * n\omega^k$$

We consider the objects n used as weights to be abbreviations for $n\omega^0$. It is straightforward to demonstrate that our notions of multiplication and addition on weights are both commutative and associative, and that weight multiplication distributes over addition. Finally, that projection ($\mathcal{N}(n\omega^k) = n$) distributes over both addition of equal priority weights and multiplication. Finally we assume a set of process variables Var.

The collection of WSCCS expressions ranged over by E is defined by the following BNF expression, where $a \in Act$, $X \in Var$, $w_i \in \mathcal{W}$, S ranging over renaming function (those $S : Act \longrightarrow Act$ such that $S(\surd) = \surd$ and $\overline{S(a)} = S(\overline{a})$), action sets $A \subseteq Act$ (with $\surd \in A$), and arbitrary *finite* indexing sets I:

$$E ::= X \mid a.E \mid \sum\{w_i : E_i | i \in I\} \mid E \times E \mid E\lceil A \mid \Theta(E) \mid E[S] \mid \mu_i\tilde{x}\tilde{E}.$$

We let Pr denote the set of closed expressions, and add 0 to our syntax, which is defined by $0 \stackrel{def}{=} \sum\{w_i E_i | i \in \emptyset\}$.

The informal interpretation of our operators is as follows:

- 0 a process which cannot proceed;
- X the process bound to the variable X;
- $a.E$ a process which can perform the action a whereby becoming the process described by E;
- $\sum\{w_i : E_i | i \in I\}$ the *weighted* choice between the processes E_i, the weight of the outcome E_i being determined by w_i. We think in terms of repeated

experiments on this process and we expect to see over a large number of experiments the process E_i being chosen with a relative frequency of $\frac{w_i}{\sum_{i\in I} w_i}$.

- $E \times F$ the synchronous parallel composition of the two processes E and F. At each step each process must perform an action, the composition performing the composition (in *Act*) of the individual actions;
- $E\lceil A$ represents a process where we only permit actions in the set A. This operator is used to enforce communication and bound the scope of actions;
- $\Theta(E)$ represents taking the prioritised parts of the process E only.
- $E[S]$ represents the process E relabelled by the function S;
- $\mu_i \tilde{x} \tilde{E}$ represents the solution x_i taken from solutions to the mutually recursive equations $\tilde{x} = \tilde{E}$.

Often we shall omit the dot when applying prefix operators; also we drop trailing 0, and will use a binary plus instead of the two (or more) element indexed sum, thus writing $\sum\{5_1 : a.0, \quad 6_2 : b.0 | i \in \{1,2\}\}$ as $5 : a \ + \ 6 : b$. Finally we allow ourselves to specify processes definitionally, by providing recursive definitions of processes. For example, we write $A \stackrel{def}{=} a.A$ rather than $\mu x.ax$.

2.1. Simple Examples

We present some simple examples of processes definable within the WSCCS syntax as a motivation for the operational semantics that follows.

2.1.1. Gambler's Ruin.

We take the actions 1, 2, 3, 4, 5, 6 and describe a fair die as the following process:

$$Die \stackrel{def}{=} 1 : \overline{1}.Die + 1 : \overline{2}.Die + 1 : \overline{3}.Die + 1 : \overline{4}.Die \ 1 : \overline{5}.Die + 1 : \overline{6}.Die,$$

and a gambler with an amount of money n, who wins 6 units of money when a six is thrown and loses his stake (of size one) on any other outcome as the following process (we add the actions *win* and *lose*):

$$\begin{aligned}
Gamb(0) &\stackrel{def}{=} 0 \\
Gamb(n) &\stackrel{def}{=} 1 : 1lose.Gamb(n-1) + 1 : 2lose.Gamb(n-1) \\
&\quad\; 1 : 3lose.Gamb(n-1) + 1 : 4lose.Gamb(n-1) \\
&\quad\; 1 : 5lose.Gamb(n-1) + 1 : 6win.Gamb(n+6)
\end{aligned}$$

The process

$$Game(n) \stackrel{def}{=} (Gamb(n) \times Die)\lceil\{\surd, \ win, \ lose\}$$

describes a betting game in which the gambler starts with n units of money. We can use this description to find the Markov system that the above process describes and we could predicate such parameters as the expected time until our gambler is bankrupt. We can make our die unfair by changing the weighting of the various outcomes, such an unfair die can be described as,

$Unf \stackrel{def}{=} 2:1.Unf + 1:2.Unf + 1:3.Unf + 1:4.Unf + 2:5.Unf + 1:6.Unf;$

this die only produces a six one time in eight. If the process Unf is used in place of the Die process in the game, our gambler would run out of money sooner.

If the group Act has the structure that the operation on number actions is to add them. For example a synchronous action of 2#3 being the same as the action 5. Then the process

$$Die \times Die$$

will describe rolling two dice and adding their outcomes. Notice, if we had a free multiplicative structure on the actions, then we would need a further process to perform the addition of the outcomes of the individual die roles.

2.1.2. Spontaneous Failure.

Most systems are subject to spontaneous failure due to hardware faults or disruption of electricity supply; but most descriptions of systems assume that they do not have behaviour of this nature. The process

$$Fail \stackrel{def}{=} m : \surd.Fail + 1 : fail.0$$

describes a system that emits the action *fail* with weight 1 at each turn and then becomes the nil process. If we take an idealised process M describing a system without spontaneous failure then the behaviour of the process

$$M \times Fail$$

will be that of M except on a *fail* action where the process will subsequently stop. Note that there is no communication structure allowing M to prevent the occurrence of a fail or even influence its likelihood.

2.1.3. Competing Priorities

One problem in giving an account of priority is when two processes wishing to communicate have incompatible priorities. Such a system could be presented as:

$P \stackrel{def}{=} \omega : a.R_1 + 1 : b.S_1$
$Q \stackrel{def}{=} \omega : \overline{b}.S_2 + 1 : \overline{a}.R_2$
$Sys \stackrel{def}{=} (P \times Q)\lceil(Act - \{a, b\})$

In the above P wishes to give a priority over b, whilst Q gives priority to $\overline{b}$ over $\overline{a}$, so their is a degree of conflict. Some authors [Cam90] forbid such constructs. In our interpretation the above system Sys will prove equal to:

$$\omega : \surd.(R_1 \times R_2)\lceil(A - \{a, b\}) + \omega : \surd.(S_1 \times S_2)\lceil(A - \{a, b\})$$

which can be interpreted as we have the *same* probability of arriving in either permitted state. This would seem a natural account of such competition.

2.2. The Semantics of WSCCS.

In this section we define the operational semantics of WSCCS. The semantics is transition based, structurally presented in the style of [Plo81], defining the actions that a process can perform and the weight with which a state can be reached. We shall define two forms of transition, the first will witness the performance of an event from the group *Act*; the second will witness the performance of some choice with a particular weight. Unfortunately for our second form of transition a relation is too weak, since we do not wish to identify the 1 transitions in $1 : P + 1 : P + 2 : Q$. Such an identification would not permit us to interpret the weights as probabilities. Consequently we must retain both of the 1 transitions in the above process, as a result we have a multirelation between states, weights and their descendants. Since we do not want unnecessary replication of transitions we must take the *least* multirelation consistent with the derivation system.

In Figure 1 we present the operational rules of WSCCS. They are presented in a natural deduction style. The transitional semantics of WSCCS is given by the least relation $\longrightarrow \subseteq WSCCS \times Act \times WSCCS$, the least multi-relation $\longmapsto \subseteq bag(WSCCS \times \mathcal{W} \times WSCCS)$ [1], which are written $E \xrightarrow{a} F$ and $E \overset{w}{\longmapsto} F$ respectively, and the least relation $does_A(E)$ satisfying the rules laid out in Figure 1. These rules respect the informal description of the operators given earlier.

The predicate $does_A(E)$ defines those states which can potentially perform an action from the set A starting from the state E. It is well defined since we have only permitted finitely branching choice expressions. This operationally defined predicate is used to produce the behaviour of the permission operator, which is to prune from the choice tree those processes that can no longer perform any action. As an example consider:

$$P \overset{def}{=} 1 : a + 2 : (3 : b + 4 : c)$$

if we take $P\lceil\{a, b\}$, then we need to retain both the 1 and the 2 transitions from P as the process could perform as action from $3 : b + 4 : c$. However, if we took $P\lceil\{a\}$ then we *must* lose the 2 transition as none of its descendants are permitted to perform any activity

2.2.1. Direct Bisimulation.

Our bisimulations will be based on the accumulation technique of Larsen and Skou [LS89]. We start by defining accumulations of evolutions for both types of transition.

Definition 2.1. Let S be a set of processes then:

- $P \overset{w}{\longmapsto} S$ with $w = \sum\{w_i | P \overset{w_i}{\longmapsto} Q \text{ for some } Q \in S\}$; [2]
- $P \xrightarrow{a} S$ iff there exists $Q \in S$ and $P \xrightarrow{a} Q$.

We define a form of bisimulation that identifies two processes if the total weight of evolving into any equivalent states is the same. This is not quite the identification we wish to make, but we will make such an identification later.

[1] Where $bag(WSCCS \times \mathcal{W} \times WSCCS)$ is the bag whose elements are those of the set $WSCCS \times \mathcal{W} \times WSCCS$, with the usual notion of bag.

[2] Remembering this is a multi-relation so some of the Q and w_i may be the same process and value. We take all occurrences of processes in S and add together all the weight arrows leading to them.

$$\overline{a.E \xrightarrow{a} E} \qquad \overline{\sum\{w_i E_i | i \in I\} \stackrel{w_i}{\longmapsto} E_i}$$

$$\frac{E \xrightarrow{a} E' \quad F \xrightarrow{b} F'}{E \times F \xrightarrow{ab} E' \times F'} \qquad \frac{E \stackrel{w}{\longmapsto} E' \quad F \stackrel{v}{\longmapsto} F'}{E \times F \stackrel{wv}{\longmapsto} E' \times F'}$$

$$\frac{E \xrightarrow{a} E' \quad F \stackrel{w}{\longmapsto} F'}{E \times F \stackrel{w}{\longmapsto} E \times F'} \qquad \frac{E \stackrel{w}{\longmapsto} E' \quad F \xrightarrow{a} F'}{E \times F \stackrel{w}{\longmapsto} E' \times F}$$

$$\frac{E \xrightarrow{a} E' \quad a \in A}{does_A(E)} \qquad \frac{E \stackrel{w}{\longmapsto} E' \quad does_A(E')}{does_A(E)}$$

$$\frac{E \xrightarrow{a} E' \quad a \in A}{E \lceil A \xrightarrow{a} E' \lceil A} \qquad \frac{E \stackrel{w}{\longmapsto} E' \quad does_A(E')}{E \lceil A \stackrel{w}{\longmapsto} E' \lceil A}$$

$$\frac{E \xrightarrow{a} E'}{E[S] \xrightarrow{S(a)} E'[S]} \qquad \frac{E \stackrel{w}{\longmapsto} E'}{E[S] \stackrel{w}{\longmapsto} E'[S]}$$

$$\frac{E_i\{\mu_i \tilde{x}.[\tilde{E}/\tilde{x}]\} \xrightarrow{a} E'}{\mu_i \tilde{x}.\tilde{E} \xrightarrow{a} E'} \qquad \frac{E_i\{\mu \tilde{x}.[\tilde{E}/\tilde{x}]\} \stackrel{w}{\longmapsto} E'}{\mu \tilde{x}.\tilde{E} \stackrel{w}{\longmapsto} E'}$$

$$\frac{E \xrightarrow{a} E'}{\Theta(E) \xrightarrow{a} \Theta(E')} \qquad \frac{E \stackrel{n\omega^k}{\longmapsto} E' \not\exists (k' > k). E \stackrel{m\omega^{k'}}{\longmapsto}}{\Theta(E) \stackrel{n}{\longmapsto} \Theta(E')}$$

Fig. 1. Operational Rules for WSCCS.

Definition 2.2. An equivalence relation $R \subseteq Pr \times Pr$[3] is a *direct bisimulation* if $(P, Q) \in R$ implies for all $S \in Pr/R$ that:

- for all $w \in \mathcal{W}$, $P \stackrel{w}{\longmapsto} S$ iff $Q \stackrel{w}{\longmapsto} S$;
- for all $a \in Act$, $P \xrightarrow{a} S$ iff $Q \xrightarrow{a} S$.

Two processes are *direct bisimulation equivalent*, written $P \stackrel{d}{\sim} Q$, if there exists a direct bisimulation R between them.

Definition 2.3.

$$\stackrel{d}{\sim} \equiv \bigcup\{R \mid R \text{ is a direct bisimulation }\}.$$

That $\stackrel{d}{\sim}$ is a direct bisimulation follows immediately from it being a union of direct bisimulations.

[3] We denote the equivalence class of a process P with respect to R by $[P]_R$. When it is clear from the context to which equivalence we are referring, we will omit the subscript.

Lemma 2.4. Let P and Q be processes such that $P \stackrel{d}{\sim} Q$. Then for all action sets A, $does_A(P)$ iff $does_A(Q)$.

Proposition 2.5. Direct equivalence is substitutive for finite processes. Thus, given $P \stackrel{d}{\sim} Q$ and $P_i \stackrel{d}{\sim} Q_i$ for all $i \in I$ then:

1. $a.P \stackrel{d}{\sim} a.Q$;	2. $\Sigma_{i\in I} w_i : P_i \stackrel{d}{\sim} \Sigma_{i\in I} w_i : Q_i$;
3. $P \times E \stackrel{d}{\sim} Q \times E$;	4. $P\lceil A \stackrel{d}{\sim} Q\lceil A$;
5. $P[S] \stackrel{d}{\sim} Q[S]$;	6. $\Theta(P) \stackrel{d}{\sim} \Theta(Q)$.

Proof. (After [Mil89, GSST90]) We proceed by induction over the structure of the process and the size of the process:

1. $a.P \stackrel{d}{\sim} a.Q$ immediate;
2. $\Sigma_{i\in I} w_i : P_i \stackrel{d}{\sim} \Sigma_{i\in I} w_i : Q_i$ immediate;
3. $P \times E \stackrel{d}{\sim} Q \times E$, we construct the equivalence relation $R = \{(P \times E, Q \times E) \mid P \stackrel{d}{\sim} Q\} \cup \{(P \times E, E \times Q) \mid P \stackrel{d}{\sim} Q\}$. So consider the weight evolutions of the two processes,

$$\begin{array}{l} \text{if } P \times E \xmapsto{w} [P' \times E'] \text{ then} \\ \qquad P \xmapsto{w_1} [P'] \text{ and } E \xmapsto{v_1} [E'] \text{ also (possibly)} \\ \qquad P \xmapsto{w_2} [E'] \text{ and } E \xmapsto{v_2} [P'] \\ \qquad\qquad \text{with } w = w_1v_1 + w_2v_2, \text{ hence} \\ \qquad Q \xmapsto{w_1} [P'] \text{ and} \\ \qquad Q \xmapsto{w_2} [E'], \text{ thus} \\ Q \times E \xmapsto{w} [Q' \times E'] = [P' \times E']. \end{array}$$

   Likewise for the action evolutions,

$$\begin{array}{l} \text{if } P \times E \xrightarrow{a} [P' \times E'] \text{ then} \\ \qquad P \xrightarrow{b} [P'] \text{ and } E \xrightarrow{c} [E'] \\ \qquad\qquad \text{with } a = bc, \\ \qquad \text{thus } Q \xrightarrow{b} [P'] \text{ hence} \\ Q \times E \xrightarrow{a} [Q' \times E'] = [P' \times E']. \end{array}$$

   The symmetric cases follow immediately by the same proof;
4. $P\lceil A \stackrel{d}{\sim} Q\lceil A$ we define $R \equiv \{(P\lceil A, Q\lceil A) \mid P \stackrel{d}{\sim} Q\}$. Now consider the weight evolutions of the processes,

$$\begin{array}{l} \text{if } P\lceil A \xmapsto{w} [P'\lceil A] \text{ then} \\ \qquad P \xmapsto{w} [P'], \text{ and hence} \\ \qquad Q \xmapsto{w} [P'], \text{ thus} \\ Q\lceil A \xmapsto{w} [Q'\lceil A]. \\ \qquad \text{From lemma 2.4 as } does_A(Q') \text{ iff } does_A(P') \text{ since } Q' \stackrel{d}{\sim} P', \\ \text{so } [Q'\lceil A] = [P'\lceil A] \text{ as required.} \end{array}$$

   Similarly for the action evolutions

$$\begin{aligned}\text{if } P\lceil A &\xrightarrow{a} [P'\lceil A] \text{ with } a \in A \text{ then}\\ P &\xrightarrow{a} [P'], \text{ and hence}\\ Q &\xrightarrow{a} [P'], \text{ thus}\\ Q\lceil A &\overset{w}{\longmapsto} [Q'\lceil A] = [P'\lceil A];\end{aligned}$$

5. $P[S] \overset{d}{\sim} Q[S]$ immediate.
6. $\Theta(P) \overset{d}{\sim} \Theta(Q)$ follows from P and Q matching on the highest power of ω they can perform, and subsequently match at all stages.

□

We proceed by the usual technique of pointwise extension to define our equivalence for finite state processes.

Definition 2.6. Let $\tilde{E}$ and $\tilde{F}$ be expressions containing variables at most $\tilde{X}$. Then we will say $\tilde{E} \overset{d}{\sim} \tilde{F}$ if for all process sets $\tilde{P}$, $\tilde{E}\{\tilde{P}/\tilde{X}\} \overset{d}{\sim} \tilde{F}\{\tilde{P}/\tilde{X}\}$.

Proposition 2.7. If $\tilde{E} \overset{d}{\sim} \tilde{F}$ then $\mu_i\tilde{X}.\tilde{E} \overset{d}{\sim} \mu_i\tilde{X}.\tilde{F}$.

2.2.2. Relative Bisimulation.

Unfortunately, the congruence given by direct bisimulation is too strong; it does not capture our notion of relative frequency, but captures total frequency. Since we would like to be able to equate processes such as,

$$2:P \;+\; 3:Q \text{ and } 4:P \;+\; 6:Q,$$

we need to weaken our notion of equality. The basic idea is that in order to show two processes equivalent, for each pair of equivalent states we can choose a *constant* factor such that the total weight of equivalent immediate derivatives is related by multiplication by that factor. If we can do this for all potentially equivalent states then we will say that the processes are the same in terms of relative frequency. Since the constant factor may well need to be a rational (and we wish to keep our numbers as simple as possible) we will actually use two constants in comparing relative frequency. This allows us to use a symmetrical definition.

Definition 2.8. We say an equivalence relation $R \subseteq Pr \times Pr$ is a *relative bisimulation* if $(P,Q) \in R$ implies that:

1. there are $c_1, c_2 \in \mathcal{P}$ such that for all $S \in Pr/R$ and for all $w, v \in \mathcal{W}$, $P \overset{w}{\longmapsto} S$ iff $Q \overset{v}{\longmapsto} S$ and $c_1 w = c_2 v$;
2. for all $S \in Pr/R$ and for all $a \in Act$, $P \xrightarrow{a} S$ iff $Q \xrightarrow{a} S$.

Two processes are *relative bisimulation equivalent*, written $P \overset{r}{\sim} Q$ if there exists a relative bisimulation R between them.

We have chosen to use multiplication by a constant rather than division as this permits us to stay within the natural numbers. We could have normalised so that the total weight actions of any state is 1, and then we would have had an equivalence that is identical to that of stratified bisimulation [SST89,GSST90].

Definition 2.9.

$$\overset{r}{\sim} \equiv \bigcup\{R \mid R \text{ is a relative bisimulation}\}.$$

Proposition 2.10. Let P and Q be processes such that $P \overset{d}{\sim} Q$, then $P \overset{r}{\sim} Q$.

Definition 2.11. Let $\tilde{E}$ and $\tilde{F}$ be expressions containing variables at most $\tilde{X}$. Then we will say $\tilde{E} \overset{r}{\sim} \tilde{F}$ if for all process sets $\tilde{P}$, $\tilde{E}\{\tilde{P}/\tilde{X}\} \overset{r}{\sim} \tilde{F}\{\tilde{P}/\tilde{X}\}$.

Proposition 2.12. $\overset{r}{\sim}$ is a congruence for finite and finite state processes.

2.3. Abstract Bisimulation

We would like a notion of equivalence that permits us to disregard the structure of the choices and just look at the total chance of reaching any particular state. This is known *not* to produce a congruence: [GSST90], but is a useful notion of equivalence. Obviously this cannot be well defined for processes with unresolved priority weight transitions.

Definition 2.13. We define the probability of a transition: $P \overset{p}{\longmapsto} P'$, if $P \overset{w}{\longmapsto} P'$ and the total weight of transitions from P is w', with $p = \frac{w}{w'}$.

Definition 2.14. We define an abstract notion of evolution as follows;

$P \overset{a[p]}{\longrightarrow} P'$ iff $P \overset{p_1}{\longmapsto} \ldots \overset{p_n}{\longmapsto} \overset{a}{\longrightarrow} P'$ with $p = \prod p_i$.

In order to define an equivalence which uses such transitions we need a notion of accumulation.

Definition 2.15. Let S be a set of processes then:

$P \overset{a[p]}{\longrightarrow} S$ iff $p = \sum\{p_i | P \overset{a[p_i]}{\longrightarrow} Q$ for some $Q \in S\}$; [4]

We can now define an equivalence that ignores the choice structure but not the choice values.

Definition 2.16. We say an equivalence relation $R \subseteq Pr \times Pr$ is an *abstract bisimulation* if $(P, Q) \in R$ implies that:

for all $S \in Pr/R$ and for all $p \in [0,1]$, $P \overset{a[p]}{\longrightarrow} S$ iff $Q \overset{a[p]}{\longrightarrow} S$.

Two processes are *abstract bisimulation equivalent*, written $P \overset{a}{\sim} Q$ if there exists a abstract bisimulation R between them.

This will not be substitutive as the following counter example demonstrates. Consider the process:

$$(1 : a + 1 : (1 : b + 2 : c))\lceil\{a, b\}$$

we can demonstrate by abstracting the inner part that it is the same as:

$(3 : a + 1 : b + 2 : c)\lceil\{a, b\} \overset{a}{\sim} 3 : a + 1 : b$

[4] Remembering this is a multi-relation so some of the Q and p_i may be the same process and value. We take all occurrences of processes in S and add together all the weight arrows leading to them.

$$(\Sigma_1)\ \Sigma_{i\in I} w_i : E_i = \Sigma_{j\in J} v_j : E_j \quad \begin{cases} \text{there is a surjection } f : I \longmapsto J \text{ with} \\ v_j = \Sigma\{w_i | i \in I \wedge f(i) = j\}, \\ \text{and for all } i \text{ with } f(i) = j \text{ then } E_i = E_j. \end{cases}$$

$$(Exp_1)\ \ a.E \times b.F = ab.(E \times F)$$
$$(Exp_2)\ \ a.E \times \Sigma_{j\in J} v_j : F_j = \Sigma_{j\in J} v_j : (a.E \times F_j)$$

$$(Exp_3)\ \ P \times Q = Q \times P$$
$$(Exp_4)\ \ (\Sigma_{i\in I} w_i : E_i) \times (\Sigma_{j\in J} v_j : F_j) = \Sigma_{(i,j)\in(I\times J)} v_i w_j : (E_i \times F_j)$$

$$(Res_1)\ \ (a.E)\lceil A = \begin{cases} a.(E\lceil A) \text{ if } a \in A \\ 0 \text{ otherwise.} \end{cases}$$

$$(Res_2)\ \ (\Sigma_{i\in I} w_i : E_i)\lceil A = \Sigma_{j\in J} w_j : (E_j \lceil A) \text{ where } J = \{i \in I | d_A(E_i)\}$$

$$(\Theta_1)\ \ \Theta(a.E) = a.\Theta(E)$$

$$(\Theta_2)\ \ \Theta(\Sigma_{i\in I} w_i : E_i) = \Sigma_{j\in J} \mathcal{N}(w_j).\Theta(E_j) \text{ where } J = \{i \in I | \exists n \text{ and } w_i = n\omega^{max_\omega(\{w_i\})}\}$$

$$(Ren)\ \ \Sigma_{i\in I} w_i : E_i = \Sigma_{i\in I} n w_i : E_i \text{ where } n \in \mathcal{P}$$

Fig. 2. Equational rules for WSCCS.

but performing the permission first and then equating is:

$$(1 : a + 1 : (1 : b)) \overset{a}{\sim} 1 : a + 1 : b$$

which is not the same. However, this relation can easily be shown to be an equivalence. It is included as it is the formalisation of the translation of WSCCS processes into Markov systems. That is, for each state distinguished with this form of transition we have a state of a Markov system and the transition probabilities are those given by the probabilities.

2.4. Equational Characterisation of WSCCS.

We present some equational laws over WSCCS processes in Figure 2, these form a sound and complete equational system over the finite processes in WSCCS [Tof90a]. We shall write $p = q$ for $p \overset{r}{\sim} q$.

Definition 2.17. Let A be an action set then the syntactic predicate, $d_A(E)$, expressing the fact that E can perform an action in A, is defined recursively as follows:

- $d_A(a.E)$ iff $a \in A$;
- If there exists $i \in I$ with $d_A(E_i)$ then $d_A(\Sigma_{i\in I} w_i : E_i)$.

Note, that whilst $d_A(E)$ performs the same role, we cannot use the predicate $does_A(E)$in our equational theory as it is defined over the operational semantics of our processes.

Definition 2.18. Let W be a set of weights $\{w_i\}$ then $max_\omega(W)$ is the maximum power of ω occurring in W.

Definition 2.19. We define a projection from weights as follows,

$$\mathcal{N}(m\omega^k) \stackrel{def}{=} m\omega^0.$$

The demonstration of the completeness of these equations for finite state processes follows from an obvious reduction of processes to one of two normal forms, either:

$a.P$ or
$\Sigma_{i \in I} w_i : P_i$

where P and the P_i are themselves in normal form. See [Tof90a] for a fuller account.

2.5. Exploiting Priority.

Our calculus contains an extremely expressive notion of priority which can be used to express many standard forms of process. We can factorise processes by exploiting the following distribution equation:

$$\Theta(P \times Q) = \Theta(P) \times \Theta(Q)$$

this holds since the projection function $\mathcal{N}$ on weights distributes over multiplication of weights, and over addition of equal priority weights. This distribution law demonstrates that the addition of priority to our weights has not disturbed the simultaneous performance of experiments when they are independent. It is simple to demonstrate that priority cannot distribute over restriction, indeed it is clear that it should not do so.

This factorisation allows the sensible definition of two different notions of asynchronous behaviour within our calculus. The first, which is analogous to asynchrony within CCS [Mil83] is to use simple non-determinism:

$\$P \stackrel{def}{=} 1 : P + 1 : \surd.\$P.$

The second exploits priority and gives us a different notion of asynchrony:

$!P \stackrel{def}{=} \omega : P + 1 : \surd.!P$

in this case our action will be performed as soon as the environment will permit it. This is a simple encoding of the concept of maximal progress which is present in many timed calculi [HR90, Yi90, CAM90]. Obviously care must be taken in the presence of restrictions to maintain the interpretation, but this will always be possible. We will be using this form of asynchronisation in both our eager buffer and the task allocation examples.

We can also use priority to give a form of broadcast. We take a broadcast action to be one which all processes capable of receiving should receive. This translates naturally into a process by exploiting a priority structure, a higher priority is given to transmitting to more receivers. So if we wish to broadcast an action $\overline{a}$ to up to n processes and thereby become the process P we can write the following process:

$Broad(\overline{a}, n, P) \stackrel{def}{=} \omega^n : \overline{a}^n.P + \omega^{n-1} : \overline{a}^{n-1}.P + \ldots + \omega : \overline{a}.P + 1 : \surd.P$

this form of communication is exploited in our example of a fault tolerant synchronisation algorithm. An example using the dynamic nature of priority is that of the implementation of a FCFS discipline which we give later.

2.6. The Eager Buffer.

We wish to define the cells of a buffer which will move stored data into the free cell closest to the data's destination. Removing the data and concentrating instead on its flow we arrive at the following definition of our cells.

For the middle cells of the buffer $(1 < n < t)$, we take the process:

$$Eag_n \stackrel{def}{=} 1 : in_n.Veag_n + 1 : \surd.Eag_n$$
$$Veag_n \stackrel{def}{=} \omega : \overline{in_{n+1}}.Eag_n + 1 : \surd.Veag_n$$

The last cell has the form,

$$Eag_t \stackrel{def}{=} 1 : in_t.Veag_t + 1 : \surd.Eag_t$$
$$Veag_t \stackrel{def}{=} 1 : \overline{out}.Eag_t + 1 : \surd.Veag_t$$

and the first cell has the form,

$$Eag_1 \stackrel{def}{=} 1 : in.Veag_1 + 1 : \surd.Eag_1$$
$$Veag_1 \stackrel{def}{=} \omega : \overline{in_2}.Eag_1 + 1 : \surd.Veag_1$$

We define our buffer as

$$Buf_t \stackrel{def}{=} \Theta((Eag_1 \times Eag_2 \times \ldots \times Eag_t)\lceil\{in, \overline{out}\})$$

The above processes Eag_b could be said to be output eager, that is they try and pass on values as soon as possible. Equally we could have defined our cells to be input eager, grabbing data from an adjacent cell as soon as possible, or even made both activities eager. We hope that it is clear that all of these forms of eagerness lead to the same buffer behaviour, and leave the proof as an exercise.

Definition 2.20. A bit operation $shrf(b)$ takes all sequences ..10.. within b and replaces them with ..01... The operations $first(b, v)$ and $last(b, v)$ make the first and last bits of the bit vector b equal to the value v respectively.

We shall now demonstrate that the above buffer is indeed eager, in the sense that data is moved as far to the right as possible. Necessarily, when concentrating on the control signals, this will resemble a logical shift right operation.

We can define another buffer, using bit vector of length t to describe its state, we use juxtaposition to represent concatenation within the bit vector:

$$Buf'(0b0) \stackrel{def}{=} 1 : in.Buf'(first(shrf(0b0),1)) + 1 : \surd.Buf'(shrf(0b0))$$

This represents a buffer with empty first and last cells, so it can move all internal data along by one cell, and possibly input data into the first cell. The following state is obtained when the first data cell is full, but the last is not:

$$Buf'(1b0) \stackrel{def}{=} 1 : \surd.Buf'(shrf(1b0))$$

so all data is simply moved towards its destination. In the following case the first and last cells are full:

$Buf'(1b1) \stackrel{def}{=} 1 : \surd.Buf'(shrf(1b1)) + 1 : \overline{out}.Buf'(last(shrf(1b1), 0))$

so if data is moved out, then the other data can be moved, using the last cell, otherwise it moves where it can but cannot occupy the last cell. Finally, we consider the case, where both input and output are possible:

$$Buf'(0b1) \stackrel{def}{=} 1 : in.Buf'(first(shrf(0b1), 1)) + 1 : \overline{out}.Buf'(last(shrf(0b1), 0)) + 1 : in\&\overline{out}.Buf'(first(last(shrf(0b1), 0), 1)) + 1 : \surd.Buf'(shrf(0b1))$$

To compare the two forms of the buffer we rename the cells of our original buffer as follows:

$Cell_m(1) \stackrel{def}{=} Veag_m$
$Cell_m(0) \stackrel{def}{=} Eag_m$

It is relatively straightforward to demonstrate that

$Buf'(b) = \Theta((Cell_1(b_1) \times Cell_2(b_2) \times \ldots \times Cell_t(b_t))\lceil\{in, \overline{out}\})$

2.7. A First Come First Served System.

In many systems a single resource can be used by several different users. Often control of that resource is supposed to be allocated on a first come first served (FCFS) basis. Evidently, if many parallel components are all requesting a resource at the same time, with no distinction being made between them, then the resource will be allocated in some (semi-)random fashion. It certainly *cannot* be guaranteed that the resource is allocated to the process that has been waiting the longest. We demonstrate how the FCFS resource allocation strategy can be described very efficiently using the priorities of WSCCS. Indeed, in our description we do not need any auxilary processes. Hence such allocation methods can be included in large system descriptions with great ease.

A FCFS request process can be described as follows:

$P(k) \stackrel{def}{=} \omega^k : \overline{get}\#a.\$.\overline{put}.\$0 + 1 : \surd.P(k+1)$

this is a process which performs a $\overline{get}\#a$ with priority k; if the get is refused it then will attempt the get again, but with the next higher priority after performing a $\surd$ action. Once the resource is claimed it is retained for a period of time and then released. The process is then capable of performing ticks indefinately.

The resource the processes are competing for is a simple semaphor:

$Res \stackrel{def}{=} \$get.Res'$
$Res' \stackrel{def}{=} \$put.Res$

Example 2.21. As an example of the FCFS system consider five processes that have arrived in such a way that they should produce a sequence of actions a, b, c, d, e:

$\Theta((Res \times P(5) \times P(4)[b/a] \times P(3)[c/a] \times P(2)[d/a] \times P(1)[e/a])\lceil\{\surd, a, b, c, d, e\})$

Start by considering the process $\Theta((Res \times P(i)[e/a])\lceil\{\surd, a, b, c, d, e\})$:

$= e.\Theta((Res' \times \$.\overline{put}.\$0)\lceil\{\surd, a, b, c, d, e\})$
$= e.(1 : \surd.\Theta((Res' \times \$.\overline{put}.\$0)\lceil\{\surd, a, b, c, d, e\}) +$
$\quad 1 : \surd.\Theta((Res \times \$0)\lceil\{\surd, a, b, c, d, e\})$

$= e.(1 : \surd.\Theta((Res' \times \$.\overline{put}.\$0)\lceil\{\surd, a, b, c, d, e\}) + 1 : \surd.\$0)$
$= e.\$0$

Now consider the process $\Theta((Res \times P(i+1)[d/a] \times P(i)[e/a])\lceil\{\surd, a, b, c, d, e\})$:

$= d.\Theta((Res' \times \$.\overline{put}.\$0 \times P(i+1)[e/a])\lceil\{\surd, a, b, c, d, e\})$
$= d.(1 : \surd.\Theta((Res' \times \$.\overline{put}.\$0 \times P(i+2)[e/a])\lceil\{\surd, a, b, c, d, e\}) +$
$1 : \surd.\Theta((Res \times \$0 \times P(i+2)[e/a])\lceil\{\surd, a, b, c, d, e\})$
$= d.(1 : \surd.\Theta((Res' \times \$.\overline{put}.\$0 \times P(i+2)[e/a])\lceil\{\surd, a, b, c, d, e\}) +$
$1 : \surd.\Theta((Res \times P(i+2)[e/a])\lceil\{\surd, a, b, c, d, e\})$
$= d.(1 : \surd.\Theta((Res' \times \$.\overline{put}.\$0 \times P(i+2)[e/a])\lceil\{\surd, a, b, c, d, e\}) +$
$1 : \surd.e.\$0)$
$= d.\surd.\$e.\$0$

Similarly we can show that $\Theta((Res \times P(i+2)[c/a] \times P(i+1)[d/a] \times P(i)[e/a])\lceil\{\surd, a, b, c, d, e\})$
$= c.\surd.\$d.\surd.\$e.\$0$

and that $\Theta((Res \times P(i+3)[b/a] \times P(i+2)[c/a] \times P(i+1)[d/a] \times P(i)[e/a])\lceil\{\surd, a, b, c, d, e\})$
$= b.\surd.\$c.\surd.\$d.\surd.\$e.\$0$

and finally that $\Theta((Res \times P(5) \times P(4)[b/a] \times P(3)[c/a] \times P(2)[d/a] \times P(1)[e/a])\lceil\{a, b, c, d, e\})$
$= a.\surd.\$b.\surd.\$c.\surd.\$d.\surd.\$e.\$0$

That is it produces the actions a, b, c, d, e in alphabetical order, and therefore has maintained a FCFS discipline.

To produce a more general example of a queue, we need a process that is capable of producing several processes to compete for the resource:

$$Spawn(n) \stackrel{def}{=} m.\surd.(Spawn(n+1) \times Pro(1)[a_n/a]) + n.\surd.Spawn(n)$$

Composing the spawning process with the resource we get a system where arbitrary numbers of processes will arrive and compete for the resource:

$$Sys \stackrel{def}{=} \Theta((Spawn(1) \times Res)\lceil\{a_1, a_2, \ldots\})$$

If this system has processes which obey the FCFS discipline then it should produce the actions a_i in ascending order, interspersed with $\surd$ actions. We could attempt to show that there is some process of this form identical to Sys, but this would require calculating the rates of process spawning and the times that each process retains the resource. For simplicity we shall demonstrate that the actions marking the aquisition of the resource are performed in ascending order directly. In order to show that this is the case we make the following auxillary definition.

Definition 2.22. Let $Pri(k)$ by the priority associated with the action $\overline{get}\#a_k$ in process Pro within the system Sys.

We can observe that the following is an invariant for the system:

for all $i < j$ if there are processes in Sys capable of performing $\overline{get}\#a_i$ and $\overline{get}\#a_j$, then $Pri(i) > Pri(j)$.

We shall prove that the above is an invariant of the system, and hence that we have constructed a FCFS system.

Proof. We shall show the invariant holds by demonstrating that it is preserved by all possible transitions of the system. As an abbreviation we write $P(i, k) \stackrel{def}{=} P(k)[a_i/a]$.

Consider the generalised initial state $\Theta((Spawn(i) \times Res)\lceil\{a_1, a_2, \ldots\})$ which is equal to:

$$n : \surd.\Theta((Spawn(i) \times Res)\lceil\{a_1, a_2, \ldots\})+$$
$$m : \surd.\Theta((Spawn(i+1) \times Res \times P(i,1))\lceil\{a_1, a_2, \ldots\})$$

In both of the possible resulting processes the invariant holds.

There are two general states which we need to consider depending on whether the resource is currently claimed or not (for $i < m$):

1. $\Theta((Spawn(m+1) \times Res \times P(i,k_i) \times \ldots \times P(m,k_m))\lceil\{a_1, a_2, \ldots\})$, now from the invariant for $j < l$ we have $k_j > k_l$ and hence this process is equal to:

   $$m : a_{i_1}.\Theta((Spawn(m+2) \times Res' \times \$\overline{put}\$nil \times P(i+1, k_{i+1}+1) \times \ldots \times P(m+1,1))\lceil\{a_1, a_2, \ldots\})$$
   $$+n : a_{i_1}.\Theta((Spawn(m+1) \times Res' \times \$\overline{put}\$nil \times P(i+1, k_{i+1}+1) \times \ldots \times P(m, k_m + 1))\lceil\{a_1, a_2, \ldots\})$$

   From the invariant the highest priority claim on the resource is that of $P(i,k_i)$, since k_i is the greatest of all the priorities. Hence, the marking action that is performed is that of the lowest numbered process currently queuing. In the first resulting process we add 1 to each priority (the least of which must be 1) and add a new priority of 1, so the order will be maintained. For the second case we simply add 1 to all but the first priority and hence the order, of the priorities for remaing queuing processes is maintained.
2. $\Theta((Spawn(m+1) \times Res' \times \$\overline{get}.\$0 \times P(i,k_i) \times \ldots \times P(m,k_m))\lceil\{a_1, a_2, \ldots\})$, this process has the following behaviour:

   $$m : \surd.\Theta((Spawn(m+2) \times Res' \times \$\overline{get}.\$0 \times P(i,k_i+1) \times \ldots \times P(m,k_m+1) \times P(m+1,1)\lceil\{a_1, a_2, \ldots\})+$$
   $$m : \surd.\Theta((Spawn(m+2) \times Res \times P(i,k_i+1) \times \ldots \times P(m,k_m+1) \times P(m+1,1)\lceil\{a_1, a_2, \ldots\})+$$
   $$n : \surd.\Theta((Spawn(m+1) \times Res' \times \$\overline{get}.\$0 \times P(i,k_i+1) \times \ldots \times P(m,k_m+1))\lceil\{a_1, a_2, \ldots\})+$$
   $$n : \surd.\Theta((Spawn(m+1) \times Res \times P(i,k_i+1) \times \ldots \times P(m,k_m+1))\lceil\{a_1, a_2, \ldots\})$$

   In all cases the order of the priorities is maintained, as 1 is added to all the existing priorities, and if a new process is added then its priority is 1 and thus must be less than all the others, whilst it has the highest existing index.

From the structure of the above states we can see that the system can *only* reach states where there are either no processes queueing and process (say n) is about to be spawned, or there are all the processes numbered from i to m queuing. Thus the invariant is maintained over all reachable states of the system. $\square$

In the presence of the invariant it is clear that in any state of the system the next visible action to be performed will be that available with the lowest index. Hence, the systems performs the actions a_i in ascending order on i, that is, it is a FCFS system.

3. A Proof Method.

Often with a probabilistic system, the computation proceeds through many states before reaching a final stable configuration, in which we regard the computation as successful. The final configuration may be a single state or a small collection of states. The number of possible intermediate states can be very large and we need an efficient proof method in order to demonstrate that the final (presumably desired) configuration of the system will be achieved. To this end we exploit a well known corollary of *the decomposition theorem* for Markov chains [GS82].

Definition 3.1. We call a set of states $\mathcal{A}$ **closed**, when for all $P \in \mathcal{A}$ if $P \xrightarrow{a[w]} P'$ then $P' \in \mathcal{A}$.

Theorem 3.2. Let P be a finite state process, whose derivatives form closed sets $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n$, then eventually P evolves to a state in $\mathcal{A}_i$, for some i.

For a proof of the above theorem see page 121 [GS82].

Corollary 3.3. If the derivatives of a process P have a single closed set $\mathcal{A}$, then P will eventually evolve to a state in $\mathcal{A}$.

The simplest manner to satisfy the conditions of Corollary 3.3 for a process P; is to exhibit a set of closed states $\mathcal{A}$ all of which are derivatives of P, and to demonstrate that from all other derivatives of P there is a derivation path leading to a state in $\mathcal{A}$. The second condition is sufficient to demonstrate that there are no other disjoint closed sets of states amongst the derivatives of P.

In the following subsections we shall exploit the above methodology to demonstrate the correctness and fault tolerance of two simple probabilistic algorithms for synchronisation and for the allocation of work amongst several processors.

3.1. Synchronisation

In many situations the maintenance of an accurate distributed clock (or system synchronisation) is exploited to simplify the control of distributed programs. We present a simple algorithm which eventually ensures that all processors will be synchronised, in which there is no central clock, i.e., any unit can fail and the synchronisation will be maintained, and which is stable to local clock errors of fixed absolute magnitude.

The algorithm can be summarised as follows:

1. Count a period s on the local clock and enter a responsive state (2);
2. For the next period d, if the synchronisation signal arrives then accept it and synchronise and start the clock again (1), otherwise go to (3);
3. Issue the synchronisation signal with some probability $p \stackrel{def}{=} \frac{m}{m+n}$, or respond to any synchronisation signal.

The number of processors that are synchronising is given by the variable z. We represent each step of the local clock in the above by two ticks of each WSCCS process and arrive at the following formalisation of the algorithm:

$$
\begin{aligned}
&Sleep(s) \stackrel{def}{=} active.\surd.Sleep(s-1)\\
&Sleep(k) \stackrel{def}{=} \surd.\surd.Sleep(k-1)\\
&Sleep(0) \stackrel{def}{=} Doze(d)
\end{aligned}
$$

The above process represents a synchronising process in state (1), it performs $2s$ ticks and then enters the state $Doze(d)$, the *active* action records the hoped for synchronisation point.

For $(0 < k \leq d)$:

$$
\begin{aligned}
&Doze(k) \stackrel{def}{=} \surd.(1 : sync.Sleep(s) + 1 : \surd.Doze(k-1)\\
&Doze(0) \stackrel{def}{=} Decide
\end{aligned}
$$

In the above, if the synchronisation *sync* is input the process returns to the start state, otherwise it performs ticks until it can issue the synchronisation signal.

For $(0 < k \leq z)$:

$$Decide \stackrel{def}{=} m : \surd.Wakeable + n : \surd.Broad(z)$$
$$Broad(k) \stackrel{def}{=} \omega^k : \overline{sync}^k.Sleep(s) + Broad(k-1)$$
$$Broad(0) \stackrel{def}{=} 1 : \surd.Sleep(s)$$
$$Wakeable \stackrel{def}{=} 1 : sync.Sleep(s) + 1 : \surd.Decide$$

The process *Decide* chooses whether or not to issue the synchronisation signal with the given probability. If the process enters the state $Broad(z)$, then it tries to contact as many processes as will listen with the synchronisation signal, by broadcasting it. In the state *Wakeable* if the synchronisation signal is received the process returns to its start state, otherwise it will retry on whether to issue the synchronisation signal.

We can describe a collection of the synchronising processes by taking a collection of times i_j for $1 \leq j \leq z$ and composing z synchronisers together.

$$Sys \stackrel{def}{=} \Theta((Sleep(i_1) \times \ldots \times Sleep(i_z)) \lceil \{active, active^2, \ldots, active^z\})$$

Clearly we should wish to be able to establish that the above system will eventually synchronise, that is to say that it will reach the state:

$$\Theta((Sleep(s) \times \ldots \times Sleep(s)) \lceil \{sync, active^2, \ldots, active^z\})$$

we shall prove that this does indeed occur by using Corollary 3.3.

Firstly we shall describe a closed set of derivatives for the process *Sys* and show that they form a (probabilistic) cycle.

Lemma 3.4. The set of states:

$$\{\Theta((\textstyle\prod_{l=1}^{l=z} Sleep(i)) \lceil \{active, active^2, \ldots, active^z\}) | 1 \leq i \leq s\} \cup$$
$$\{\surd.\Theta((\textstyle\prod_{l=1}^{l=z} Sleep(i-1)) \lceil \{active, active^2, \ldots, active^z\}) | 1 \leq i \leq s\} \cup$$
$$\{\Theta((\textstyle\prod_{l=1}^{l=z} Doze(j)) \lceil \{active, active^2, \ldots, active^z\}) | 1 \leq j \leq d\} \cup$$
$$\{\surd.\Theta((\textstyle\prod_{l=1}^{l=z} Doze(j-1)) \lceil \{active, active^2, \ldots, active^z\}) | 1 \leq j \leq d\} \cup$$
$$\{\Theta((\textstyle\prod_{l=1}^{l=z} Decide) \lceil \{active, active^2, \ldots, active^z\})\} \cup$$
$$\{\Theta((\textstyle\prod_{l=1}^{l=k} Wakeable \times \prod_{m=1}^{m=z-k} Broad(z)) \lceil \{active, \ldots, active^z\}) | 1 \leq k \leq z\} \cup$$
$$\{\Theta((\textstyle\prod_{m=1}^{m=z} Broad(z)) \lceil \{active, active^2, \ldots, active^z\})\} \cup$$
$$\{\Theta((\textstyle\prod_{l=1}^{l=z} Wakeable) \lceil \{active, active^2, \ldots, active^z\})\}$$

(which we call $\mathcal{C}$) is closed.

Proof. We consider each of the states in turn (for $1 < i < s$):

$$\begin{array}{c} \Theta((\prod_{l=1}^{l=z} Sleep(s)) \lceil \{active, active^2, \ldots, active^z\}) \\ \downarrow active^z \\ \surd.\Theta((\prod_{l=1}^{l=z} Sleep(s-1)) \lceil \{active, active^2, \ldots, active^z\}) \end{array}$$

$$\Theta((\prod_{l=1}^{l=z} Sleep(i)) \lceil \{active, active^2, \ldots, active^z\})$$
$$\downarrow \surd$$
$$\surd.\Theta((\prod_{l=1}^{l=z} Sleep(i-1)) \lceil \{active, active^2, \ldots, active^z\})$$

For $i \geq 1$:

$$\surd.\Theta((\prod_{l=1}^{l=z} Sleep(i-1)) \lceil \{active, active^2, \ldots, active^z\})$$
$$\downarrow \surd$$
$$\Theta((\prod_{l=1}^{l=z} Sleep(i-1)) \lceil \{active, active^2, \ldots, active^z\})$$

and for $i = 0$

$$\surd.\Theta((\prod_{l=1}^{l=z} Sleep(0)) \lceil \{active, active^2, \ldots, active^z\})$$
$$\downarrow \surd$$
$$\Theta((\prod_{l=1}^{l=z} Doze(d)) \lceil \{active, active^2, \ldots, active^z\})$$

$$\Theta((\prod_{l=1}^{l=z} Doze(d)) \lceil \{active, active^2, \ldots, active^z\})$$
$$\downarrow \surd$$
$$\surd.\Theta((\prod_{l=1}^{l=z} Doze(d-1)) \lceil \{active, active^2, \ldots, active^z\})$$

For $d > j \geq 1$

$$\surd.\Theta((\prod_{l=1}^{l=z} Doze(j-1)) \lceil \{active, active^2, \ldots, active^z\})$$
$$\downarrow \surd$$
$$\Theta((\prod_{l=1}^{l=z} Doze(j-1)) \lceil \{active, active^2, \ldots, active^z\})$$

and for $j = 0$

$$\surd.\Theta((\prod_{l=1}^{l=z} Doze(0)) \lceil \{active, active^2, \ldots, active^z\})$$
$$\downarrow \surd$$
$$\Theta((\prod_{l=1}^{l=z} Decide) \lceil \{active, active^2, \ldots, active^z\})$$

For the more interesting state, let C_m^n by the usual combinatoric choice function, then:

$$\Theta((\prod_{l=1}^{l=z} Decide) \lceil \{active, active^2, \ldots, active^z\})$$
$$\downarrow \surd[C_z^k m^k n^{(z-k)}]$$
$$\Theta((\prod_{l=1}^{l=k} Wakeable \times \prod_{m=1}^{m=z-k} Broad(z)) \lceil \{active, active^2, \ldots, active^z\})$$

$$\Theta((\prod_{l=1}^{l=k} Wakeable \times \prod_{m=1}^{m=z-k} Broad(z)) \lceil \{active, active^2, \ldots, active^z\})$$
$$\downarrow \surd$$
$$\Theta((\prod_{l=1}^{l=z} P_{(s)}) \lceil \{active, active^2, \ldots, active^z\})$$

For the final two states:

$$\Theta((\prod_{m=1}^{m=z} Broad(z)) \lceil \{active, active^2, \ldots, active^z\})$$
$$\downarrow \surd$$
$$\Theta((\prod_{l=1}^{l=z} Sleep_{(s)}) \lceil \{active, active^2, \ldots, active^z\})$$

$$\Theta((\prod_{l=1}^{l=z} Wakeable)\lceil \{active, active^2, \ldots, active^z\}))$$
$$\downarrow \surd$$
$$\Theta((\prod_{l=1}^{l=z} Decide)\lceil \{active, active^2, \ldots, active^z\})$$

□

We now need to demonstrate that for any other derivatives of *Sys* there is a derivation path leading to a state in our closed set.

Lemma 3.5. For any derivative of *Sys* there is a derivation path leading to a state in $\mathcal{C}$.

Proof. Consider a general state of the synchroniser, not in $\mathcal{C}$ with none of the sub-processes in either of the *Decide*, *Wakeable* or *Broad*(k) states. From any state with sub-processes in the above states consider some path that removes all such sub-processes, this will always exist and have non-zero probability.

From any state where all sub-processes are *Sleep*(i) or *Doze*(j) states with non-zero values we can clearly evolve to a state in the following form:

$$\Theta((\prod_{i=1}^{i=l} Decide \times \prod_{j=1}^{j=m} Doze(d_j) \times \times \prod_{k=1}^{k=n} Sleep(s_k))\lceil \{activ, \ldots, active^z\})$$

with $l+m+n = z$ and $d_j > 0$ and $s_k > 0$. That is we simply perform $\surd$ actions until the first sub-processes reach the decide state.

From the above state we simply insist that when any process reaches a *Decide* state it **does not** issue the synchronisation message, (we assume the *total* number of times this decision is taken to be t), eventually (after $2 * (max(s_k) + d)$ ticks) the process will reach the state:

$$\Theta((\prod_{l=1}^{l=z} Decide)\lceil \{active, active^2, \ldots, active^z\})$$

with the probability of taking this path being p^t. Hence, from any state there is a derivative path to a state in $\mathcal{C}$. □

At this point we can indeed derive our desired theorem:

Theorem 3.6. Given the process *Sys* with any values for $i_1, \ldots, i_z$, then eventually *Sys* reaches the state:

$$\Theta((Sleep(s) \times \ldots \times Sleep(s))\lceil \{active, active^2, \ldots, active^z\})$$

and cycles through the states of $\mathcal{C}$ as shown.

Proof. From Lemmas 3.4 and 3.5 and Corollary 3.3. □

A straightforward use of probability theory demonstrates that the expected cycle time of the complete system (in the original units is):

$$s + d + \frac{1}{1-p^z}$$

in the above the cycle time's dependence on p gets very small for large z, in which case the cycle will have length $s+d+1$ with a reasonably high accuracy. The standard deviation of the cycle length is:

$$\frac{p^z}{(1-p^z)^2}$$

which can be made very small as z becomes large. Thus we can achieve cycles of reasonably fixed durations.

3.1.1. Error Properties of the Synchronisers.

It is clear in the above analysis that if any of the individual cyclers fail, that is becomes the process \$0, then the system as a whole will be unaffected (except for a slight change in the mean cycle length). Of more interest is the problem of different local clocks not agreeing over duration. We can model this by taking a process with an integer error of magnitude e in the amount of time s before it starts to wait for the synchronisation signal. Such a process can be described as follows.

For $0 < k < s - e$, $0 < j < d$ and $0 < l \leq z$:

$$
\begin{aligned}
&Esleep(s-e) \stackrel{def}{=} active.\surd.Esleep(s-e-1)\\
&Esleep(k) \stackrel{def}{=} \surd.\surd.Ehsleep(k-1)\\
&Esleep(0) \stackrel{def}{=} Ehdoze(d)\\
&Edoze(j) \stackrel{def}{=} \surd.(1 : sync.Esleep(s-e) + 1 : \surd.Edoze(j-1)\\
&Edoze(0) \stackrel{def}{=} Edecide\\
&Edecide \stackrel{def}{=} m : \surd.Ewakeable + n : \surd.Ebroad(z)\\
&Ebroad(l) \stackrel{def}{=} \omega^k : sync^k.Esleep(s-e) + Ebroad(l-1)\\
&Ebroad(0) \stackrel{def}{=} \surd.Esleep(s-e)\\
&Ewakeable \stackrel{def}{=} 1 : sync.Esleep(s-e) + 1 : \surd.Edecide
\end{aligned}
$$

and we can consider a system with this process composed with the original synchronisers:

$$Sys \stackrel{def}{=} \Theta((Esleep(s_1) \times Sleep(s_2) \times \ldots \times Sleep(s_{z+1}))\lceil\{active, active^2, \ldots, active^z\})$$

Assuming that $e < d$ then we can show that the following collection of states forms a closed set:

$$
\begin{aligned}
&\{\Theta((Esleep(s-e-h) \times \textstyle\prod_{l=1}^{l=z} Sleep_{(h)})\lceil \{active, \ldots, active^z\})| 0 \leq h \leq s-e\} \cup\\
&\{\surd.\Theta((Esleep(s-e-h-1) \times \textstyle\prod_{l=1}^{l=z} Sleep_{(h-1)})\lceil \{active, \ldots, active^z\})| 0 \leq h \leq s-e\} \cup\\
&\{\surd.\Theta((Edoze(i) \times \textstyle\prod_{l=1}^{l=z} Sleep_{(i)})\lceil \{active, \ldots, active^z\})| 1 \leq i \leq e\} \cup\\
&\{\surd.\Theta((Edoze(i) \times \textstyle\prod_{l=1}^{l=z} Sleep_{(i)})\lceil \{active, \ldots, active^z\})| 1 \leq i \leq e\} \cup\\
&\{\Theta((Edecide \times \textstyle\prod_{l=1}^{l=z} Doze(j))\lceil \{active, \ldots, active^z\})| 1 \leq j \leq d-e\} \cup\\
&\{\surd.\Theta((Edecide \times \textstyle\prod_{l=1}^{l=z} Doze(j-1))\lceil \{active, \ldots, active^z\})| 1 \leq j \leq d-e\} \cup\\
&\{\Theta((Edecide \times \textstyle\prod_{l=1}^{l=z} Decide)\lceil \{active, \ldots, active^z\})\} \cup\\
&\{\Theta((Ebroad(z) \times \textstyle\prod_{l=1}^{l=k} Wakeable \times \prod_{m=1}^{m=z-k} Broad(z))\lceil \{active, \ldots, active^z\})\} \cup\\
&\{\Theta((Ebroad(z) \times \textstyle\prod_{m=1}^{m=z} Broad(z))\lceil \{active, \ldots, active^z\})\} \cup\\
&\{\Theta((Ebroad(z) \times \textstyle\prod_{l=1}^{l=z} Wakeable)\lceil \{active, \ldots, active^z\})\} \cup\\
&\{\Theta((Ewakeable \times \textstyle\prod_{l=1}^{l=k} Wakeable \times \prod_{m=1}^{m=z-k} Broad(z))\lceil \{active, \ldots, active^z\})| 1 \leq k < z\} \cup\\
&\{\Theta((Ewakeable \times \textstyle\prod_{m=1}^{m=z} Broad(z))\lceil \{active, \ldots, active^z\})\} \cup\\
&\{\Theta((Ewakeable \times \textstyle\prod_{l=1}^{l=z} Wakeable)\lceil \{active, \ldots, active^z\})\}
\end{aligned}
$$

In the above set the interesting states are those of the form:

$$\Theta((Edecide \times \textstyle\prod_{l=1}^{l=z} Doze(j))\lceil \ \{active, active^2, \ldots, active^z\})$$

In this case if the erroneous process decides to send the synchronisation signal, then all the dozing processes will respond, that is, it evolves to the state:

$$\surd.\Theta((Esleep(s-e-h) \times \prod_{l=1}^{l=z} P_{(h)}) \lceil \{active, active^2, \ldots, active^z\})$$

otherwise it will not give the signal (and neither can any other process) in which case it will be in the state:

$$\surd.\Theta((Edecide \times \prod_{l=1}^{l=z} Doze(j-1)) \lceil \{active, active^2, \ldots, active^z\})$$

both of which are in the set of closed states.

A similar closed set can be constructed with $-e < d$, which we omit for reasons of space. The demonstration that from all other states there is a derivation leading to these states is similar to that given earlier and omitted.

Thus we have an algorithm for synchronisation which is unaffected by local clock errors whose absolute magnitude is less than the value of d. This algorithm was motivated by the attempt to provide an account of a synchronisation behaviour observed for some species of ants, a full account of this work can be found in [Tof90b,Tof93] and in its biological context in [THF92].

3.2. Work Load Division

The problem of allocating a workload over several processing units is common to many distributed systems. We present an algorithm for allocating processors to tasks; which is dynamic, tolerant of errors, and avoids unnecessary reallocation of processors. The system we consider is one where there are some number of processors k handling t different types of task, with amounts work arriving at each task type in some (possibly random) fashion. For simplicity we number the tasks, and consider them arranged linearly with the lowest numbered on the left and the highest on the right[5]. Processors can change the type of task they will perform and will do so in response to an overloading of processors performing their current task type. The intent of such a system is to maximize the amount of work performed by allocating the processors to task type in such a way that, if it is possible to do so, the loading requirements are met. Obviously we would like such a system to dynamically reconfigure itself to meet changing work requirements, and to minimize the scope and amount of interprocess communication.

We consider that the tasks are numbered sequentially. The algorithm works in the following manner:

1. a processor tries to perform a task within its current task area;
2. if there is no work, attempt to communicate with processors in the tasks immediately above and below you;
3. if that communication continually fails in a particular direction too many times consider moving in that direction with some probability (dependent on the direction);
4. return to state (1).

The last 3 behaviours have the effect of leaving all unoccupied processors distributed evenly amongst the tasks.

[5] This originated from thinking of task zones arranged along a production line system.

We start by defining a process that places an amount of work, in the set $Range \stackrel{def}{=} \{0,1,2,\ldots,r\}$, into each task in accordance with an arbitrary distribution:

$$Prod_z(r) \stackrel{def}{=} \Sigma_{i=0}^{i=r} 1 : \overline{work}_z^i.\surd.\sum_{i\in Range} w_i.\ \surd.Prod_z(i)$$

We describe our allocation algorithm as follows:

$$Wrk_z(k_l,k_r) \stackrel{def}{=} \omega : work_z.\surd.\surd.Wrk_z(0,0) + 1 : Tsk_z(k_l,k_r)$$

In the above, the processor has found work of the type it currently wishes to perform so it does it and stays with this task type. Once if fails to find work it attempts to ensure that all waiting processors are evenly allocated to task types, by communicating with processors above and below it in the task sequence. The action $task_{z+1}$ contacting a processor to the right, and $\overline{task_z}$ informing a processor to the left of its presence. Only *unoccupied* processors engage in these communications.

$$Tsk_z(k_l,k_r) \stackrel{def}{=} \omega : task_{z+1}\overline{task_z}.\surd.Wrk(0,0)+ 1 : Tsk'_z(k_l,k_r)$$

if it can perform the action $task_{z+1}\overline{task_z}$ then it assumes that the balance is maintained and will not move. In the following if contact cannot be made to either the task below, or above, or both:

$$\begin{aligned} Tsk'_z(k_l,k_r) \stackrel{def}{=} & (\omega : (1.task_{z+1}.Downd_z(k_l,k_r+1)+ 1 : \overline{task_z}.Upd_z(k_l+1,k_r))+ \\ & 1 : \surd.Bothd_z(k_l+1,k_r+1)) \end{aligned}$$

then the processor *considers* whether to move in the appropriate direction in order to reestablish balance. The parameters k_l and k_r indicate the number of times contact has been refused to the left and the right respectively. If balance may have been lost with the task zone above:

$$Upd_z(k_l,k_r) \stackrel{def}{=} \begin{cases} \surd.Wrk_z(k_l,k_r) & \text{if } k_r < k_{rc} \\ m : \surd.Task_{z+1}(0,0) + n : \surd.Wrk_z(0,0) & \text{otherwise.} \end{cases}$$

when the contact, to the right, has been lost for more than k_{rc} times then decide to move in that direction with probability $\frac{m}{m+n}$. Similarly, if contact has been lost with the task below:

$$Downd_z(k_l,k_r) \stackrel{def}{=} \begin{cases} \surd.Wrk_z(k_l,k_r) & \text{if } k_l < k_{lc} \\ m' : \surd.Wrk_{z-1}(0,0) + n' : \surd.Wrk_z(k_l,k_r) & \text{otherwise.} \end{cases}$$

when the contact, to the left, has been lost for more than k_{lc} times then decide to move in that direction with probability $\frac{m'}{m'+n'}$. Finally if contact has been lost in both directions:

$$\begin{aligned} Both_z(k_l,k_r) \stackrel{def}{=} & mn' : \surd.Wrk_{z+1}(0,0) + m'n : \surd.Wrk_{z-1}(0,0)+ \\ & (mm' + nn') : \surd.Wrk_z(k_l,k_r) \end{aligned}$$

$$Bothd_z(k_l,k_r) \stackrel{def}{=} \begin{cases} Both_z(k_l,k_r) & \text{if } k_l \geq k_{lc} \text{ and } k_r \geq k_{rc} \\ Upd_z(k_l,k_r) & \text{if } k_r \geq k_{rc} \text{ and } k_l < k_{lc} \\ Downd_z(k_l,k_r) & \text{otherwise.} \end{cases}$$

then if the number of failed contacts has exceeded the critical number; choose whether to move in that direction with the appropriate probability. Finally, we assume that we have a fixed number of different task types ranging from 1 to t, so we have to provide versions of the above for the extremal task types, where there is no task below or above respectively.

$$Wrk_1(k_l,k_r) \stackrel{def}{=} \omega : work_z.\surd.\surd.Wrk_1(0,0) + 1 : Tsk_1(k_l,k_r)$$
$$Tsk_1(k_l,k_r) \stackrel{def}{=} \omega : task_2.\surd.Wrk_1(0,0) + 1 : \surd.Upd_1$$
$$Upd_1(k_l,k_r) \stackrel{def}{=} \begin{cases} \surd.Wrk_1(k_l,k_r) & \text{if } k_r < k_{rc} \\ m : \surd.Wrk_2(0,0) + n : \surd.Wrk_1(k_l,k_r) & \text{otherwise.} \end{cases}$$

$$Wrk_t(k_l,k_r) \stackrel{def}{=} \omega : work_z.\surd.\surd.Wrk_t(0,0) + 1 : Tsk_t(k_l,k_r)$$
$$Tsk_t(k_l,k_r) \stackrel{def}{=} \omega.\overline{task_t}.\surd.Wrk_t(0,0) + 1 : \surd.Downd_t(k_l,k_r)$$
$$Downd_t(k_l,k_r) \stackrel{def}{=} \begin{cases} \surd.Wrk_t(k_l,k_r) & \text{if } k_l < k_{lc} \\ m' : \surd.Wrk_{t-1}(0,0) + n' : \surd.Wrk_z(k_l,k_r) & \text{otherwise.} \end{cases}$$

So our description of the task allocation system will be as follows (given a sequence x_i of values in the range $1 \ldots t$):

$$Ex \stackrel{def}{=} \Theta(((\prod_{i=1}^{k} Wrk_{x_i}(0,0)) \times Prod_1 \times \ldots \times Prod_t) \lceil \{\surd\})$$

In the above the constants x_i describe the initial allocation of processors to task types. The constants k_{lc} and k_{rc} fix the 'elasticity' of the system, that is how likely a processor is to move.

3.2.1. Fixed Work Levels.

Initially for simplicity we assume that work levels in each task type are fixed, and we shall demonstrate that if the work load and the processor number match then the system can allocate sufficient processors into each task to carry out the work required. This is the case when the producer process is of the following form:

$$Pr_z(r) \stackrel{def}{=} \Sigma_{i=0}^{i=r} 1 : \overline{work}_z^i.\surd.\surd.Pr_z(w_z)$$

We take the work in task z to be w_z. If we assume that we have k processors and that $\sum_{i=1}^{i=t} w_z = k$ then we prove that the system Ex will allocate processors correctly to task types.

Theorem 3.7. The system

$$Ex' \stackrel{def}{=} \Theta(((\prod_{i=1}^{k} Wrk_{x_i}(0,0)) \times Pr_1(w_1) \times \ldots \times Pr_t(w_t)) \lceil \{\surd\})$$

with $\sum_{i=1}^{i=t} w_z = k$ eventually evolves into the state:

$$\Theta(((\prod_{i=1}^{w_1} Wrk_1(0,0)) \times \prod_{i=1}^{w_t} Wrk_t(0,0) \times Pr_1(w_1) \times \ldots \times Pr_t(w_t)) \lceil \{\surd\})$$

We omit the details of the proof (see [Tof91b])for reasons of space, except to observe that the following is a closed set of states, for the process Ex':

$$\{\Theta(((\prod_{i=1}^{w_1} Wrk_1(0,0)) \times \prod_{i=1}^{w_t} Wrk_t(0,0) \times Pr_1(w_1) \times \ldots \times Pr_t(w_t)) \lceil \{\surd\})$$

$$\surd.\surd.\Theta(((\prod_{i=1}^{w_1} Wrk_1(0,0)) \times \prod_{i=1}^{w_t} Wrk_t(0,0) \times Pr_1(w_1) \times \ldots \times Pr_t(w_t)) \lceil \{\surd\})$$

$$\Theta(((\prod_{i=1}^{w_1} Wrk_1(0,0)) \times \prod_{i=1}^{w_t} Wrk_t(0,0) \times Pr_1(w_1) \times \ldots \times Pr_t(w_t)) \lceil \{\surd\})\}$$

In order to show that the above system is dynamic we should like to demonstrate that as we vary the values of w_i then the allocation of processors changes to meet the new demands upon the system. Obviously this can only work if the changes in the w_i occur sufficiently infrequently to allow the system to 'catch up'. We observe that if we took an arbitrary allocation of processors to task types as our starting position; clearly this could be the correct allocation of processors for one set of workloads w_i. Now consider running this system, *but* with fixed workloads w_i', from the above theorem 3.7. If there is a correct allocation of processors the system will reach it, so the processors will eventually be reconfigured correctly. The reason for the dynamism is that the stability of the system is dynamic (with respect to workloads), so it can 'track' any changes in demand upon it, eventually reallocating processors in the correct pattern.

3.2.2. Performance with Erroneous Processors

We can consider the effect on the performance of our allocation system of various forms of erroneous processors. We assume that we have k potential processing units, one of which is erroneous. The simplest is one that performs no work, that is a process:

$$Lazy \stackrel{def}{=} \$0$$

In this case the system will behave as if it had $k-1$ processors available and allocate them correctly amongst tasks.

A more interesting form of erroneous process is one that is 'locked' into the performance of a single particular task, for example the process:

$$Fix_2 \stackrel{def}{=} \omega : work_2.\surd.\surd.Fix_2 + 1 : \surd.\surd.\surd.Fix_2$$

which will only perform work of task type 2. In this case provided there is work in task 2, the system can correctly allocate **all** k processors to tasks, but obviously when there is no work to be done of type 2, the system cannot allocate processors correctly.

This algorithm was original derived to give an account of the allocation of work amongst ants with only a single physical form. For a full account see [Tof91, Tof93] and for the biological implications see [TF92].

4. Conclusions and further work.

We have presented a calculus of relative frequency that is both elegant and simple. We can express any degree of relative frequency between processes; for example:

$$n : A + m : B \qquad \omega : A + n : B$$

express respectively that A and B occur, in a fixed ratio, or with A dominant. The notion of priority developed is very flexible and gives us a high degree of expressive power. Unlike the work of [BBK86], we have encoded priority syntactically into our processes and hence it is dynamic with respect to particular actions.

The applicability of the calculus is demonstrated well by the four examples presented. It should be noted that the inclusion of probabilistic information may make process definitions more complex but it allows access to some powerful and general proof methods. The preservation of priority information over our parallelism operator permits rapid identification of the highest prioritised object within a process, and thus can greatly reduce the effort required in proofs.

The calculus has proved quite successful in describing the properties of some complex systems, especially in terms of the ease of proof in some of these circumstances. Further examples of the use of the calculus in describing complex biological and computational systems are being investigated. As a methodology for performing calculations it has had some success and we are investigating further more complex examples to try and discover more powerful proof and calculational methodologies.

References

[BBK86] Baeten, J., J. Bergstra and J. Klop, Syntax and defining equations for an interrupt mechanism in process algebra, *Fundamenta Informatica* IX, pp 127-168, 1986.

[CAM90] Chen, L., S. Anderson and F. Moller, A Timed Calculus of Communicating Systems, Research Report, Dept. Of Computer Science, Edinburgh University, LFCS-127, 1990

[Cam89] Cammilleri. J., Introducing a Priority Operator to CCS, Computer Laboratory Technical Report, Cambridge University, 1989.

[Chr90] Christoff, I., Testing Equivalences and Fully Abstract Models for Probabilistic Processes, *in Proceedings Concur '90*, LNCS 458, 1990.

[GS82] Grimmet, G.R. and D.R. Stirzaker, *Probability and Random Processes*, Oxford Science Publications, 1982.

[GSST90] van Glabbek, R., S. A. Smolka, B. Steffen and C.Tofts, Reactive, Generative and Stratified Models of Probabilistic Processes, *in proceedings LICS 1990*, 1990.

[Hoa85] C. A. R. Hoare, *Communicating Sequential Processes*, Prentice-Hall 1985.

[HR90] M. Hennessey and T. Regan, A Temporal Process Algebra, Technical Report, Department of Cognitive Science, Sussex University, 1990.

[HTF91] Hatcher, M., C. Tofts and N. Franks, Mutual Exclusion as a Mechanism for Information Exchange in Ant Nests, *in Proceedings of the First European Conference on Social Insects*, 1991.

[Jon90] Jones, C.C.M., *Probabilistic Non-determinism*, PhD Thesis University of Edinburgh, 1990.

[Kei] Keilson, J., *Markov Chain Models - Rarity and exponentiality*, Applied Mathematical Sciences 28, Springer Verlag.

[JS90] Jou, C. and S. Smolka, Equivalences, Congruences and Complete Axiomatizations for Probabilistic Processes, *in Proceedings Concur '90*, LNCS 458, 1990.

[LS89] Larsen, K.G. and A. Skou. Bisimulation through probabilistic testing. *in proceedings POPL 1989*, 1989.

[Mil83] Milner, R., Calculi for Synchrony and Asynchrony, *Theoretical Computer Science* 25(3), pp 267-310, 1983.

[Mil90] Milner, R., *Communication and Concurrency*, Prentice Hall, 1990.

[MT90] Moller, F., and C. Tofts, A Temporal Calculus of Communicating Systems, *in proceedings Concur '90*, LNCS 458, 1990.

[Plo81] Plotkin, G.D., A structured approach to operational semantics. Technical report Daimi Fn-19, Computer Science Department, Aarhus University. 1981.

[RR86] Reed, G., and W. Roscoe, A Timed Model for CSP, *inProceedings ICALP '86*, LNCS 226, 1986.

[SS90] Smolka, S., and B. Steffen, Priority as Extremal Probability, *in proceedings Concur '90*, LNCS 458, 1990.

[SST89] Smolka, S., B. Steffen and C. Tofts, unpublished notes. Working title, Probability + Restriction ⇒ priority.

[THF92] Tofts,C., M.J.Hatcher, N. Franks, Autosynchronisation in Leptothorax Acervorum; Theory, Testability and Experiment, *Journal of Theoretical Biology* 157: 71-82.

[TF92] Tofts, C., and N. Franks, Doing the Right Thing: Ants, Bees and Naked Mole Rats, *Trends in Evolution and Ecology* 7: 346-349.

[Tof89] Tofts, C., Timing Concurrent Processes, LFCS-report number 103, 1989.

[Tof90a] Tofts, C., A Synchronous Calculus of Relative Frequency, *in proceedings CONCUR '90*, Springer Verlag, LNCS 458, 1990

[Tof90b] Tofts, C., The Autosynchronisation of *Leptothorax Acervorum* (Fabricius) Described in WSCCS, LFCS-Report Number 128.

[Tof91] Tofts, C., Task Allocation in Monomorphic Ant Species, LFCS-Report Number 128.

[Tof93] Tofts, C., Using Process Algebra to Descirbe Social Insect Behaviour, *Transactions of the Society for Computer Simulation*, 1993.

[Yi90] Yi W., Real-Time Behaviour of Asynchronous Agents, *in proceedings Concur '90*, LNCS 458, pp 502-520, 1990.