Analytic and Locally Approximate Solutions to Properties of Probabilistic Processes

C. Tofts*†

ABSTRACT Recent extensions to process algebra can be used to describe performance or error rate properties of systems. We examine how properties of systems expressed in these algebras can be elicited. Particular attention is given to the ability to describe the behaviour of system components parametrically. We present how analytic formulae for performance properties can be derived from probabilistic process algebraic descriptions; demonstrating how local approximate solutions can be derived for the properties when their exact solutions would be too computationally expensive to evaluate. As an example we derive the performance of an Alternating Bit Protocol with respect to its error and retry rates.

1 Introduction

Process algebra [Mil80, Mil83, BK84, Hoa85, BBK86, Mil90] is a methodology for formally calculating the behaviour of a system in terms of the behaviours of its components. Recent extensions have added: timing properties [RR86, Tof89, MT90, Yi90, CAM90]; probabilistic properties [GSST90, Tof90, SS90, Tof94]; priority properties [BBK86, Cam89, Tof90, SS90, Tof94] and combinations of the above [Tof90, Han92, Tof94]. Process algebras with these extensions can be exploited to formally analyse the performance (in terms of either success or failure) of the design of systems [VW92,Tof93]. The analysis of a design can be greatly facilitated if an analytic solution to the performance of the system can be generated from an abstract description of the performance of its components. A possibly more important question is how tolerant to error (in the precise value of component parameters) are system level predictions.

Within the process algebra community the standard approach to a ver-

*This work is supported by an EPSRC Advanced Fellowship.

†Department of Computer Science, The University, Manchester, M13 9PL, email: cmnt@cs.man.ac.uk

ification problem, is to describe and compose the system components and then verify by comparing the constructed system's behaviour with another (presumably correct) process [Chr90,JS90], or observing its compliance with a logical predicate [Han94,HJ94]. Whilst in many cases, where for instance design criterion are known in advance, this can be an appropriate methodology, it is however limited for the analysis of choices in system design. Often the requirement is to predict the effect of the component choice on system performance in the context of a service/cost trade-off, rather than compliance with a particular *ab initio* service requirement. Of great importance is the ability to 'track' the effect of a single component upon system performance. To achieve this we need two things, firstly a syntactic presentation of system components, secondly an abstract method of calculating the component's contribution to the system's performance.

Within a system subject to failure system requirements are often expressed in terms like; *the probability of error is less than 0.05.* It is hard to see how to interpret such a requirement in terms of the behaviour at a particular state. Indeed such requirements would often be re-expressed as *the probability of failure at* **any state** *is less than 0.05.* Whilst this condition is certainly sufficient to ensure the conformance of a system to the requirement, is it reasonable? Consider the following WSCCS [Tof90,Tof94] process:

$$\begin{aligned} P_1 &\overset{def}{=} 9.\surd : P_1 + 1.\surd : P_2 \\ P_2 &\overset{def}{=} 1.error : P_1 + 9.\surd : P_1 \end{aligned}$$

The process P_1 certainly does not obey the condition that the probability of error in all states is less than 0.05 as this probability is 0.1 in state P_2. However, the process will only spend 10% of its time in state P_2 hence the probability of error is only 0.01, which does indeed meet our performance requirement. In order to calculate the error probability of this system we need to know the probability of the system being in any particular state. These probabilities can only be evaluated with respect to the complete system, and hence any logic suitable to express these properties will need to express probabilities of being in a particular state, and thus will not be an abstraction on any underlying transition description.

A frequently used method to formally derive the compliance of a probabilisitic system with some requirements is to express those requirements in the form of a 'standard' process [Chr90,JS90,Tof90], then demonstrating an equality between the intended implementation and the standard. If we attempt to describe our requirement on errors in this fashion we might write the following process:

$$Q \overset{def}{=} 95.\surd : Q + 5.error : Q$$

The above being a process which certainly does not produce errors at a greater rate than 0.05. There's appears to be no sensible formal relation between the process Q and our previous example P_1. Again the reason for this incompatibility is that we compare processes on a state by state basis.

A possibly more realistic question would be the following. Given the process:

$$\begin{aligned} R_1 &\stackrel{def}{=} p.\surd : R_1 + 1.\surd : R_2 \\ R_2 &\stackrel{def}{=} 1.error : P_1 + q.\surd : P_1 \end{aligned}$$

what values of the expressions p and q will ensure that the process does not produce error actions at a greater rate than 0.05?

In many cases systemic requirements are expressed in terms of average performance. That is to say the average time before an error is seen will be greater than some amount, or alternatively the average time to see a 'good' outcome will be less than some amount. In order that such performance parameters can be derived we need to know not only the probability of reaching a particular state, but also how long it will take system to do so.

In Section 2 we present an extension to WSCCS to permit reasoning over weight expressions containing variables, and demonstrate how a Markov chain [Kei,Kle75,GS82] can be derived from a WSCCS process. In Section 3 we discuss how the properties of terminating processes can be derived. In Section 4 we discuss the solving for properties of finite processes. In particular, we examine how approximate analytic solutions can be derived when computing an exact analytic solution to the problem will be infeasible. The form of approximation we shall obtain will be in the form of a polynomial expansion of small perturbations about particular values for systemic parameters, and hence they are approximations valid only in a particular locality. We can obtain solutions over an arbitrary range of system parameters by exploiting a series of local approximations for our performance problem.

2 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 a free abelian group *Act* over a set of atomic action symbols with identity $\surd$, the inverse of a being $\overline{a}$, and action product denoted by $\#$. As in SCCS, the complementary actions a (conventionally input) and $\overline{a}$ (output) form the basis of communication. Within our group we define that $\overline{a} = a^{-1}$.

2.1 Expressions

We define a set of expressions.

Definition 2.1 *A* relative frequency expression (RFE) *is formed from the following syntax, with x ranging over a set of variable names VRF, and c ranging over a fixed field (such as $\mathcal{N}$ or $\mathcal{R}$):*

$$e ::= x|c|e+e|e*e$$

Further we assume that the following equations hold for relative frequency expressions:

$$\begin{array}{lcl} e+f & = & f+e \\ (e+f)+g & = & e+(f+g) \\ e*f & = & f*e \\ (e*f)*g & = & e*(f*g) \\ e*(f+g) & = & e*f+e*g \end{array}$$

alternatively, we have commutative and associative addition and multiplication, with multiplication distributing over addition. We shall assume that two expressions are equivalent if they can be shown so by the above equations.

In the sequel we shall omit the $*$ in expressions, denoting expression multiplication by juxtaposition. It should be noted that unlike other calculi with expressions [Mil90, Hen91] the value of our expressions can have **no effect** on the structure of the transition graph of our system. Hence we should not expect that adding this extra structure to our probabilistic process algebra will cause any new technical difficulties.

2.2 Weights

We also take a set of weights $\mathcal{W}$, denoted by w_i, which are of the form $e\omega^k$. In the weight the relative frequency expression e denotes the relative frequency with which a process guarded by this weight will be chosen. The priority with which this choice should be taken is denoted by the strictly positive natural k. We take the following multiplication and addition rules (assuming $k \geq k'$) over weights.

$$\begin{array}{ll} e\omega^k + f\omega^{k'} = e\omega^k = f\omega^{k'} + e\omega^k & \quad e\omega^k + f\omega^k = (e+f)\omega^k = e\omega^k + f\omega^k \\ e\omega^k * f\omega^{k'} = (ef)\omega^{k+k'} = f\omega^{k'} * e\omega^k & \end{array}$$

As abbreviations we use e for the weight $e\omega^0$, and ω^k for the weight $1\omega^k$.

2.3 The Calculus

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 functions, 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:

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

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

The informal interpretation of our operators is as follows:

- $\mathbf{0}$ 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 $\mathbf{0}$, and will use a binary plus instead of the two (or more) element indexed sum, thus writing $\sum\{1_1.a : \mathbf{0},\ 2_2 : b.\mathbf{0}|i \in \{1,2\}\}$ as $1.a + 2.b$. Finally we allow ourselves to specify processes definitionally, by providing recursive definitions of processes. For example, we write $A \overset{def}{=} a.A$ rather than $\mu x.ax$. The weight n is an abbreviation for the weight $n\omega^0$, and the weight w^k is an abbreviation for the weight $1\omega^k$.

The semantics, congruences and equational theory of this (minor) extension of WSCCS are essentially identical to that of [Tof95] up to arithmetic on weight expressions.

The congruences of WSCCS[Tof90,Tof94] are important as they permit us to algebraically manipulate our processes. However, in many instances these equivalences are too fine, consider the following pair of processes:

$$2.(2.P + 4.Q) \qquad\qquad 4.P + 8.Q$$

in many instances we should like to be able to consider these processes as equivalent. Hence, 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. Whilst this notion of equivalence is useful it is known *not* to produce a congruence [SST89] for the complete language. However, such problems do not arise if we restrict our process syntax to only allow a single depth of summation, in which case our abstract relationship $\overset{a}{\sim}$, defined below, coincides with the original probability preserving congruence [Tof94].

Definition 2.2 *We define an abstract notion of evolution as follows;*

$$P \xrightarrow{a[w]} P' \text{ iff } P \overset{w_1}{\longmapsto} \ldots \overset{w_n}{\longmapsto} \xrightarrow{a} P' \text{ with } w = \prod w_i.$$

As an example, $5.(3.(2.a : Q + 4.b : P) + 1.c : R) + 7.d : S \xrightarrow{a[30]} Q$.

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

Definition 2.3 *Let S be a set of processes then:*

$$P \xrightarrow{a[w]} S \text{ iff } w = \sum\{w_i | P \xrightarrow{a[w_i]} Q \text{ for some } Q \in S\};$$ [1]

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

Definition 2.4 *We say an equivalence relation $R \subseteq Pr \times Pr$ is an* abstract bisimulation *if $(P, Q) \in R$ implies that:*

there are $e, f \in RFE$ such that for all $S \in Pr/R$ and for all $w, v \in \mathcal{W}$, $P \xrightarrow{a[w]} S$ iff $Q \xrightarrow{a[v]} S$ and $ew = fv$.

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

In particular this description of a WSCCS process gives us (essentially) a **probability transition graph**[Paz71].

Definition 2.5 *A probabilistic transition graph is a quintuple (V, T, s_0, A, RFE) where V is a set of states, T a set of transitions $\subseteq V \times (a \times p) \times V$, $s_0 \in V$ is an initial state, A ranged over by a an alphabet, and RFE ranged over by p the set of relative frequency expressions.*

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

3 Terminating Systems

As an example consider the following simple game. Two identical (possibly) biased coins are tossed repeatedly. If the coins both show heads then the game is won, if the coins both show tails then the game is lost, otherwise the coins are tossed again. What is the probability of winning the game? And how many tosses will be needed on average to see an outcome?

$$\begin{array}{lcl} Coin & \overset{def}{=} & p.\overline{head} : Coin + q.\overline{tail} : Coin \\ GR & \overset{def}{=} & 1.head^2\#\overline{win} : 0 \\ & & 1.head\#tail : GR \\ & & 1.tail^2\#\overline{lose} : 0 \\ Game & \overset{def}{=} & (Coin \times Coin \times GR)\lceil\{win, lose\} \end{array}$$

The probability of winning a game can be computed by solving the following equation[2]:

$$P(win) \;=\; \tfrac{2pq}{(p+q)^2}P(win) + \tfrac{p^2}{(p+q)^2}.1$$

and the average number of coin tosses equired to reach an outcome:

$$\begin{array}{lcl} E(Game) & = & \frac{2pq}{(p+q)^2}(E(Game)+1) + \frac{p^2}{(p+q)^2}(E(0)+1) \\ & & +\frac{q^2}{(p+q)^2}(E(0)+1) \\ E(0) = 0 & & \end{array}$$

In the above we can rearrange the first equation to obtain the following:

$$E(Game) \;=\; \tfrac{2pq}{(p+q)^2}E(Game) + \tfrac{p^2}{(p+q)^2}E(0) + \tfrac{q^2}{(p+q)^2}E(0) + 1$$

Definition 3.1 *The total output of a state* $T(s) = \sum\{p|s \xrightarrow{a[p]} s'\}$.

Definition 3.2 *Let* $Win \subseteq A$ *be a set of winning actions, and* $P(s_0, Win)$ *be the probability of observing an action in the set* Win *starting from state* s_0*, and the average number of ticks before an action in* Win *is observed* $D(s_0, win)$.

$P(Win, s_0)$ is the solution of the following set of simultaneous equations, for all $s \in V$

$$\begin{array}{lcl} P(s, Win) & \overset{def}{=} & \sum\{\frac{p_i}{T(s)}P(s', Win) | s \xrightarrow{a[p_i]} s', a \notin Win\} \\ & & +\sum\{\frac{p_j}{T(s)} | s \xrightarrow{a} a[p_j], a \in Win\} \end{array}$$

Similarly we can define $D(Win, s_0)$ to be the a solution of the following set of simulataneous equations:

[2]In general we obtain a set of simultaneous eqautions, one for each state.

$$\begin{aligned}
D(s, Win) &\overset{def}{=} \infty \text{ if } s \overset{a[p]}{\not\rightarrow} \\
D(s, Win) &\overset{def}{=} \sum\{\tfrac{p_i}{T(s)} D(s', Win) | s \overset{a[p_i]}{\rightarrow} s', a \notin Win\} \\
&\quad +1
\end{aligned}$$

Hence given a probabilistic transition graph with n states we can produce a set of n simultaneous equations which describe the probabilities and averages we are interested in. Generating the equations from the graph is straightforward and the equations can subsequently be solved by any symbolic mathematics package.

4 Finite State Non-terminating Systems

Consider the following process:

$$\begin{aligned}
W1 &\overset{def}{=} 6.sunny : W1 + 4.cloudy : W2 \\
W2 &\overset{def}{=} 5.cloudy : W2 + 5.sunny : W1
\end{aligned}$$

If we assume that the environment (of the process) is unbiased with respect to the *sunny* and *cloudy* actions then the above system can be represented by the following Markov [Kei, Paz71, Kle75, GS82] transition matrix:

$$\begin{pmatrix} 0.6 & 0.5 \\ 0.4 & 0.5 \end{pmatrix}$$

A question that is asked about the above system is with what probability is the action *sunny* seen. This question can be answered by knowing with what probability the system is likely to be in state $W1$ or $W2$ at an arbitrary time. In Markov chain theory this is known as the **stable distribution**[Kei,GS82] of the chain. For a chain whose transition matrix is $\underline{\underline{A}}$ then a stable distribution $\underline{v}$ is one which satisfies the following equation:

$$\underline{\underline{A}}\underline{v} = \underline{v}$$

and $|\underline{v}|$ is equal to 1.

In the case of the transition system above the stable distribution [Kei, Kle75,GS82] is given by the vector:

$$\begin{pmatrix} 5/9 \\ 4/9 \end{pmatrix}$$

and hence the probability of observing a *sunny* action is:

$$\begin{aligned}
P(sunny) &= \tfrac{5}{9}\tfrac{6}{10} + \tfrac{4}{9}\tfrac{5}{10} \\
&= \tfrac{5}{9}
\end{aligned}$$

Whilst in principle it is possible to convert a probability graph into its associated Markov chain and then solve for the stable distribution (by exploiting eigen theory) this is a highly inefficeint method of solving the problem. Given an n state probability transition graph the associated Markov chain matrix will be of size n^2. To represent the transition system as a matrix is clearly highly inefficient, and would prevent the consideration of systems composed of many components as their state space tends to grow exponentially in the number of components. An alternative manner of presenting the system is as follows. Remembering the original equation:

$$\underline{\underline{A}}\underline{\pi} = \underline{\pi}$$

by defining $r_i(\underline{\underline{A}})$ as the ith row of the matrix $\underline{\underline{A}}$ this is equivalent to solving the set of equations:

$$r_i(\underline{\underline{A}}).\underline{\pi} = \underline{\pi}_i$$

Whilst this would appear to still require $O(n^2)$ memory to represent the problem this is generally not the case. The probabilistic transition graphs that result,in practice, from process algebraic descriptions tend to be very sparse. On the whole very few of the states of a system are reachable from any particular state, in fact there is generally a (small) bound (k) on the number of permitted transitions from any particular state and therefore in this representation an amount $O(kn)$ of memory will be necessary to represent the solution.

We can define the necessary set of simultaneous[3] equations directly in terms of the original graph as follows:

$$\underline{\pi}_s \quad = \quad \{\sum \tfrac{p_j \underline{\pi}_j}{T(s_j)} | s_j \overset{a[p_j]}{\rightarrow} s\}$$

together with the condition that $\sum\{\underline{\pi}_s\} = 1$. the solution vector $\underline{\pi}$ being a stable distribution[4] for the transition system.

In this case the unstructured sparseness of the equation set makes the use of standard symbolic mathematical equation packages very inefficient. A sparse equation solver was written to directly solve sets of equations generated by the above. By solving equations in inverse order of their fan out a considerable speed up can be achieved. The system generates a back substitution list which can be evaluated using a symbolic mathematics package.

To calculate the mean occurence of an action, the probability of that action occuring at a particular state is multiplied by the probability of

[3]The set of equations derived for Markov transition matrix will *not* be independent [GS82] and hence an extra condition is neede to ensure a unique solution. This condition is derived from the definition that a probability distribution must sum to 1

[4]Care should be excersised when using stable distributions as their uniqueness is only guaranteed under restricted circumstances [Kei,GS82]

being in that state. For an exmaple of this form of calculation performed by our toolset see Example 1.1 in the Appendix.

Unfortunately, the solution of symbolic simultaneous equations requires NP-space in the number of equation to represent the solutions. In practice it appears that symbolic solutions are unfeasible for systems of more than about 30 states.

An alternative definition [GS82] of the stable distribution of a markov system is presented in the following fashion:

$$\underline{\pi_{i+1}} = \underline{A\pi_i}$$

with $\underline{\pi} = lim_{i \rightsquigarrow \infty} \underline{\pi}_i$ if there is a unique stable distribution.

Using the above definition we can define an iterative calculation over the probability transition graph in the following fashion:

$$\underline{\pi_{i+1}}_s = \{\sum \frac{p_j \underline{\pi_i}_j}{T(s_j)} | s_j \overset{a[p_j]}{\rightarrow} s\}$$

If an attempt is made to calculate an exact solution to the above iteration procedure then the same representation problem is encountered. However, it is possible to exploit the above method to provide an approximate solution (in the sense of a Taylor's expansion) to the stable vector problem. By truncating the $\underline{\pi_i}$ to a particular accuracy after each iteration of the calculation. If the terms (in a variable x say) are maintained to order k then, standard numerical solution of eigensystems theory [Wil65] shows that the solution will have an absolute error of $O(x^k)$.

Hence the following procedure can be exploited to give an approximate solution to the distribution problem, choose any non-zero length 1 $\underline{\pi_0}$[5]:

1. Compute $\underline{\pi_{i+1}}$ from $\underline{\pi_i}$;
2. truncate $\underline{\pi_{i+1}}$ to required accuracy k;
3. repeat from 1 until stability is achieved.

In practice one can compute a central approximation and then compute the further terms by increasing the approximation level steadily until the desired level is reached. An example of this solution method applied to the performance of the Alternating Bit Protocol can be found in the Appendix 1.2.

5 Conclusions

Whilst it is possible to verify the behaviour of a system by checking the process that describes it against another process[Mil80, Mil90, Chr90, JS90,

[5]In practice we use the vector $\underline{\pi_0}_i = \frac{1}{n}$ when the system has n states.

Tof90, SS90] or a predicate[Mil90, Han94, HJ94] this is often not the best approach. In many cases the intention of the design analysis is to determine how well a system can function which is why simulation[BDMN79, Bir79, Kre86, BFS87] is often resorted to. It is important in such circumstances to be able to identify the contribution of the underlying components to the overall system performance. The verify strategy works well when system requirements are known in advance but in many cases the design problem is one of: what is the best way of using these components to solve a particular problem? In this case the components are fixed, and we need to be able to derive the resulting systemic behaviour.

It might seem that we are not exploiting the algebraic properties of the process algebraic description in deriving our systemic properties. This is not the case. When the transition system for any process is computed within our tool we exploit the algebraic equivalences to try to produce as small a transition graph as possible. This does not necessarily produce a minimal system, but will exploit as much of the syntactic identity as made available in the description of the system provided. In practice this is only of value when the system contains repeated components, such as the two identical coins in our original example, where the number of states required to represent both coins can be reduced from 4 to 3 immediately. Over a large system these gains can significantly reduce the size of the states space, if we had six coins then we reduce the state space from 64 to 7.

Whilst it is true that for the majority of problems a symbolic approach to process representation will not admit a computationally feasible analytic solution, this approach still has major advantages. As we describe performance aspects of the system's components symbolically and construct its probability transition graph in terms of these symbols (a computationally costly operation, even if all of the transition probabilities are constants) it is subsequently possible to instantiate the graph with particular values of interest for the component's performance. Hence, with little extra computation cost, we can study the behaviour of our system under a wide range of conditions.

The generation of local approximations to the solutions of systems is of great importance. It has long been known that the behaviour of complex systems can critically dependent on the precise values of their parameters. In any real implementation of a system the true values of its components performances are liable to vary slightly from the exact values in our models. Local approximations allow us to assess the effect that these small variations may have on the systems true behaviour. For instance if performance could be heavily compromised by a small variance in one components performance it may be a good idea to redesign the system to be more tolerant or replace that component.

A sublanguage of WSCCS and the algorithms in this paper have been implemented as a set of SML functions (Probabilistic Algebra Tools set) which can be obtained from cmnt@cs.man.ac.uk. In terms of scale the exact solution generator can cope with systems of about 30 states, and will execute upon systems of this scale in 2 hours on a SPARCstation 2. The approximation method can cope with systems of 1000's of states and can take 24 hours to execute on such systems. Automatic scanning functions have been written to generate the piecewise approximations. For problems where there are no free weight parameters (all of the system constants are purely numerical) the tool set can successfully solve problems with 10000s of states.

6 REFERENCES

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

[BDMN79] G. Birtwistle, O-J Dahl, B. Myhrhaug and K. Nygaard, Simula Begin, 2nd Edition, Studentliteratur, Lund, Sweden, 1979.

[BFS87] P. Bratley, B. Fox and L. Schrage, A guide to simulation, second edition, 1987.

[Bir79] G. Birtwistle, DEMOS — discrete event modelling on Simula. Macmillen, 1979.

[BK84] J.A. Bergstra, J.W. Klop, The algebra of recursively defined processes and the algebra of regular processes, in Proc 11th ICALP, Springer LNCS 172, pp 82-85, 1984.

[CAM90] L. Chen, S. Anderson and F. Moller, A Timed Calculus of Communicating Systems, LFCS-report number 127

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

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

[CPS93] Cleaveland, R., J. Parrow and B. Steffen, The Concurrency Workbench: A Semantics-Based Tool for the Verification of Finite-State Systems, ACM Transactions on Programming Languages and Systems, 15(1):36–72, 1993.

[DLSB82] V.A. Dyck, J.D. Lawson, J.D. Smith and R.J. Beach, Computing: An Introduction to Structured Problem Solving Using Pascal: Reston, Reston, 1982.

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

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

[Han94] M.R. Hansen, Model checking discrete duration calculus, FACS 6A:826-845, 1994.

[Hen91] M. Hennessy, A proof system for CCS with value passing, FACS 3: 346-366.

[HJ94] H. Hansson and B. Jonsson, A Logic for Reasoning about Time and Reliability, FACS (6):512-535, 1994.

[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.

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

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

[Kin69] J.F.C. Kingman, Markov Population Processes, Journal of Applied Probability, 6:1-18, 1969.

[Kle75] L. Kleinrock, Queueing Systems, Volumes I and II, John Wiley, 1975.

[Kre86] W. Kreutzer, System Simulation, Addison Wesley, 1986.

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

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

[Mil80] R. Milner, Calculus of Communicating System, LNCS92, 1980.

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

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

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

[OW78] G. F. Oster and E. O. Wilson, Caste and Ecology in Social Insects, Princeton University Press, 1978.

[Paz71] A. Paz, Introduction to probabilistic automata, Academic Press, 1971.

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

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

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

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

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

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

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

[Tof90] C. Tofts, A Synchronous Calculus of Relative Frequency, CONCUR '90, Springer Verlag, LNCS 458.

[Tof93] C. Tofts, Exact Solutions to Finite State Simulation Problems, Research Report, Department of Computer Science, University of Calgary, 1993.

[Tof94] C. Tofts, Using Process Algebra to Describe Social Insect Behaviour, Transactions on Simulation, 1993.

[Tof94] C. Tofts, Processes with Probabilities, Priorities and Time, FACS 6(5): 536-564, 1994.

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

[VW92] S. F. M. van Vlijmen, A. van Waveren, An Algebraic Specification of a Model Factory, Research report, University of Amsterdam Programming research Group, 1992.

[Wil65] J. Wilkinson, The Numerical Eigenvalue Problem, Oxford University press 1965.

1 PRobabilistic Algebra Toolset (PRAT)

The basic process definition mechanism is to present a file in Edinburgh concurrency workbench [CPS93] like syntax. The system then generates an extended probabilistic transition graph (it takes account of priorities) and provides a set of analysis functions which can be applied to the system.

1.1 Weights

Weights are defined by the following syntax, where n is an integer and *string* an ascii character string:

$$e ::= n|\mathit{string}|e - e^{6}$$
$$w ::= e@n$$

So the following are weights: `5, 1-p, 5@2, 1-p@3`. the last two being weights at priority level 2 and 3 repsectively. Note that we do not allow symbolic priorities, as this would actually affect the computational structure.

1.2 Actions

Actions are defined as products of powers of strings;

$$A ::= \mathit{string}[\hat{\ } n]|A\#A$$

again we do not allow symbolic action powers.

So the following are actions: `a, a^-1,a#b^-2, c^4#a#b^3`.

To form a permission free group we provide a binding operator for sets of actions:

```
bs Set a,b,c
```

binds the name `Set` to the actions `a,b,c`.

1.3 Processes

We define the following constructions on processes which we present by example; it should be noted that we only allow one depth of operator application, this permits automatic absorbtion of equivalent state in parallel compositions:

[6]It should be noted that the current parser works LR so that $1 - p - q$ is actually $1 - p + q$

Sequential	`bs Coin p.head:C1 + 1-p.tail:C2`
Parallel	`bpa Sys S1|S2`
Permission	`bperm S1 Sys/Set`
Priority	`bpi Sn S`
Pri(Perm(Par))	`btr Sys S1|S2|S3/Set`
Perm(Par)	`bpc Sys S1|S2/Set`
Comment	`*this is a comment`

As the sytem constructs processes it prints out the number of states it has allocated so far as an indication of the work left to do.

1.4 Analysis

The following functions are presented to allow the maintenance, exploration and analysis of systems:

- `cle()` clear the current process environment.
- `rf(filename)` read a process definition from `filename`.
- `dupo(filename)` duplicate all output to the file.
- `co()` close duplicate output file.
- `sim(Pname)` simulate the process state called `Pname`. The simulator presents a menu of actions. Typing the number of the action causes the system to continue from the labelled state. Hitting return takes option 0 and hitting q exits the simulator.
- `fd(Pname)` find deadlocks in the process `Pname`. If a deadlock is found then the shortest transition to that state is printed.
- `ll(Pname)` find livelocks in the process `Pname`.
- `do_prob(Pname,Win,Lose)` generate a set of equations describing the probability of seeing the `Win` action, ignoring other actions but terminating on the `Lose` action. Action syntax as above.
- `do_mean(Pname,Win,Lose)` generate a set of equations describing the mean number of ticks to see a `Win` or `Lose` action, ignore all other actions.
- `ibv(vn,real)` bind the weight variable `vn` to the real value `real`.
- `sc(Pname)` generate a set of back substitutions for the stable distribution of process `Pname`.
- `solmn(Pname,action,file)` produce an expression for the mean number of `action` in process `Pname` output the results to `file`. This is separate from the above to allow reuse of the stable solution information.

- **genfun(Pname,Action,vn,low,hi,step,inR,gR,aL)** generate a piecewise approximation to the mean number of actions `Action` the parameters are as follows:

Pname	the process;
Action	the action;
vn	the variable name to range over;
low	lower limit of solution generation;
hi	upper limit of solution generation;
step	step between solutions;
inR	initial number of iterations at approx level 0, to generate coarse approximation;
gR	number of iterations at final approx level;
aL	required accuracy of the final answer.

- **genml(evrn,aprx,file)** generate an SML function to evaluate the piecewise approximation generated by above function, the error variable is given by `evrn` and `file` is the name of a file to copy the result to.

The system supplies two iterative solution packages, one for numeric solutions the other for local approximations, we describe their use below:

	Numerical	Analytical
Initialise	**start_iterate(Pname)**	**SI(Pname)**
Iterate	**itn(n)**	**APN(n)**
Print Sol	**pt()**	**CAP()**
Mean	**ma(action)**	**AM(action)**
Mean (Cyclic)	**mcyc(action,n)**	**APC(acs,n)**

In the above the `n` in the cyclic means is the number of states in the cycle. Often good approximations to nearly cyclic systems can be obtained cheaply by exploiting this function. A further function `sal(n)` is supplied to set the approximation level in the second set of functions.

As a first example we provide a system with an exact analytic solution.

Example 1.1 *Consider two processes competing for the same resource. Each process issues a request with probability* $1/p$ *after the last time it had the resource, and then will release the resource with probability* $1/q$ *at each instant.*

```
*Simple competition example

bs U1 p.t:U1G + 1-p.t:U1
bs U1G 1@1.get^-1:U1Got + 1.baulk:U1G
bs U1Got q.put^-1:U1 + 1-q.t:U1Got

bs Res 1.get:RG + 1.t:Res
bs RG 1.put:Res + 1.t:RG

basi C baulk

btr Sys U1|U1|Res/C
```

Having solved the equations the above system generates we obtain the average number of baulk actions seen at each tick is given by the following eqaution:

```
        2       3               -15          2          3
(8. p  - 4. p  - 4.44089 10     p q + 4. p  q - 8. p  q +

    4                   2 2 2
  4. p  q + 1. (2. - 2. p)  p  q ) /

       2       3                              -15  2          3
  (8. p  - 4. p  + 8. p q + 4.44089 10     p  q - 8. p  q +

    4          2        2 2       3 2       4 2
  4. p  q + 4. q  - 4. p  q  - 4. p  q  + 4. p  q )
```

As a result of using a real representation for numerical values in the toolset there are rounding errors[7] *in the above and if these are corrected we can obtain the following formula:*

$$\frac{p^2(4-2p+2q-4pq+4p^2q+(1-p)^2q^2)}{4p^2-2p^3+4p^3q-4p^3q+2p^4q+2q^2-2p^2q^2-2p^3q^2+2p^4q^2}$$

As a second example we demonstrate a performance analysis of the alternating bit protocol.

Example 1.2 *In [Mil90] Milner presents an implementation of the Alternating Bit Protocol in CCS, and demonstrates that the protocol is correct. Perforce this implementation ignores the temporal and probabilistic properties of the system and its components.*

Our alternating bit protocol realisation is depicted in Figure 1.

[7]Caused by the use of real numbers in the analysis system, easily identifiable as the terms are insignificant. The exponential growth of terms exhibited by products of processes forced the use of (truncated) real aritmetic in the tool, rather than the prefered (exact) integer arithmetic

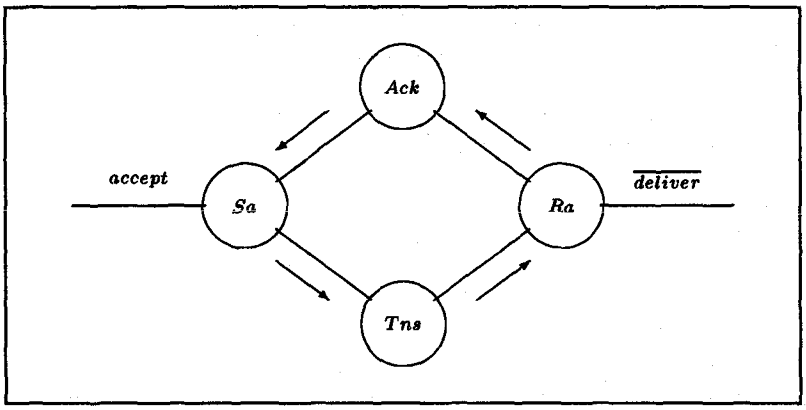

FIGURE 1. Alternating Bit Protocol

The process Sa will work in the following manner. After accepting a message, it sends it with bit b along the channel Tns and waits. Subsequently there are three possibilities:

- *it times out and retransmits the message;*
- *it gets an acknowledgement b from the Ack line (signifying a correct transmission), so that it can now accept another message;*
- *it gets an acknowledgement ¬b from the Ack line (signifying a superfluous extra acknowledgement of earlier message) which it ignores.*

The replier Ra works in a dual manner. After a message is delivered it sends an acknowledgement with bit b along the Ack line. Subsequently there are again three possibilities:

- *it times out, and retransmits the acknowledgement;*
- *it gets a new message with bit ¬b from the Tns line, which it delivers and acknowledges with bit ¬b;*
- *it gets a repetition of the old message with bit b which it ignores.*

We assume that messages are lost by the medium with probability err on each transmission. The sender and replier processes will retry with probability rt at for each tick whilst they are waiting for an acknowledgement or the message bit to change. In order that we can apply our peturbation theory to the variables err and rt, we assume perturbations of ere and rte upon their basic values.

```
*Probabilistic alternating bit protocol
*
*Chris Tofts 9/8/94 after CWB versions
*
```

```
*The basic sender process, note do everything asynchronously
*this should be Sa 1.send:Sa1 + 1.t:Sa

bs Sa 1.t:Sa1

*send out a signal as soon as possible

bs Sa1 1@1.s0^-1:Sa2 + 1.t:Sa1

*wait for acknowledgement to come through

bs Sa2 1.rack0:S1s + 1.rack1:Sa1 + rt-rte.t:Sa1 + 1-rt-rte.t:Sa2

*tell the world that it got through OK and invert sending bit

bs S1s 1.succ:S1

*the dual of the above system for sending with bit set to 1
*this should be bs S1 1.send:S11+ 1.t:S1

bs S1 1.t:S11
bs S11 1@1.s1^-1:S12 + 1.t:S11
bs S12 1.rack1:Sas + 1.rack0:S11 + rt-rte.t:S11 + 1-rt-rte.t:S12
bs Sas 1.succ:Sa

*the receiver for all of our endeavours....

bs Ra  1.r0:Rar1+1.r1:Ra2 + rt-rte.t:Ra2 + 1-rt-rte.t:Ra
bs Rar1 1.receive:Ra1

*try to send the data as quickly as possible

bs Ra1 1@1.sack0^-1:R1 + 1.t:Ra1 + 1.r0:R11 + 1.r1:Ra12
bs Ra2 1@1.sack1^-1:Ra + 1.t:Ra2 + 1.r1:R12 + 1.r0:Rar1
bs R1 1.r1:Ra12 + 1.r0:R11 + rt-rte.t:R11 + 1-rt-rte.t:R1
bs Ra12 1.receive:R12
bs R11 1@1.sack0^-1:R1 + 1.t:R11 + 1.r0:R11 + 1.r1:Ra12
bs R12 1@1.sack1^-1:Ra + 1.t:R12 + 1.r1:R12 + 1.r0:Rar1

*the lower channel for sending data on
*we send out data as soon as possible after the transmission
*time if it is not lost to error...

bs Ml1 1.s0:Ml1a0 + 1.s1:Ml110 + 1.t:Ml1

*decide if the data was transmitted OK, or was subject to error

bs Ml1a0 1-err-ere.t:Ml1a + err-ere.t:Ml1
bs Ml1a 1@1.r0^-1:Ml1 + 1.t:Ml1a
bs Ml110 1-err-ere.t:Ml11 + err-ere.t:Ml1
bs Ml11 1@1.r1^-1:Ml1 + 1.t:Ml11
```

```
*this is the transmitting medium for the return of the data

bs M12 1.sack0:M12a1 + 1.sack1:M1211 + 1.t:M12
bs M12a1 1-err-ere.t:M12a + err-ere.t:M12
bs M12a 1@1.rack0^-1:M12 + 1.t:M12a
bs M1211 1-err-ere.t:M121 + err-ere.t:M12
bs M121 1@1.rack1^-1:M12 + 1.t:M121

*That's all the sequential bits done so we can now have a go at putting
*it all together...

basi Allow send, receive, succ

*this is the complete system

btr ABP Ra|M11|M12|Sa/Allow
```

The PRAT tool can output a local approximation function to compute the average number of succ *actions observed per tick. For a value of* $rt = 0.5$ *and* $rte = 0.0$ *and* err *in the range* 0.05 *to* 0.35, *we obtain the following piecewise approximation for the average* succ *rate in the form of an SML function:*

```
fun ABP(vl)
=if 0.05<=vl andalso vl<=0.15
 then let val ere = (0.05+0.15)/2.0 - vl in
(26.2801394665775-285.522119242462 * ere * ere * ere
-94.9741260228557 * ere * ere + 26.6274947013716 * ere )/
(270.683502505543+1.32871491587139E~12 * ere * ere * ere -
7.47846229387505E~13 * ere -3.12638803734444E~13 * ere * ere )
end
else if 0.15<=vl andalso vl<=0.25
 then let val ere = (0.15+0.25)/2.0 - vl in
(11.7544078755176-83.86909484614 * ere * ere * ere
-16.6668138922172 * ere * ere +19.7005844688574 * ere )/
(139.000000000001-3.19744231092045E~13 * ere * ere * ere
- 4.01456645704457E~13 * ere -3.92574861507455E~13 * ere * ere )
end
else if 0.25<=vl andalso vl<=0.35
 then let val ere = (0.25+0.35)/2.0 - vl in
(9.68600554074613-41.7301838198085 * ere * ere * ere
+1.29897621781181 * ere * ere +20.937477181878 * ere )/
(139.0-1.59872115546023E~14* ere * ere * ere-
3.5438318946035E~13 * ere -2.46025422256935E~13 * ere * ere )
end
else 0.0;
```

As an example for a value of $err = 0.2$ *we obtain the result* 0.0845640854353778, *which equates (by inversion) to an average transmission time of* 11.8253510914415.

The total time to construct the process graph and the above approximations was about 1/2 hour on a SPARCstation 2.