Exact Solutions to Finite State Simulation Models

C. Tofts*
Department of Computer Science,
University College Swansea,
Swansea, SA2 8PP.
Wales, UK.
email: c.m.n.Tofts@swansea.ac.uk

October 30, 1993

Abstract

Many Monte Carlo simulation problems are essentially finite state. We demonstrate how such problems may be represented within a probabilistic process algebra. We demonstrate how it is possible to convert such processes into linear probabilistic equations, and solve such equations exactly with available computer algebra systems.

1 Introduction

One of the most common uses of simulation is to evaluate the behavior of complex probabilistic systems. Such problems naturally arise when considering the outcome of games and the performance of queues. The use of simulation to evaluate such systems is limited in several fashions. Firstly, particular values and/or distributions have to be chosen for any probabilities in the system to enable their evaluation. Secondly, the implementor has to choose a particular number of executions over which to sample. Thirdly, the accuracy of the final results is highly dependent on the randomness of the generated results. Fourthly, there is little or no way of evaluating whether the simulating program is an actual implementation of the desired problem.

For a large class of these problems, namely those with a finite number of states we propose a different approach. The system can be described in an appropriate process algebra, allowing the formal evaluation of system properties, and then automatically converted to an appropriate set of linear probabilistic equations. These equations then being solved by any standard computer algebra system. Such an approach allows us to generate exact solutions for problems whose state space is finite.

The process algebraic notion is both compact and managable. It allows us to repesent systems with large (exponential) numbers of states, with a small number of simple linear equations. These methdologies are therefore suitable both as notations, as a result of their compactness, and for reasoning, from their formal presentation.

We can approximate infinite state space problems, by restricting them to finite approximations,(which is what happens implicitly within any computer implementation anyway!) and consequently **know** how much of the state space we have explored. Importantly we can the exploit our

*This work is supported by an SERC Advanced Fellowship. The author gratelfully acknowleges funds to visit the University of Calgary from an operating grant given by NESRC, Canada to Prof. G. Birtwistle.

methodology to detect just how much of the potential state space we have successfully explored, and enlarge it if the approximation is too gross.

Evidently, any finite set of experiments on an infinite state space system will only explore some sub-space of that system, but in a simulation we would be unaware of which. Indeed our simulation may well only explore the most likely areas of the graph, possibly neglecting extreme cases.

The process algebra we choose to exploit in describing our systems is the *Weighted Synchronous Calculus of Synchronous Systems* (WSCCS) [Tof90,Tof93a,Tof93b], a derivative of Milner's Synchronous Calculus of Communicating Systems, [Mil83,Mil89]. WSCCS is highly expressive allowing us to describe probabilistic and priority constraints on the choices a process makes about its future behavior. As it is a synchronous calculus it has a simple notion of time which can be exploited to understand the timing properties of systems described in WSCCS. This calculus has been exploited to derive the properties of large scale and complex problems in the field of behavioral ecology [Tof90b, Tof91, Tof93a], formal results were obtained over systems that were traditionally simulated.

To realise the translation of our processes into probabilistic graphs, and the translation of those graphs into simultaneous equation problems, we use an SML program. The translation program currently produces output suitable for solution by the MAPLE(TM) system. Maple is a 'standard' computer algebra system. Currently our implementations are limited to work with probabilities of fixed values, but generalising the current implementation to work with arbitrary probability values is a straightforward task which will shortly be undertaken.

In this paper we present some simple motivating examples of our approach, as an informal introduction to WSCCS, are given in Section 2. We present some **exact** solution to traditional Monte Carlo simulation exercises in Section 3, which demonstrate the scale of the systems addressable. We examine the results of using our methods to approximate systems with an infinite numbr of states in Section 4. Some queue theoretic examples are studied in Section 5. For completeness, a full description of the semantics and equational theory of WSCCS is included in Appendix A.

2 Introductory Examples

A full description of a process algebra (SCCS) with the underlying computational structure of WSCCS can be found in Milner's excellent book [Mil89].

As an introduction to the WSCCS process language, consider the following simple game. Two coins are tossed: if they both come up heads then a gambler will win; if they both come up tails then the gambler loses; on the other outcome the gambler plays again. We can describe this simple system by the following WSCCS processes:

$$\begin{aligned}
Coin &\overset{def}{=} 1.'head : Coin + 1.'tail : Coin\\
TwoCoins &\overset{def}{=} Coin \times Coin\\
Gambler &\overset{def}{=} 1.head^2\#win : Nil + 1.tail^2\#lose : Nil + 1.head\#tail.Gambler\\
Game &\overset{def}{=} (Gambler \times TwoCoins)\lceil\{win, lose\}
\end{aligned}$$

In the above the process *Coin* can perform either of the actions *'head* or *'tail* becoming the process *Coin* again. We understand such recursive definitions as unfoldings, copying the body of the process given on the right of the definition whenever it is needed. We have weights 1 on each of the transitions suggesting that they have equal weight. Weights are used as this avoids the need to normalize when we limit the behaviours of our processes. Weights can be interpreted as probabilities by totaling the weight out of any state; then the probability that any descendant process will be reached is the weight with which that process is labeled, over the total weight from the initial process. The plus (+) operator is intended to mean choice between possible future behaviors. Unlike the choice of CCS and SCCS, this plus is **not** simply absorbative. When two choices are indentical and we wish to combine them, we must sum the weights with which those choices can be made, this conserves our ability to interpret weights as probabilities.

The process $TwoCoins$ is two $Coin$ processes executing in *synchronous* parallel. For this process to perform an action both of its components must perform some action, and the parallel process will perform the composition of the actions. We use the operator # to denote the parallel composition of actions, and assume that it is both associative and commutative; these assumptions are natural as # represents a notion of parallelism which is generally assumed to be both associative and commutative. If we expand our $Twocoins$ process, we find that it has the following equivalent description:

$$\begin{aligned} Twocoins \;=\; & 1.'head\#'head.Twocoins \\ & +1.'tail\#'head.Twocoins \\ & +1.'head\#'tail.Twocoins \\ & +1.'tail\#'tail.Twocoins \end{aligned}$$

In the above we have unfolded the definition of $Twocoins$ once, and allowed ourselves to make the obvious syntactic identifications. The weight of any state is formed by multiplying the weights of the two transitions that produced it from the component processes. As # is commutative we have $'head\#'tail =' tail\#'head$.

So if we use the above identification we can reduce the $Twocoins$ process to the following:

$$\begin{aligned} TwoCoins \;=\; & 1.'head^2.Twocoins \\ & +2.'tail\#'head.Twocoins \\ & +1.'tail^2.Twocoins \end{aligned}$$

Notice that the mixed result has weight 2 and is thus twice as likely to come up as either of the others. So our combination of two fair coins has the correct probabilities for each of its possible outcomes.

In order that processes within our calculus can communicate with each other we divide actions into two kinds, **input** actions which we write as (say) *head*, and **output** actions which we write as *'head*. When we multiply two of these together with our parallel action composer #, they combine together and cancel forming a $\surd$ or 'tick' action (the output of one process has been read and absorbed by the other). The *Gambler* process interacts with the *TwoCoin* process by unbaisedly accepting its output. Thus the action $head^2\#win$ can combine with the action $'head^2$ to form the action win, similarly for the other actions of the *Gambler*. Throughout most of the remainder of the paper we shall denote an output a action by $'a$ rather than $'a$, this results for importing the syntax of mechanically presentable processes. The formal statement of the properties of the # operator on actions is given in Appendix A.

The *Gambler* process wishes to sample the coins, so it performs input actions dual to the outputs of the coins, plus two witness actions win and *lose* that allow us to see the result. Remember the communication on the coin actions means that they vanish. All of the gamblers choices are guarded by 1 weights as they have no bias (in reality is *allowed* no bias), towards any particular action.

Finally we need to ensure that actions such as *win#'tail#head*, do not occur, since they are malformed. That is we wish to ensure that only communications occur on the head and tail actions between the *Gambler* process and the $Twocoins$ process. Since it is this interaction that is supposed to dictate the gambler's chances of winning. The operator $\lceil\{win, lose\}$ states that only actions which are formed as a product of input or output win or *lose* actions **only** are permitted to occur.

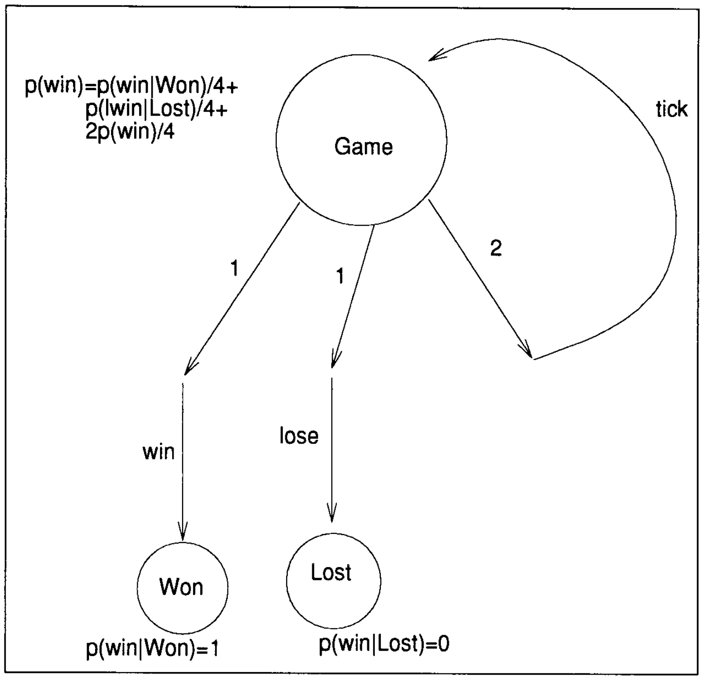


Figure 1: Transition diagram for the simple game process.

Notice that the $\surd$ action can occur since it is the same as $'win\#win$, and this is a product of *win* actions. Since neither the *head* or *tail* action is permitted our example erroneous action is now forbidden. To form the permitted part of a process we simply delete all weight paths leading to subprocesses that are **never** permitted to perform any action.

Thus we can see that our *Game* system is the following process:

$$\begin{aligned} Game &= 1.win : Nil + \\ &\quad 1.lose : Nil + \\ &\quad 2.\surd : Game \end{aligned}$$

i.e., we are only left with the three choices, in which either a tick is performed or a *win* or a *lose*.

Alternatively we can view our processes as graphs, see Figure 1. If we wished to compute the probability of winning such a game we would have to solve the following set of linear equations:

$$\begin{aligned} p(Game) &= (1.p(Win : Nil) + 1.p(lose : Nil) + 2.p(Game))/4 \\ p(win : Nil) &= 1 \\ p(lose : Nil) &= 0 \end{aligned}$$

To solve the set of equations generated from our processes we used the computer algebra system MAPLE. Like many similar software systems it can solve large systems of simultaneous equations. Obviously any computer algebra system would be suitable for our purposes, but there are benefits if it can handle large numbers of sparse simultaneous equations rapidly.

For such a simple problem it is straightforward to see the solution to the above equations. Indeed we see that the probability of winning is $\frac{1}{2}$ as we expect. Whilst this example is simple it demonstrates the basis of a general principle. If we convert a process to sum normal form and assign success (probability 1) or fail (probability 0) to the terminal nodes of its graph, the weights can be used to generate a set of linear equations. The solution, of this set of linear equations, tells us the probability of reaching the terminal nodes we labeled as successful. The conversion of WSCCS processes with a finite number of states into a weighted transition graph and its subsequent conversion into a set of linear equations has been automated as an SML program.

A related question is the number of times the coins would have to be tossed before the game was completed, this again can be represented as the solution of the following linear equations:

$$\begin{aligned} d(Game) &= (1.d(Win : Nil) + 1.d(lose : Nil) + 2.d(Game))/4 + 1 \\ d(win : Nil) &= 0 \\ d(lose : Nil) &= 0 \end{aligned}$$

It is straightforward to demonstrate that the solution of the above equations is $d(Game) = 2$. Again the activity of converting a process into a set of linear equations which specify its expected lifetime has been automated as an SML program.

There is a further expressive capability within the language WSCCS. We can define when one (or more) of our choices has priority over the others. For example, a cheating gambler might only want to see the occurrence of $'heads^2$ actions. Such a system could be described as follows:

$$Cheat \overset{def}{=} \omega.head^2\#win : Nil + 1.tail^2\#lose : Nil + 1.head\#tail.Cheat$$

the ω weight represents an infinite valued weight and the above process is understood as meaning, that whenever the action $head^2$ is offered by an environment then it should be taken in preference to any other choice. Much like our permission operator, we need an operator to tell processes to take account of priority information. We write $\Theta(P)$ to mean take the highest priority part of the process P. When this operator is used the weight of the highest prioritised parts of P are reduced to appropriate natural value weights (see Appendix A for a full account of weight arithmetic). We allow ourselves to write powers of ω to denote relatively higher levels of priority. Thus:

$$Rcheat \overset{def}{=} \omega^2.head^2\#win : Nil + \omega.head\#tail.Rcheat + 1.tail^2\#lose : Nil$$

is a cheating gambler, who if the environment will not let them win at this turn, will perform a *head#tail* in preference to taking a loss. This gambler will only lose if both coins will produce only tail actions. We compose weights and priorities in the obvious fashion, i.e., by multiplication and addition of powers where appropriate. Conflicts of priority lead to probabilistic outcomes, as two objects with equal priority will get equal weights when the priority operator is applied.

Notice, that it is possible to condition events both on input **and** on output. This can lead to a conflict when trying to interpret probabilities. The process descriptions require great care if neither of the input or output actions occur within an unbaised probabilistic sum (that is one where all the weights are 1). For an example of the resolution of such a problem see Section 3.1.

2.1 Craps

As a more substantial example we present a non-standard version of the dice game craps.Our rules are that on the first roll of a pair of die the 'shooter' wins if they roll a 7 or a 12; they lose if they roll a 2 or a 11; on any other score they continue rolling until they either roll they same score and win, or roll a 7 and lose.

This game can be formulated as the following WSCCS processes:

$$Dice \overset{def}{=} 1.'two.Dice + 2.'three.Dice + 3.'four.Dice + 4.'five.Dice + 5.'six.Dice + 6.'seven.Dice + 5.'eight.Dice + 4.'nine.Dice + 3.'ten.Dice + 2.'eleven.Dice + 1.'twelve.Dice$$

The above process describes rolling a pair of die. Each of the output actions is the total value of the roll, and the weights with which they are associted are those for a pair unbaised dice. The following process describes the initial state of the game, in which the gambler wins immediately if the dice roll is 7 or 12, loses if it is 2 or 11, and enters a state dependent on the roll value in any other case.

$$Gam \overset{def}{=} 1.two\#lose.Nil + 1.eleven\#lose.Nil + 1.seven\#win.Nil + 1.twelve\#win.Nil + 1.three.Gam3 + 1.four.Gam4 + 1.five.Gam5 + 1.six.Gam6 + 1.eight.Gam8 + 1.nine.Gam9 + 1.ten.Gam10$$

In the above the states $Gam3$ to $Gam6$ and $Gam8$ to $Gam10$, are thos of the gambler needing to 'make his point', of the dice roll of the same value. The state below is an example of the gambler needing to throw a 3 to win:

$$Gam3 \overset{def}{=} 1.seven\#lose.Nil + 1.three\#win.Nil + 1.two.Gam3 + 1.four.Gam3 + 1.five.Gam3 + 1.six.Gam3 + 1.eight.Gam3 + 1.nine.Gam3 + 1.ten.Gam3 + 1.eleven.Gam3 + 1.twelve.Gam3$$

above if the rool is a 3 the gambler wins, if it is a 7 then the gambler loses, otherwise the dice are rolled again. The following states are identical to the above, except that a different value of the roll is required for the gambler to make thei point.

$$Gam4 \overset{def}{=} 1.seven\#lose.Nil + 1.four\#win.Gam4 + 1.three.Gam4 + 1.two.Gam4 + 1.five.Gam4 + 1.six.Gam4 + 1.eight.Gam4 + 1.nine.Gam4 + 1.ten.Gam4 + 1.eleven.Gam4 + 1.twelve.Gam4$$

$$Gam5 \overset{def}{=} 1.seven\#lose.Nil + 1.five\#win.Nil + 1.two.Gam5 + 1.four.Gam5 + 1.three.Gam5 + 1.six.Gam5 + 1.eight.Gam5 + 1.nine.Gam5 + 1.ten.Gam5 + 1.eleven.Gam5 + 1.twelve.Gam5$$

$$Gam6 \overset{def}{=} 1.seven\#lose.Nil + 1.six\#win.Nil + 1.two.Gam6 + 1.four.Gam6 + 1.five.Gam6 + 1.three.Gam6 + 1.eight.Gam6 + 1.nine.Gam6 + 1.ten.Gam6 + 1.eleven.Gam6 + 1.twelve.Gam6$$

$$Gam8 \overset{def}{=} 1.seven\#lose.Nil + 1.eight\#win.Nil + 1.two.Gam8 + 1.four.Gam8 + 1.five.Gam8 + 1.six.Gam8 + 1.three.Gam8 + 1.nine.Gam8 + 1.ten.Gam8 + 1.eleven.Gam8 + 1.twelve.Gam8$$

$$Gam9 \overset{def}{=} 1.seven\#lose.Nil + 1.nine\#win.Nil + 1.two.Gam9 + 1.four.Gam9 + 1.five.Gam9 + 1.six.Gam9 + 1.eight.Gam9 + 1.three.Gam9 + 1.ten.Gam9 + 1.eleven.Gam9 + 1.twelve.Gam9$$

$$Gam10 \overset{def}{=} 1.seven\#lose.Nil + 1.ten\#win.Nil + 1.two.Gam10 + 1.four.Gam10 + 1.five.Gam10 + 1.six.Gam10 + 1.eight.Gam10 + 1.nine.Gam10 + 1.three.Gam10 + 1.eleven.Gam10 + 1.twelve.Gam10$$

Finally we can form the craps game by composing the gambler and the dice processes together in parallel, and forcing communication on the outputs of the die. It should be noted that in all of the gambler processes there is **no** bias towards any of the outcomes.

$$Game \overset{def}{=} (Gam \times Dice)\lceil\{win, lose\}$$

A weighted transition graph for the above system is presented in Figure 2. We have labeled states with the equation numbers from the system below which relate to that particular state. Some of the states are labeled twice (our state equivalence checker is very simple) but this will not affect the outcome of the arithmetic.

We can reduce the probability of winning to the following set of equations, these are wrapped in the appropriate syntax to induce MAPLE to solve them, the value we are interested in is that obtained for p16.

```
solve({p0=(2*p1+4*p1+2*p1+4*p1+5*p1+3*p1+1*p1+1*p1+6*p13+3+5*p1+0)/36,
p1=(2*p0+4*p0+4*p0+2*p0+5*p0+3*p0+1*p0+5*p0+3+1*p0+6*p12+0)/36,
p2=(2*p3+4*p3+2*p3+4*p3+5*p3+3*p3+1*p3+1*p3+6*p13+5+3*p3+0)/36,
p3=(2*p2+4*p2+4*p2+2*p2+5*p2+3*p2+1*p2+5+3*p2+1*p2+6*p12+0)/36,
p4=(2*p5+4*p5+4*p5+5*p5+1*p5+2*p5+1*p5+6*p13+3+3*p5+5*p5+0)/36,
p5=(2*p4+4*p4+4*p4+5*p4+1*p4+3+5*p4+3*p4+1*p4+2*p4+6*p12+0)/36,
p6=(2*p7+4*p7+5*p7+3*p7+1*p7+2*p7+1*p7+6*p13+4+3*p7+5*p7+0)/36,
p7=(2*p6+4*p6+5*p6+3*p6+1*p6+5*p6+3*p6+1*p6+2*p6+4+6*p12+0)/36,
p8=(2*p9+4*p9+5+4*p9+2*p9+3*p9+1*p9+1*p9+6*p13+3*p9+5*p9+0)/36,
p9=(2*p8+4*p8+4*p8+2*p8+5+3*p8+1*p8+5*p8+3*p8+1*p8+6*p12+0)/36,
p10=(2*p11+4*p11+2+4*p11+5*p11+3*p11+1*p11+1*p11+6*p13+3*p11+5*p11+0)/36,
p11=(2*p10+4*p10+4*p10+2+5*p10+3*p10+1*p10+5*p10+3*p10+1*p10+6*p12+0)/36,
p12=(0),
p13=(0),
p14=(2*p15+4*p15+5*p15+3*p15+1*p15+1*p15+6*p13+4+3*p15+5*p15+2*p15+0)/36,
p15=(2*p14+4*p14+5*p14+3*p14+1*p14+5*p14+3*p14+1*p14+2*p14+6*p12+4+0)/36,
p16=(4*p15+2*p11+5*p9+7+4*p7+3*p5+5*p3+3*p1+3*p13+0)/36,
p=0
},
{p0,p1,p2,p3,p4,p5,p6,p7,p8,p9,
p10,p11,p12,p13,p14,p15,p16,p});
```

Similarly we can produce the following equations to represent the average duration of the game, and the value we are interested in is d16.

```
solve({d0=(2*d1+4*d1+2*d1+4*d1+5*d1+3*d1+1*d1+1*d1+6*d13+3*d1+5*d1+0)/36+1,
d1=(2*d0+4*d0+4*d0+2*d0+5*d0+3*d0+1*d0+5*d0+3*d0+1*d0+6*d12+0)/36+1,
d2=(2*d3+4*d3+2*d3+4*d3+5*d3+3*d3+1*d3+1*d3+6*d13+5*d13+3*d3+0)/36+1,
d3=(2*d2+4*d2+4*d2+2*d2+5*d2+3*d2+1*d2+5*d12+3*d2+1*d2+6*d12+0)/36+1,
d4=(2*d5+4*d5+4*d5+5*d5+1*d5+2*d5+1*d5+6*d13+3*d13+3*d5+5*d5+0)/36+1,
d5=(2*d4+4*d4+4*d4+5*d4+1*d4+3*d12+5*d4+3*d4+1*d4+2*d4+6*d12+0)/36+1,
d6=(2*d7+4*d7+5*d7+3*d7+1*d7+2*d7+1*d7+6*d13+4*d13+3*d7+5*d7+0)/36+1,
d7=(2*d6+4*d6+5*d6+3*d6+1*d6+5*d6+3*d6+1*d6+2*d6+4*d12+6*d12+0)/36+1,
d8=(2*d9+4*d9+5*d13+4*d9+2*d9+3*d9+1*d9+1*d9+6*d13+3*d9+5*d9+0)/36+1,
d9=(2*d8+4*d8+4*d8+2*d8+5*d12+3*d8+1*d8+5*d8+3*d8+1*d8+6*d12+0)/36+1,
d10=(2*d11+4*d11+2*d13+4*d11+5*d11+3*d11+1*d11+1*d11+6*d13+3*d11+5*d11+0)/36+1,
d11=(2*d10+4*d10+4*d10+2*d12+5*d10+3*d10+1*d10+5*d10+3*d10+1*d10+6*d12+0)/36+1,
d12=(0),
d13=(0),
```

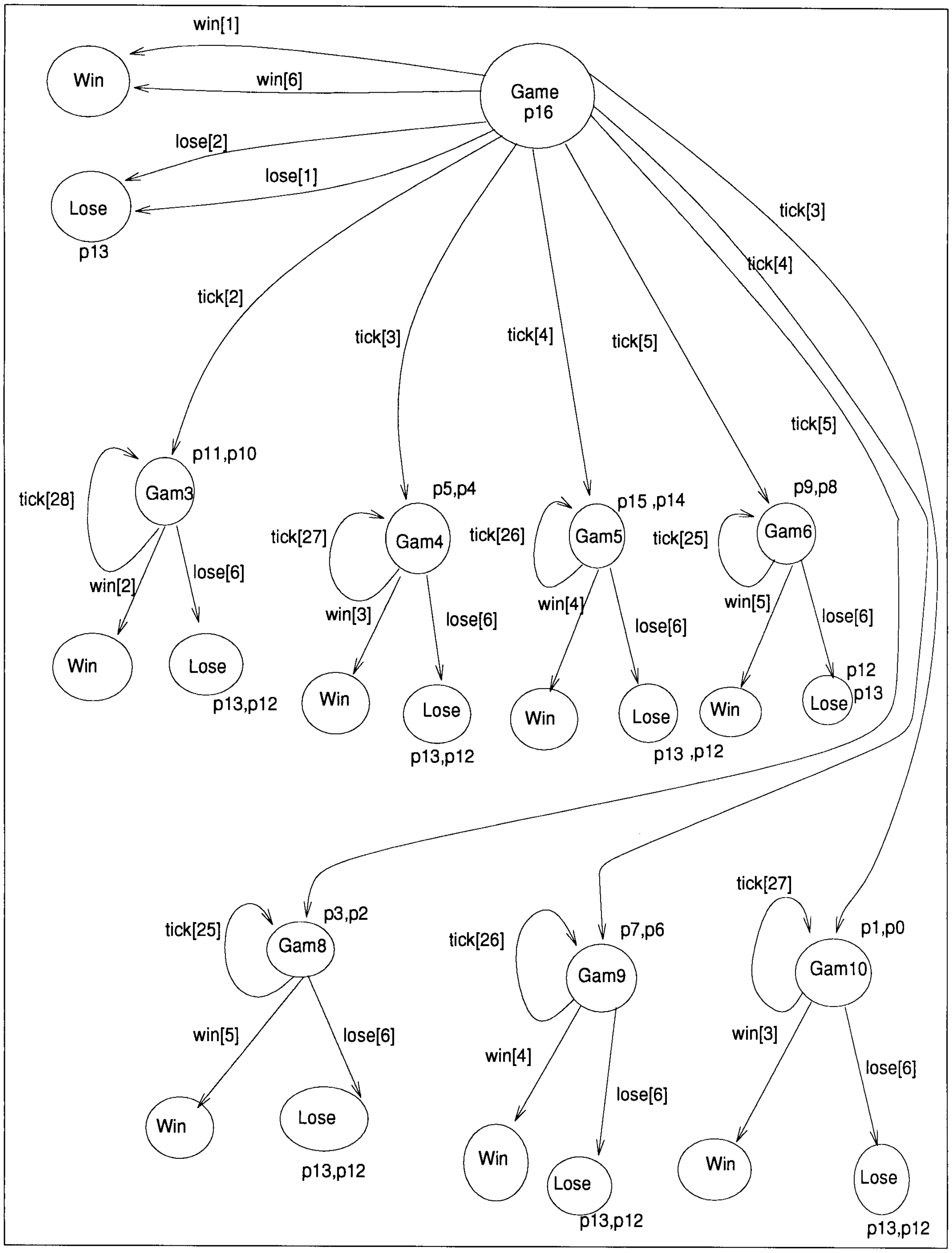


Figure 2: Transition system for the game of craps.

```
d14=(2*d15+4*d15+5*d15+3*d15+1*d15+1*d15+6*d13+4*d13+3*d15+5*d15+2*d15+0)/36+1,
d15=(2*d14+4*d14+5*d14+3*d14+1*d14+5*d14+3*d14+1*d14+2*d14+6*d12+4*d12+0)/36+1,
d16=(4*d15+2*d11+5*d9+7*d13+4*d7+3*d5+5*d3+3*d1+3*d13+0)/36+1,
d},
{d0,d1,d2,d3,d4,d5,d6,d7,d8,d9,
d10,d11,d12,d13,d14,d15,d16,d});
```

If we use the above as input to a MAPLE system. Then we discover that the probability of winning, given by p16 is $\frac{1897}{3960}$ (0.4790) and the average duration of the game, given by d16, is $\frac{2503}{660}$ rolls of the dice (3.8). Even for this relatively simple system we already have a number of equations that would make it extremely tedious to solve by hand, and liable to error.

2.2 Play Offs

At the end of the season in both baseball and ICE-hockey, there are seven match play offs. The difference between the two is, that in baseball a sequence of matches is played hhaaahh (h is home, a is away), and in hockey the sequence is hhaahah. Given that a teams chances of winning at home is 55% and symmetrically of winning away 45%. What is the chance of the team that starts playing at home winning the series, and is it dependent on the game order.

We can represent the baseball playoff by the following collection of WSCCS processes:

$$\begin{aligned} HomeG &\stackrel{def}{=} 45.'loseh.HomeG + 55.'winh\#win.HomeG + 1.\surd.HomeG \\ AwayG &\stackrel{def}{=} 45.'wina\#win.AwayG + 55.'losea.AwayG + 1.\surd.AwayG \end{aligned}$$

The two processes above describe the porbability of winning when playing either at home or away. The $\surd$ action will only be relevant when the game is of the other type, since our permission will ensure that it would have to combine with something that is not permitted when the game is of the appropriate type. We need a simple counter process to keep track of the number of wins:

$$\begin{aligned} Count &\stackrel{def}{=} 1.'win.Count1 + 1.\surd.Count \\ Count1 &\stackrel{def}{=} 1.'win.Count2 + 1.\surd.Count1 \\ Count2 &\stackrel{def}{=} 1.'win.Count3 + 1.\surd.Count2 \\ Count3 &\stackrel{def}{=} 1.champ\#'win.Nil + 1.\surd.Count3 \end{aligned}$$

The counter produces the action *champ* if the team achieves four wins, and then stops any further games. Finally, we need a process which goes through seven games in the appropriate sequence of aways and homes:

$$\begin{aligned} BB1 &\stackrel{def}{=} 1.winh.BB2 + 1.loseh.BB2 \\ BB2 &\stackrel{def}{=} 1.winh.BB3 + 1.loseh.BB3 \\ BB3 &\stackrel{def}{=} 1.wina.BB4 + 1.losea.BB4 \\ BB4 &\stackrel{def}{=} 1.wina.BB5 + 1.losea.BB5 \\ BB5 &\stackrel{def}{=} 1.wina.BB6 + 1.losea.BB6 \\ BB6 &\stackrel{def}{=} 1.winh.BB7 + 1.loseh.BB7 \\ BB7 &\stackrel{def}{=} 1.winh.Nil + 1.loseh.Nil \end{aligned}$$

We can now form a process representing a playoff series by combining the two froms of game together with the home/away generator, and passing the *win* actions to the counter, to evaluate whether the total will pass 4 or not.

$$\begin{aligned}
Gm &\stackrel{def}{=} (HomeG \times AwayG) \\
BBC &\stackrel{def}{=} (BB1 \times Gm) \\
BBC1 &\stackrel{def}{=} BBC\lceil\{win\} \\
BBS &\stackrel{def}{=} (BBC \times Count)\lceil\{champ\}
\end{aligned}$$

We discover that the probability of winning is $\frac{33006061}{64000000}$ or 51.57%.

We can present our ice-hockey play offs as the following process, and substitute it for $BB1$ in the system above.

$$\begin{aligned}
H1 &\stackrel{def}{=} 1.winh.H2 + 1.loseh.H2 \\
H2 &\stackrel{def}{=} 1.winh.H3 + 1.loseh.H3 \\
H3 &\stackrel{def}{=} 1.wina.H4 + 1.losea.H4 \\
H4 &\stackrel{def}{=} 1.wina.H5 + 1.losea.H5 \\
H5 &\stackrel{def}{=} 1.winh.H6 + 1.loseh.H6 \\
H6 &\stackrel{def}{=} 1.wina.H7 + 1.losea.H7 \\
H7 &\stackrel{def}{=} 1.winh.Nil + 1.loseh.Nil
\end{aligned}$$

We discover that the probability of winning a hockey series is $\frac{33006061}{64000000}$, that is exactly the same as for the baseball series. However, if we look at the mean duration of the series. We discover that the hockey series has mean duration $\frac{40942069}{6400000}$ or 6.397198, and the baseball series $\frac{41102053}{6400000}$ or 6.422196. We investigated other patterns of home and away games and found that the baseball pattern had the maximum mean number of games.

3 Problems from [Kre86]

Since [Kre86] is a well known and respected text book in the field of simulation, we present solutions to the problems he sets as Monte Carlo simulation exercises in Chapter 5. Our solutions will consist of processes which describe the problems, and then solving for appropriate probabilities that describe the systems behavior.

3.1 Pixie Airways

Albatross airways runs 15 flights a day. On each flight a pixie is required as an attendant. It is airline policy to keep three pixies on reserve to replace other pixies if they fall sick. In the original presentation of this problem [Kre86, pp224], no maximum is given for the number of available pixies. We have chosen that number to be 18, as this is the number of flights plus the intended number of reserves. On any given day pixies will fall ill with the following probabilities:

Number sick	Probability
0	0.20
1	0.25
2	0.20
3	0.15
4	0.10
5	0.10
≥ 6	0

We model the pixies as a single process, which will provide a number of *'pix* actions depending on how many of the pixies are ill. It will also provide three *'rpix* actions which describe an attempt to reserve 3 pixies. Flights attempt to synchronize with a *pix* action to get a non-reserved pixy. If that fails, then it will take one of the reserves and record the fact, by performing a *reserve* action. If neither of the above is possible then the flight cannot take off and this will be recorded by the performance of a *nogo* action. We split the behaviour over two ticks, on the first a probabilistic decision is made on the number of available pixies; on the second the assignment of pixies to flights is made.

$$\begin{aligned} Pix &\overset{def}{=} 20.\surd.`pix^{15}\#`rpix^3 : Pix \\ &+25.\surd.`pix^{14}\#`rpix^3 : Pix \\ &+20.\surd.`pix^{13}\#`rpix^3 : Pix \\ &+15.\surd.`pix^{12}\#`rpix^3 : Pix \\ &+10.\surd.`pix^{11}\#`rpix^3 : Pix \\ &+10.\surd.`pix^{10}\#`rpix^3 : Pix \end{aligned}$$

In the above the $\surd$ actions remove problems with conditioning both on input and output. The weights of the respective states are thos given in the table.

$$Flight \overset{def}{=} \surd.(\omega^2.pix.Flight + \omega.rpix\#reserve.Flight + 1.nogo.Flight)$$

As stated the flight uses priorities to establish its preferences for a pixy over a resrve pixy, and a reserve over none. We need to run 15 flights, so we shall take 15 copies of the flight process in parallel, which is abbreviated by $Flight^{15}$. Composing the flights and the pixy system together, and only allowing the actions recording whether a reserve was used, or the flight was unable to take off, we form the following process:

$$Airway \overset{def}{=} \Theta((Pix \times Flight^{15})\lceil\{reserve, nogo\})$$

To understand the behavior of this system we need to know with what frequency reserve pixies will be used. In other words, how often we see *reserve*, $reserve^2$ or $reserve^3$ actions, and how often flights are unable to take off as a result of too many pixies being unwell; witnessed by the performance of *nogo* actions.

Is is straightforward (using the equational theory) to demonstrate that:

$$\begin{aligned} Airway &= 20.\surd.\surd.Airway \\ &\quad +25.\surd.\surd.Airway \\ &\quad +20.\surd.\surd.Airway \\ &\quad +15.\surd.\surd.Airway \\ &\quad +10.\surd.reserve.Airway \\ &\quad +10.\surd.reserve^2.Airway \qquad \text{(Simplifying)} \\ &= 80.\surd.\surd.Airway \\ &\quad +10.\surd.reserve.Airway \\ &\quad +10.\surd.reserve^2.Airway \end{aligned}$$

In other words 80% of the time no reserves are needed, 10% of the time one of the reserves is need, and 10% of the time 2 reserves are needed. On no occasion with this pattern of illness amongst the pixies will a flight be unable to take off.

3.2 Silly Beetle

The drunken beetle problem. The beetle performs a random walk on a square grid of size 10×10 with death resulting if it falls off the grid. The game is played by making 10 movements and then the beetle lives if it has not fallen off the grid. To simplify state counting, we decompose the position of the beetle into horizontal and vertical coordinates and represent each coordinate with a simple processes. The movement over the grid is given by a separate process, which drives the beetle along one of the four axes. The time limit is provided by a simple process that gives 9 ticks and then describes the beetle as living. If the beetle were to die on the tenth move then the system output would be the action *live#die* which we shall assume means the beetle has died. We believe the beetle to have lived only if we see the atomic action *live*.

$$\begin{aligned}
B1H &\stackrel{def}{=} 1.die\#left.Nil + 1.right.B2H + 1.\surd.B1H \\
B2H &\stackrel{def}{=} 1.left.B1H + 1.right.B3H + 1.\surd.B2H \\
B3H &\stackrel{def}{=} 1.left.B2H + 1.right.B4H + 1.\surd.B3H \\
B4H &\stackrel{def}{=} 1.left.B3H + 1.right.B5H + 1.\surd.B4H \\
B5H &\stackrel{def}{=} 1.left.B4H + 1.right.B6H + 1.\surd.B5H \\
B6H &\stackrel{def}{=} 1.left.B5H + 1.right.B7H + 1.\surd.B6H \\
B7H &\stackrel{def}{=} 1.left.B6H + 1.right.B8H + 1.\surd.B7H \\
B8H &\stackrel{def}{=} 1.left.B7H + 1.right.B9H + 1.\surd.B8H \\
B9H &\stackrel{def}{=} 1.left.B8H + 1.right.B10H + 1.\surd.B9H \\
B10H &\stackrel{def}{=} 1.left.B9H + 1.die\#right.Nil + 1.\surd.B10H
\end{aligned}$$

The above simple counter process records the horizontal position of the beatle, it's movements being directed by *left* and *right* actions. If the horizontal position required is 10 or 0, then the fact the beatle has died is recorded by the performance of the *die* action. A similar counter process records the vertical position of the beatle:

$$\begin{aligned}
B1V &\stackrel{def}{=} 1.die\#up.Nil + 1.down.B2V + 1.\surd.B1V \\
B2V &\stackrel{def}{=} 1.up.B1V + 1.down.B3V + 1.\surd.B2V \\
B3V &\stackrel{def}{=} 1.up.B2V + 1.down.B4V + 1.\surd.B3V \\
B4V &\stackrel{def}{=} 1.up.B3V + 1.down.B5V + 1.\surd.B4V \\
B5V &\stackrel{def}{=} 1.up.B4V + 1.down.B6V + 1.\surd.B5V \\
B6V &\stackrel{def}{=} 1.up.B5V + 1.down.B7V + 1.\surd.B6V \\
B7V &\stackrel{def}{=} 1.up.B6V + 1.down.B8V + 1.\surd.B7V \\
B8V &\stackrel{def}{=} 1.up.B7V + 1.down.B9V + 1.\surd.B8V \\
B9V &\stackrel{def}{=} 1.up.B8V + 1.down.B10V + 1.\surd.B9V \\
B10V &\stackrel{def}{=} 1.up.B9V + 1.die\#down.Nil + 1.\surd.B10V
\end{aligned}$$

We now provide a process that moves the beatle at random unbaisedly amongst the four major axes.

$$BC \overset{def}{=} 1.'left.BC + 1.'right.BC + 1.'up.BC + 1.'down.BC$$

Combining this with the processes recording the beatles position, and permitting the *die* action only, we have a beatle that will randomly walk on a 10×10 grid.

$$\begin{aligned} BSYS1 &\overset{def}{=} (BC \times B5H \times B5V) \\ BSYS &\overset{def}{=} BSYS1\lceil\{die\} \end{aligned}$$

We now need to permit the beatle to take ten steps. Since it takes one step per tick, this can be arranged by letting the above process execute for ten ticks and then stopping it. Finally at the point the process stops we should like some action to demonstrate whether the stop was because the beatle lived or died.

$$BM \overset{def}{=} \surd.\surd.\surd.\surd.\surd.\surd.\surd.\surd.\surd.\surd.live.Nil$$

Since our parallel is synchronous the 'free' composition of the above with the random walk system will limit it to ten steps.

$$BG \overset{def}{=} BSYS \times BM$$

This problem was solved after some experience of using the tools, defining the processes took 10 minutes, converting to 394 simultaneous linear equations on a Sparc2 took 3 minutes, using the Maple system took 2 minutes. Total elapsed time to **exact** solution 15 minutes. We found that the beetle will live $\frac{488971}{524288}$ (0.933) of the time.

Since this problem generates a large number of states, relatively straightforwardly, we used it to investigate the robustness of our system. We obtained results for the problem when the beetle takes a greater number of steps before the 'game' terminates.

Steps	P(live)	Number of equations	Time to solve
11	0.91	464	40 seconds
12	0.89	534	54 seconds
13	0.86	604	72 seconds
20	0.67	1094	250 seconds
25	0.55	1444	450 seconds
30	0.45	1794	700 seconds
35	0.37	2144	1192 seconds
40	0.30	2494	1903 seconds
50	0.20	3194	4412 seconds
60	0.13	3894	5542 seconds

3.3 Louis' Destiny

Before we describe the scenario, we discuss an interesting descriptive problem that arises within WSCCS. One of the parts of this problem is that of a user which wishes to acquire, one of two resources that allocate themselves to it probabilistically, e.g., each one is only willing to be taken 90% of the time say. We could approach coding this problem as follows:

$$\begin{aligned} Res &\overset{def}{=} 9.give : GotRes + 1.\surd : Res \\ TRes &\overset{def}{=} Res \times Res \\ Take &\overset{def}{=} 1.\overline{give} : Succ + 1 : \surd : Fail \\ SYS &\overset{def}{=} (Tres \times Take)\lceil(Act - \{give\}) \end{aligned}$$

Where the state *Succ* represents successfully getting at least one of the resources, and the state *Fail* represents what is going to happen if neither of them is successfully acquired. A simple probabilistic analysis demonstrates that the system should reach a state of the form $(Succ \times (GotRes \times Res))\lceil(Act - \{give\})$ 99% of the time. However, if we expand the above process we discover that,

$$\begin{aligned} Sys &= 18.\surd.(Succ \times (GotRes \times Res))\lceil(Act - \{give\}) \\ &\quad +1.\surd.(Fail \times (Res \times Res))\lceil(Act - \{give\}) \end{aligned}$$

or a success rate of only $\frac{18}{19}$. One might at this point claim that there is an error in the process system! **This is not the case.** The process *Take* actually encodes "take one of the resources given that two are **never** allowed to become available". That is, it never can permit the two processes to become available - that choice is forbidden.

To encode this system correctly we need to be far more subtle. What is actually occurring is the two resources independently choose to make themselves available with probability 90%. Then if either of them is available it is taken, if the other made itself available it is ignored. Thus the acquisition of such a resource is a two stage process, firstly detect whether it is willing to be acquired, secondly if needed use it. Hence we arrive at the following process definitions:

$$\begin{aligned} Res &\stackrel{def}{=} 9.resA : (1.getR : GotRes + 1.\surd : Res) + 1.\surd : Res \\ Tres &\stackrel{def}{=} Res \times Res \\ Taker &\stackrel{def}{=} 1.\overline{resA}^2 : getR : Succ + 1.\overline{resA} : getR : Succ + 1.\surd : Fail \\ Sys &\stackrel{def}{=} (Taker \times Tres)\lceil(Act - \{resA, getR\}) \end{aligned}$$

If we expand the above system we discover that:

$$\begin{aligned} Sys &= 81.\surd : \surd : (Succ \times GotRes \times Res)\lceil(Act - \{resA, getR\})+ \\ &\quad 18.\surd : \surd : (Succ \times GotRes \times Res)\lceil(Act - \{resA, getR\})+ \\ &\quad 1.\surd : (Fail \times Res \times Res\lceil(Act - \{resA, getR\}) \end{aligned}$$

and in this case, we do indeed succeed 99 % of the time. The above demonstrates that restrictions have a non-trivial effect in terms of the system probabilities, rather than just purely enforcing communications. Hence it is important to check what the processes actually imply in terms of probabilities.

The problem we wish to solve is given on page 224 of [Kre86] and we restate it is as follows. A leprechaun called Louis works as an icicle miner, he owns two centipedes XXY and YYY, used for transport to and from work, both of which are somewhat unwilling to perform this task. In the evening Louis will eat some number of bags of chips, if he eats 10 or more, his centipedes will respond to his entreaties to transport him 90% of the time. If, however, he has less than 10 bags of chips, his centipedes are responsive only 50% of the time. Centipedes are very sensitive to the eating habits of their owners. Louis has a nymphfriend Gwendolin, who drops by and gives him a lift to or from work with probability 30%. Naturally Louis prefers to travel with Gwendolin, so he will not even try to wake his centipedes in this case. If Louis is stranded at home then he gets turned into a toad by his employer, an irate wizard, and goes off to live in wedded bliss with Gwendolin. If he gets stranded at work, then he goes off to Morven and spends the rest of his life drunkenly carousing. We want to calculate the probability that Louis eventually ends up in a blissful state.

Hence we arrive at the following description of the 'Louis' destiny' problem:

$$\begin{array}{rcl}
CentH & \stackrel{def}{=} & 1.mr10.(9.avCH.(1.\overline{gCH}.CentW + 1.\surd.CentH) + 1.\surd.CentH)+ \\
& & 1.l10.(1.avCH.(1.\overline{gCH}.CentW + 1.\surd.CentH) + 1.\surd.CentH)+ \\
& & 1.\surd.CentH \\
CentW & \stackrel{def}{=} & 1.mr10.(9.avCW.(1.\overline{gCW}.CentH + 1.\surd.CentW) + 1.\surd.CentW)+ \\
& & 1.l10.(1.avCW.(1.\overline{gCW}.CentH + 1.\surd.CentW) + 1.\surd.CentW)+ \\
& & 1.\surd.CentW
\end{array}$$

The above two processes represent a centipede at home and at work respectively. In both cases they are supplied with the number of bag of chips that the user has eaten, and then make themselves available appropriately. If a caterpillar is used as transport then it changes from being at home or at work as appropriate. The next process describes how many bags of chips Lewis has eaten. Since it only matters wether he has had more or less than 10, we simply give probabilities for those events.

$$\begin{array}{rcl}
Louis & \stackrel{def}{=} & 4.\surd.L10HG + 6.\surd.L9HG \\
L10HG & \stackrel{def}{=} & 3.\surd.L10W + 7.\surd.L10H \\
L9HG & \stackrel{def}{=} & 3.\surd.L9W + 7.\surd.L9H
\end{array}$$

If Louis is lucky he gets a lift to work with Gwendolin, but we must remember how many bags of chips he ate in order to present the information for a possible centipede ride home. If Gwendolin doesn't give him a lift he must try to obtain a centipede to transport him. So firstly he 'tells' them how many bags of chips he had, and then if one (or more) of them is willing and at the correct place he uses it as transport to work:

$$\begin{array}{rcl}
L10H & \stackrel{def}{=} & 1.\overline{mr10}.L10HD + \omega.\overline{mr10}^2.L10HD \\
L10HD & \stackrel{def}{=} & 1.\overline{avCH}^2.gCH.L10W + 1.\overline{avCH}.gCH.L10W \\
L9H & \stackrel{def}{=} & 1.\overline{l10}.L9HD + \omega.\overline{l10}^2.L9HD \\
L9HD & \stackrel{def}{=} & 1.\overline{avCH}^2.gCH.L9W + 1.\overline{avCH}.gCH.L9W + 1.\surd.toad.Nil
\end{array}$$

When Louis is at work, he can get a lift home from Gwendolin and then will eat his nightly supply of chips:

$$\begin{array}{rcl}
L10W & \stackrel{def}{=} & 3.\surd.Louis + 7.\surd.L10WC \\
L9W & \stackrel{def}{=} & 3.\surd.Louis + 7.\surd.L9WC
\end{array}$$

Otherwise he has to get one of his centipedes to transport him from work to home:

$$\begin{array}{rcl}
L10WC & \stackrel{def}{=} & 1.\overline{mr10}.LWD + \omega.\overline{mr10}^2.LWD \\
LWD & \stackrel{def}{=} & 1.\overline{avCW}^2.gCW.Louis + 1.\overline{avCW}.gCW.Louis \\
L9WC & \stackrel{def}{=} & 1.\overline{l10}.LWD + \omega.\overline{l10}^2.LWD
\end{array}$$

The complete system is Louis at home having (presumably) eaten, with the two difficult centipedes.

$$LD \stackrel{def}{=} \Theta((Louis \times CentH \times CentH)\lceil\{toad, morv\})$$

An analysis of the above problem, using a combination of process to linear probabilistic equations and MAPLE, demonstrates that Louis will be turned into a toad and therefore marry his beloved with probability exactly $\frac{104211116700}{28953925571}$ of the time (as a decimal 0.36), and the average number of ticks he works before he resolves his future is $\frac{3128310267610}{202677478997}$ or 15.44 ticks. Unfortunately as we have constructed the above we cannot interpret the ticks as days since whenever he gets a lift from Gwendolin he skips three ticks of interaction with his centipedes.

So we retime the relevant processes and define them as follows:

$$\begin{aligned}
L10HG &\overset{def}{=} 3.\surd.\surd.\surd.\surd.L10W + 7.\surd.L10H \\
L9HG &\overset{def}{=} 3.\surd.\surd.\surd.\surd.L9W + 7.\surd.L9H \\
L10W &\overset{def}{=} 3.\surd.\surd.\surd.\surd.Louis + 7.\surd.L10WC \\
L9W &\overset{def}{=} 3.\surd.\surd.\surd.\surd.Louis + 7.\surd.L9WC
\end{aligned}$$

Constructing the LD system as before but using these new definitions for the 4 relevant processes. We discover (thankfully) that the probability of Louis living in wedded bliss (sic) is exactly the same as in the unbalanced version of the process. The average duration of the process is as a decimal 19.255 ticks. The number of ticks required to complete a days activity is 9, so we see that it takes on average just over 2 days, before Louis fate is established. An interesting observation is that the unbalanced version has 14 more states, and thus will be (marginally) slower to verify.

4 Approaching Inifinity

We present an example where the system can have an infinite number of states, which we shall truncate to a finite number. Within our approach we shall be able to observe the precise amount, by weight, of the system state space that we have **actually** explored. If we wish to know how much we can rely on the data produced by such approximations, then a knowledge of the precise degree of that approximation is evidently of great importance.

4.1 Snorks and Snarks

We restate the problem from page 266 of [Kre86] (adapted from [DLSB82]). An arena has seven doors, behind each of which is either a ferocious snark or a beautiful snork. A succession of gladiators have to pick a door. If the gladiator picks a snark then he is immediately killed, and the snark is replaced behind the door. If he is lucky and picks a snork, then he is rewarded and freed, one of the snarks is killed and the snork is replaced. The question is to determine how many gladiators will be killed on average, and how many will live.

$$Glad \overset{def}{=} 1.die\#snark.Glad + 1.free\#snork.Glad$$

The above process describes the gladiators repeatedly trying the doors, the probabilities of the respective outcomes are given by the following process:

$$\begin{aligned}
Ar4 &\overset{def}{=} 4.\overline{snark}.Ar4 + 3.\overline{snork}.Ar3 \\
Ar3 &\overset{def}{=} 3.\overline{snark}.Ar3 + 3.\overline{snork}.Ar2 \\
Ar2 &\overset{def}{=} 2.\overline{snark}.Ar2 + 3.\overline{snork}.Ar1 \\
Ar1 &\overset{def}{=} 1.\overline{snark}.Ar1 + 3.\overline{snork}.Nil
\end{aligned}$$

In the above arena process the probability of picking a snork or a snark is given in terms of the number of snarks remaining. Each time one is killed we move to the state with the appropriate smaller probability of picking a snark. This state change is caused by the gladiator picking a snork, that is interacting on the *snork* action. So the tournament is given by letting the gladiators interact with the arena, and only recording whether the gladiator was freed or died:

$$Tor \stackrel{def}{=} (Glad \times Ar4)\lceil\{die, free\}$$

In order to see how many gladiators lived or died we add some couter processes.

$$Tor1 \stackrel{def}{=} (Tor \times Fr0 \times Die0)\lceil\{tmf, tmd\}$$

The following process records the number of gladiators that have been freed:

$$\begin{aligned}
Fr0 &\stackrel{def}{=} 1.\overline{free}.Fr1 + 1.t.Fr0\\
Fr1 &\stackrel{def}{=} 1.\overline{free}.Fr2 + 1.t.Fr1\\
Fr2 &\stackrel{def}{=} 1.\overline{free}.Fr3 + 1.t.Fr2\\
Fr3 &\stackrel{def}{=} 1.\overline{free}\#tmf.Nil + 1.t.Fr3
\end{aligned}$$

When the maximum (4) gladiators have been freed this process records the fact by performing a *tmf* action. This process records the number that have died.

$$\begin{aligned}
Die0 &\stackrel{def}{=} 1.\overline{die}.Die1 + 1.t.Die0\\
Die1 &\stackrel{def}{=} 1.\overline{die}.Die2 + 1.t.Die1\\
Die2 &\stackrel{def}{=} 1.\overline{die}.Die3 + 1.t.Die2\\
Die3 &\stackrel{def}{=} 1.\overline{die}.Die4 + 1.t.Die3\\
Die4 &\stackrel{def}{=} 1.\overline{die}.Die5 + 1.t.Die4\\
Die5 &\stackrel{def}{=} 1.\overline{die}.Die6 + 1.t.Die5\\
Die6 &\stackrel{def}{=} 1.\overline{die}.Die7 + 1.t.Die6\\
Die7 &\stackrel{def}{=} 1.\overline{die}.Die8 + 1.t.Die7\\
Die8 &\stackrel{def}{=} 1.\overline{die}.Die9 + 1.t.Die8\\
Die9 &\stackrel{def}{=} 1.\overline{die}\#tmd.Nil + 1.t.Die9
\end{aligned}$$

When more gladiators have been killed than the system has capacity for, the fact is recorded by the performance of a *tmd* action.

Solution for 4 going free, with at most 10 dying is 0.97. This problem is interesting in that we know (simple analysis) that 4 gladiators **must** eventually escape. So if we could allow an infinite number to die, then we would be guaranteed to see 4 gladiators escaping. Thus by taking a restricted number of possible deaths and noting how likely four gladiators are to escape we can detect how much of the potential state space, by weight, we have examined. In this case we have only neglected about 3% of all possible cases. Clearly by increasing the number of possible deaths we can get as close to the whole state space as we need. At this point it only needed 53 equations to describe the system, so we are well within any mechanical limitations.

This approach to detecting the onset of an infinite state system, will be possible in all such systems. We simply add a terminating action that indicates the system needing more states, and can always (up to mechanical limitations) evaluate the probability of observing that outcome. This **immediately** tells us how much of the probabilistic state space has been neglected.

Many of the simulation systems of greatest interest are essentially infinite state. Even a large number of executions of these systems over an extended period of time will only explore a limited amount of the available state space. Unfortunately, exploring the system in this fashion will leave us unaware of the extent of the state exploration. A more controlled approach to exploring the states of the system will leave the modeller in a better position to understand the validity of their results. By using a process algebraic methodology and exploiting actions that mark the end of the state exploration we can control and observe the limits of our approximation.

The other advantage of this method is that it enforces limits on our need for computational resources, we do not repeat state explorations unnecessarily. Whilst with an algebraic representation, an implementation is likely to be slower in evaluating the behaviour at each state, it will not repeatedly examine the same state. Thus in a system with low probability of reaching extremal states the algebraic approach is liable to **reduce** computational resource needs, simply as a result of removing redundant state exploration.

We shall now evaluate the average number of gladiators killed, firstly we tabulate the probabilities of at least n gladiators getting killed for the first 13 values. From these we can immediately calculate the probability of n getting killed, and hence the mean number of gladiators killed.

n	P(at least n)	$P(n)$
1	0.9036	0.1660
2	0.7376	0.1815
3	0.5561	0.1609
4	0.3952	0.1263
5	0.2689	0.0921
6	0.1767	0.0634
7	0.1133	0.0422
8	0.0711	0.0271
9	0.0440	0.0172
10	0.0268	0.0106
11	0.0162	0.0066
12	0.0096	0.004
13	0.0057	0.0057

So the average number of gladiators that die is approximately 2.8. Alternatively if we only permitted a maximum of 13 gladiators to die, then on average we would see only 2.8 gladiators die. An interesting point about the above is that in [Kre86] it is suggested that the above scenario be simulated on 100 occasions to obtain an estimate of this value. In that period it is highly unlikely that 12 or 13 deaths amongst gladiators would be seen. So we could expect our value to be at least as reliable as one generated by 100 simulations.

5 Examples from Queueing Theory

In order that we may understand the outcomes of queue theoretic problems we need a slightly different approach. Whilst these problems can often be addressed by finite state processes, they will not in general terminate, and thus our earlier transformation technique will not work

We shall assume that our processes are producing actions that witness the state of the system on all transitions, such as the number of people in a queue. We have automated two judgements on such systems. Firstly, the persistent state probability vector for the system assuming it is aperiodic. Secondly, the state average of the witness action. That is, the expected value of the action over the entire execution. As before these solutions are achieved by the production of appropriate input for the MAPLE system.

As an illustrative example consider the following artificial system:

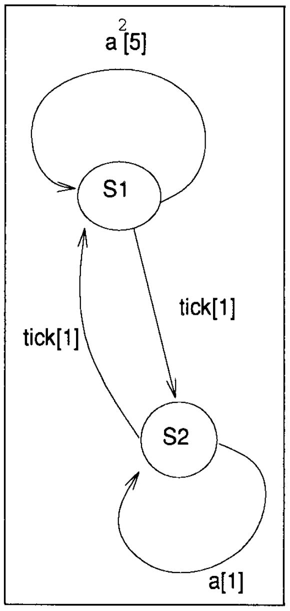


Figure 3: A simple permanent process.

$$S1 = 5.a^2.S1 + 1.\surd.S2$$
$$S2 = 1.a.S2 + 1.\surd.S1$$

in state $S1$ if the process moves to $S2$ ($\frac{1}{6}$ of the time) then it outputs $\surd$, otherwise it outputs a^2 and returns to $S1$ ($\frac{5}{6}$ of the time). In state $S2$ it can perform a $\surd$ ($\frac{1}{2}$ of the transitions) and move to state $S1$, otherwise it performs the action a and stays in state $S2$. We wish to know what is the average number of a actions performed over the execution of the above system. A transition graph for the above system is presented in Figure 3.

We can regard the process $S1$ as being described by a Markov chain whose transition matrix is

$$\begin{pmatrix} \frac{5}{6} & \frac{1}{2} \\ \frac{1}{6} & \frac{1}{2} \end{pmatrix}$$

A straightforward calculation shows that the persistent probability vector for this system is $(\frac{3}{4}, \frac{1}{4})$. Since from state $S1$ we would see a occur with expected amount $\frac{10}{6}$ and from $S2$ $\frac{1}{2}$ we would expect the amount of the action a occuring to be $\frac{3}{4}\frac{10}{6} + \frac{1}{4}\frac{1}{2}$ or $\frac{11}{8}$. Thus if we observed this process over a long time we would expect the average number of a actions being perfromed at any instant to be $\frac{11}{8}$, obviously we would have to interpret this in terms of our intended meaning for a actions.

5.1 A Simple Bounded Queue

Let us compute the average number of persons queuing in a bounded queue, having at most 5 people in it. We choose an arrival rate of 1 per 5 ticks and a service rate of 2 per 5 ticks. The problem can be encoded as the following set of processes:

$$\begin{aligned}
Q0 &\stackrel{def}{=} 1.q\#start.Q1 + 1.\surd.Q0 \\
Q1 &\stackrel{def}{=} 1.q\#serve\#start.Q1 + 1.q^2\#start.Q2 + 1.q.Q1 + 1.serve.Q0 \\
Q2 &\stackrel{def}{=} 1.q^2\#serve\#start.Q2 + 1.q^3\#start.Q3 + 1.q^2.Q2 + 1.q\#serve.Q1 \\
Q3 &\stackrel{def}{=} 1.q^3\#serve\#start.Q3 + 1.q^4\#start.Q4 + 1.q^3.Q3 + 1.q^2\#serve.Q2 \\
Q4 &\stackrel{def}{=} 1.q^4\#serve\#start.Q4 + 1.q^5\#start.Q5 + 1.q^4.Q4 + 1.q^3\#serve.Q3 \\
Q5 &\stackrel{def}{=} 1.q^5\#serve\#start.Q5 + 1.q^5.Q5 + 1.q^4\#serve.Q4
\end{aligned}$$

The above process records how many people are in the queue. New elements enter the queue by performing a *start* action, and objects leave the queue when they are served, the performance of a *serve* action. This process has no system probabilities it just records the state, in terms of the length of the queue, and reports it via the q action. The process Enq produces objects to enter the queue at the desired rate:

$$Enq \stackrel{def}{=} 1.'start.Enq + 4.\surd.Enq$$

A server process will serve an item in the queue if there is one in it in preference to doing nothing. After it has started to serve someone it then takes a geometric time to complete the service.

$$\begin{aligned}
SF &\stackrel{def}{=} \omega.'serve.SB + 1.\surd.SF \\
SB &\stackrel{def}{=} 2.\surd.SF + 3.t.SB
\end{aligned}$$

The order in which these process are composed is important, firstly we from the counter and the server, using the priority information to ensure that the service takes place as soon as the srever if free:

$$Qu5 \stackrel{def}{=} \Theta((Q0 \times SF)\lceil\{q, start\})$$

The we form the complete system by adding the queue member producing process:

$$QSYS \stackrel{def}{=} (Enq \times Qu5)\lceil\{q\}$$

This qives a mean number of entities in the queue of 0.902 per tick. Queueing theory on an unbounded queue suggests this value should be 1. We tried with a limit of 10 customers and discovered the average queue length to be, 1.006. Although we would expect our average to be greater than that for a continuos time queue as the server in this system takes at least one unit of time to do anything.

5.2 Single Server Multiple User

We can study the waiting times of multiple users of a single server. A flow diagram for the system we examine is presented in Figure 4. The properties of such systems are studied in [Adi72, Kle75]. Each user waits for some input from the server, it then performs some work generating new input for the user. We can describe this system by the following collection of WSCCS processes:

$$\begin{aligned}
SYS &\stackrel{def}{=} 1.sa.SA + 1.sb.SB + 1.sc.SC + 1.\surd.SYS \\
SA &\stackrel{def}{=} 2.\surd.SA + 1.'da.SYS \\
SB &\stackrel{def}{=} 2.\surd.SB + 1.'db.SYS \\
SC &\stackrel{def}{=} 2.\surd.SC + 1.'dc.SYS
\end{aligned}$$

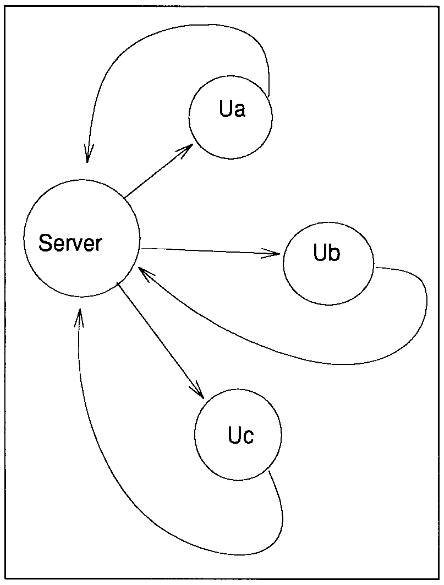


Figure 4: Single Server Multiple User System.

The serever takes in work from any of the users, it then processes the work and records completion by the performance of the appropriate *da*, *db* or *dc* action.

$$\begin{aligned} UA &\stackrel{def}{=} 1.da.UAB + 1.\surd.UA \\ UAB &\stackrel{def}{=} 1.\surd.UAW + 2.\surd.UAB \\ UAW &\stackrel{def}{=} \omega.'sa.UA + 1.wait.UAW \end{aligned}$$

User A waits until its task is finished by the server, it then takes a geometrically distributed time to generate some new work for the system to perform, and tries to get the system to start work on it as soon as possible. The users B and C are identical to the user A up to renaming of the actions witnessing the movement of work.

$$\begin{aligned} UB &\stackrel{def}{=} 1.db.UBB + 1.\surd.UB \\ UBB &\stackrel{def}{=} 1.\surd.UBW + 2.\surd.UBB \\ UBW &\stackrel{def}{=} \omega.'sb.UB + 1.wait.UBW \\ UC &\stackrel{def}{=} 1.dc.UCB + 1.\surd.UC \\ UCB &\stackrel{def}{=} 1.\surd.UCW + 2.\surd.UCB \\ UCW &\stackrel{def}{=} \omega.'sc.UC + 1.wait.UCW \end{aligned}$$

The *wait* action in user C records when it has work for the server to do, but the server has not yet taken it. Since the system is symmetric in A, B and C, we only need to record the delay on one of them. Finally we build the system, by taking the prioritiesed part of the server and user processes in parallel, allowing the *wait* action for recording purposes.

$$SS \stackrel{def}{=} \Theta((SYS \times UAB \times UBB \times UCB)\lceil\{wait\})$$

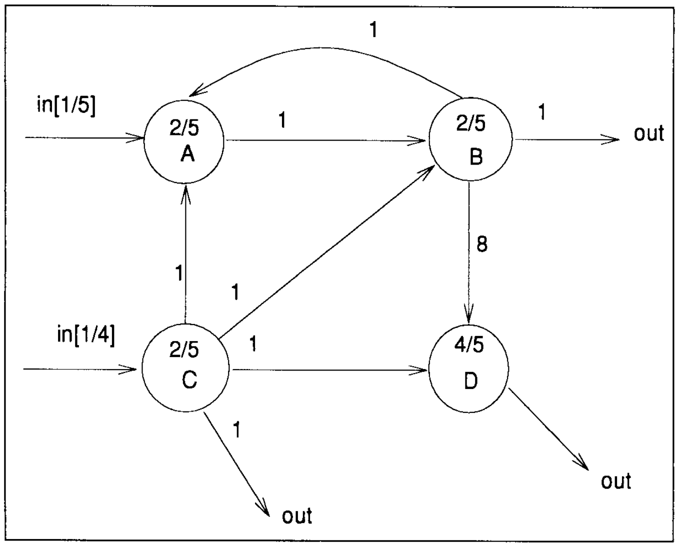


Figure 5: A Queuing Network Example.

The simplest manner in which we can determine system usage is to replace the first line of the server description with the following:

$$SYS \overset{def}{=} 1.sa.SA + 1.sb.SB + 1.sc.SC + 1.free.SYS$$

Rename the *wait* action to $\surd$, and form the complete system as before, but permitting the *free* action to occur. The usage of the system is one minus the amount of time it is free.

With a server having probability of $\frac{1}{2}$ of dealing with a request from any user per tick, the mean amount of waiting is 0.22, and the usage is 93%. If we set the average user work time to 12 ticks and the average service time to 2.01 ticks, then the average amount of time a user spends waiting is 3%, and the server is active for approximately 80% of the time.

5.3 Queueing Networks

We shall describe the queueing network of Figure 5 as a collection of WSCCS processes and analyse some of its behaviours.

We encode the above process with limits of 5 items queuing per buffer, similar to the specialisation in [GN67] of the work of [Jac57], a demonstration of how such systems can be made into Markov systems is given in [Kin69]. We assume that internal messages take priority over external queueing requests as define the system as follows:

$$\begin{aligned} Enqa &\overset{def}{=} 1.'sa.Enqa + 4.\surd.Enqa \\ Enqc &\overset{def}{=} 1.'sc.Enqc + 3.\surd.Enqc \end{aligned}$$

The above two proesses enque work at nodes A and C respectively. The next process records the number of items queueing at node A.

$$\begin{aligned}
QA0 &\stackrel{def}{=} 1.q\#sa.QA1 + 1.\surd.QA0 \\
QA1 &\stackrel{def}{=} 1.q\#sa\#serve.QA1 + 1.q^2\#sa.QA2 + 1.q.QA1 + 1.serve.QA0 \\
QA2 &\stackrel{def}{=} 1.q\#sa\#serve.QA2 + 1.q^3\#sa.QA3 + 1.q^2.QA2 + 1.q\#serve.QA1 \\
QA3 &\stackrel{def}{=} 1.q\#sa\#serve.QA3 + 1.q^4\#sa.QA4 + 1.q^3.QA3 + 1.q^2\#serve.QA2 \\
QA4 &\stackrel{def}{=} 1.q\#sa\#serve.QA4 + 1.q^5\#sa.QA5 + 1.q^4.QA4 + 1.q^3\#serve.QA3 \\
QA5 &\stackrel{def}{=} 1.q\#sa\#serve.QA5 + 1.q^5.QA5 + 1.q^4\#serve.QA4
\end{aligned}$$

This process serves the queue that is formed at node A

$$\begin{aligned}
SAF &\stackrel{def}{=} \omega.'serve.SAB + 1.\surd.SAF \\
SAB &\stackrel{def}{=} 2.da.SAF + 3.\surd.SAB
\end{aligned}$$

we form the queue at A as we formed the single queue:

$$QA \stackrel{def}{=} \Theta((QA0 \times SAF)\lceil\{da, q, sa\})$$

We need a process to redirect the output of A to the appropriate nodes, with the appropriate probabilities:

$$\begin{aligned}
RDA &\stackrel{def}{=} 1.'da.RDAB + 1.\surd.RDA \\
RDAB &\stackrel{def}{=} \omega.'sb.RDA + 1.\surd.RDA
\end{aligned}$$

We assume an eager model, where if buffers are full then internal queuing events are thrown away. Finally we can form the queue at node A leaving only actions that record the size of the queue, allow items to be queued at A, and record items being removed from A to be queued at other nodes.

$$QAS \stackrel{def}{=} (QA \times RDA)\lceil\{q, sa, 'sb\}$$

The following sequence of definitions form a similar system for the node B.

$$\begin{aligned}
QB &\stackrel{def}{=} QA[db/da, sb/sa] \\
RDB &\stackrel{def}{=} 8.'db.RDBD + 1.'db.RDB + 1.'db.RDBA + 1.\surd.RDB \\
RDBD &\stackrel{def}{=} \omega.'sd.RDB + 1.\surd.RDB \\
RDBA &\stackrel{def}{=} \omega.'sa.RDB + 1.\surd.RDB \\
QBS &\stackrel{def}{=} (QB \times RDB)\lceil\{q, 'sa, sb, 'sd\}
\end{aligned}$$

These form the queue at node C.

$$\begin{aligned}
QC &\stackrel{def}{=} QA[sc/sa, dc/da] \\
RDC &\stackrel{def}{=} 1.'dc.RDCA + 1.'dc.RDC + 1.'dc.RDCB + 1.'dc.RDCD + 1.\surd.RDC \\
RDCA &\stackrel{def}{=} \omega.'sa.RDC + 1.\surd.RDC \\
RDCB &\stackrel{def}{=} \omega.'sb.RDC + 1.\surd.RDC \\
RDCD &\stackrel{def}{=} \omega.'sd.RDC + 1.\surd.RDC \\
QCS &\stackrel{def}{=} (QC \times RDC)\lceil\{q, 'sa, 'sb', sc, 'sd\}
\end{aligned}$$

And these form the queue at D.

$$
\begin{aligned}
QD0 &\stackrel{def}{=} QA0[sd/sa] \\
SD &\stackrel{def}{=} \omega.'serve.SDB + 1.\surd.SD \\
SDB &\stackrel{def}{=} 4.dd.SD + 1.\surd.SDB \\
QD &\stackrel{def}{=} \Theta((QD0 \times SD)\lceil\{dd, q, sd\}) \\
RDD &\stackrel{def}{=} 1.'dd.RDD + 1.\surd.RDD \\
QDS &\stackrel{def}{=} (QD \times RDD)\lceil\{q, sd\}
\end{aligned}
$$

Finally we form the network but placing the nodes in communication with each other, and the processes which enqueue work from the outside of the network. We permit the action q which records the total number of items in queues at all nodes in the networks. It is relatively straightforward to adjust the processes to record local queues, or usage levels.

$$QN \stackrel{def}{=} \Theta((QAS \times Enqa \times QBS \times QCS \times Enqc \times QDS)\lceil\{q\})$$

Unfortunately, whilst we can encode this problem in a relatively small amount of text, it has approximately 15,000 states which is well beyond the capacity of the current analysis methods. We include it as an example of a large system which can be encoded in a relatively brief form yet is (very) difficult to examine. Hopefully, developments of these approaches using more subtle calculations will permit reasoning over such large systems.

6 Conclusions and Further Work

The approach of describing Monte Carlo simulation problems within a process algebra seems to be highly promising. The descriptions we arrive at are highly compact, and it is possible to derive some of their properties automatically. The current limitations are that process algebras do not have the kind of automatic data gathering facilities that are possible within conventional problem description languages; although hopefully the provision of suitable tools will overcome some of this limitation.

In the Introduction we stated that it should be possible to solve these problems in general, without instantiating any probabilities in the actual systems. Unfortunately, at the moment the only support tool for WSCCS does not have a facility for the algebraic manipulation of weights. Since, this manipulation is fundamentally simple we hope to see such a system developed in the near future. In principle it is no harder to solve these problems in general, using these methods, than it is to solve them in specific cases.

We believe that our approach stays close to the simulation ideal of allowing the behavior of 'concrete' systems to be observed directly. That is the real world problem and its simulation description are very closely related. The process algebraic approach takes this idea one stage further, we permit the description of the system 'as seen', but then permit the formal derivation of the properties of the system.

7 Bibliography.

[Adi72] I. Adiri, Queueing Models for Multiprogramming Computers, Proc. of International Symp. on Computer Communication Networks and Teletraffic, Polytech Press, N.Y. pp441-448, 1972.

[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.

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

[Cam89] J. Cammilleri. 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.

[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.

[GN67] W.J. Gordon and G.F. Newell, Closed Queueing Systems with Exponential Servers, Operations research 15:245-265, 1967.

[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.

[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] M. Hatcher, C. Tofts and N. Franks, Mutual Exclusion as a Mechanism for Information Exchange in Ant Nests, To appear.

[Jac57] J.R. Jackson, Networks of Waiting Lines, OPerations Research 5:518-521, 1957.

[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.

[Kle] 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.

[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.

[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.

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

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

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

[Tof93a] C. Tofts, Using Process Algebra to Describe Social Insect Behaviour, Transactions on Simulation, 1993.

[Tof93b] C. Tofts, Processes with Probabilities, Priorities and Time, Submitted Formal Aspects of Computer Science.

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

A The Language WSCCS.

Our language WSCCS is an extension of Milner's SCCS (Milner, 1983) 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. We also take a set of weights $\mathcal{W}$, denoted by w_i, which are of the form[1] $n\omega^k$ 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 $k \geq k'$), we consider the objects n used as weights to be abbreviations for $n\omega^0$:

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

[1] Here n is the relative frequency with which this choice should be taken and k is the priority level of this choice.

and 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 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;
- $\sum\{w_iE_i|i \in I\}$ 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 $1\omega^k$.

A.1 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 (Plotkin, 1981), defining the actions that a process can perform and the weight with which a state can be reached. In Figure 6 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 $\longrightarrow \subseteq WSCCS \times Act \times WSCCS$ and the least multi-relation $\longmapsto \subseteq$

$$\frac{}{a.E \xrightarrow{a} E} \qquad \frac{}{\sum\{w_i.E_i | i \in I\} \xmapsto{w_i} E_i}$$

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

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

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

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

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

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

$$\frac{E \xrightarrow{a} E'}{\Theta(E) \xrightarrow{a} \Theta(E')} \qquad \frac{E \xmapsto{n\omega^k} E' \nexists (k' > k).E \xmapsto{m\omega^{k'}}}{\Theta(E) \xmapsto{n} \Theta(E')}$$

Figure 6: Operational Rules for WSCCS.

$bag(WSCCS \times \mathcal{W} \times WSCCS)$ [2], which are written $E \xrightarrow{a} F$ and $E \xmapsto{w} F$ respectively, satisfying the rules laid out in Figure 6. These rules respect the informal description of the operators given earlier. The reason that processes are multi-related by weight is that we may specify more than one way to choose the same process with the same weight and we have to retain all the copies. For example, the process

$$1.P + 1.P + 1.Q$$

can evolve to the process P with cummulative weight 2, so that we have to retain both evolutions.

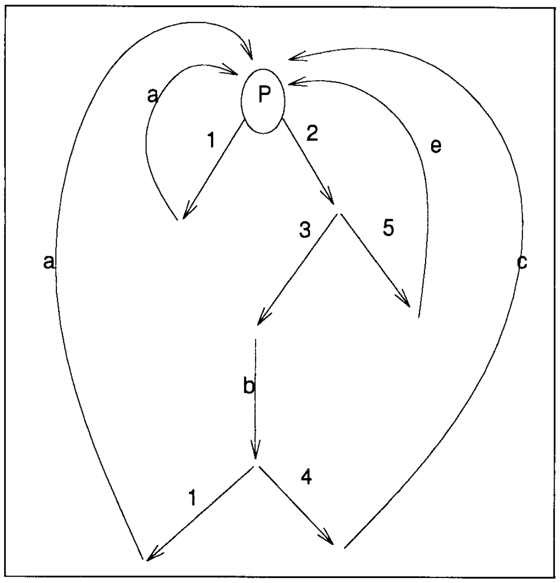
The predicate $does_A(E)$ is well defined since we have only permitted finitely branching choice expressions. The action of the permission operator is to prune from the choice tree those processes that can no longer perform any action.

A.1.1 The Structure of Processes.

From the semantics of WSCCS processes we can see that any process is a digraph, that is a collection of weight transitions that eventually lead to a single action transition. So there is no explicit choice

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

Figure 7: Transition graph for PEX



betweens actions, the only form of non-determinism is between weights. Given the process:

$$PEX \overset{def}{=} 1.a.PEX + 2.(3.b.(1.a.PEX + 4.c.PEX) + 5.e.PEX)$$

we can think of it as describing the transition graph given in Figure 7.

A.1.2 Direct Bisimulation.

Our bisimulations will be based on the accumulation technique of Larsen and Skou (1989). We start by defining accumulations of evolutions for both types of transition.

Definition A.1 *Let S be a set of processes then:*

- $P \overset{w}{\longmapsto} S$ *with* $w \overset{def}{=} \sum\{w_i | P \overset{w_i}{\longmapsto} Q$ *for some* $Q \in S\}$*;*[3]
- $P \overset{a}{\longrightarrow} S$ *iff there exists* $Q \in S$ *and* $P \overset{a}{\longrightarrow} Q$*.*

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 indentification we wish to make, but we will make such an identification later.

Definition A.2 *An equivalence relation* $R \subseteq Pr \times Pr$[4] *is a* direct bisimulation *if* $(P, Q) \in R$ *implies for all* $S \in Pr/R$ *that:*

[3]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.

[4]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 refering, we will omit the subscript.

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

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

Definition A.3

$$\overset{d}{\sim} \equiv \bigcup\{R \mid R \textit{ is a direct bisimulation }\}.$$

That $\overset{d}{\sim}$ is an equivalence follows immediately from it being a union of equivalences.

Lemma A.4 *Let P and Q be processes such that $P \overset{d}{\sim} Q$. Then for all action sets A, $does_A(P)$ iff $does_A(Q)$.*

Proposition A.5 *Direct equivalence is substitutive for finite processes. Thus, given $P \overset{d}{\sim} Q$ and $P_i \overset{d}{\sim} Q_i$ for all $i \in I$ then:*

1. $a.P \overset{d}{\sim} a.Q$; *2. $\Sigma_{i\in I} w_i P_i \overset{d}{\sim} \Sigma_{i\in I} w_i Q_i$;*
3. $P \times E \overset{d}{\sim} Q \times E$; *4. $P\lceil A \overset{d}{\sim} Q\lceil A$;*
5. $P[S] \overset{d}{\sim} Q[S]$.

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

Definition A.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 A.7 *If $\tilde{E} \overset{d}{\sim} \tilde{F}$ then $\mu_i \tilde{X}.\tilde{E} \overset{d}{\sim} \mu_i \tilde{X}.\tilde{F}$.*

A.1.3 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,

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

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 A.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 normalized 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 (Smolka *et al.*, 1989; van Glabbek *et al.*, 1990).

Definition A.9

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

Proposition A.10 *Let P and Q be processes such that $P \overset{d}{\sim} Q$, then $P \overset{r}{\sim} Q$.*

Definition A.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 A.12 *$\overset{r}{\sim}$ is a congruence for finite and finite state processes.*

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 (Smolka *et al.*, 1989), but is a useful notion of equivalence.

Definition A.13 *We define an abstract notion of evolution as follows;*

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

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

Definition A.14 *Let S be a set of processes then:*

$$P \xrightarrow{a[w]} S \textit{ iff } w = \sum\{w_i | P \xrightarrow{a[w_i]} Q \textit{ for some } Q \in S\};$$ [5]

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

Definition A.15 *We say an equivalence relation $R \subseteq Pr \times Pr$ is an* abstract bisimulation *if $(P, Q) \in R$ implies that:*

there are $c_1, c_2 \in \mathcal{P}$ such that for all $S \in Pr/R$ and for all $w, v \in \mathcal{W}$, $P \xrightarrow{a[w]} S$ iff $Q \xrightarrow{a[v]} S$ and $c_1 w = c_2 v$.

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

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

$$(\mathit{Exp}_1)\ \ a.E \times b.F = ab.(E \times F) \qquad (\mathit{Exp}_2)\ \ a.E \times \Sigma_{j\in J} v_j F_j = \Sigma_{j\in J} v_j (a.E \times F_j)$$

$$(\mathit{Exp}_3)\ \ (\Sigma_{i\in I} w_i E_i) \times (\Sigma_{j\in J} v_j F_j) = \Sigma_{(i,j)\in(I\times J)} v_i w_j (E_i \times F_j)$$

$$(\mathit{Res}_1)\ \ (a.E)\lceil A = \begin{cases} a.(E\lceil A) \text{ if } a \in A\\ \mathbf{0} \text{ otherwise.}\end{cases}$$

$$(\mathit{Res}_2)\ \ (\Sigma_{i\in I} w_i E_i)\lceil A = \Sigma_{j\in J} w_j (E_j\lceil A) \text{ where } J = \{i \in I | d_A(E_i)\}$$

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

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

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

Figure 8: Equational rules for WSCCS.

A.2 Equational Characterisation of WSCCS.

We present some equational laws over WSCCS processes in Figure 8, these form a sound and complete equational system over the finite processes in WSCCS. We shall write $p = q$ for $p \overset{r}{\sim} q$.

Definition A.16 *Let A be an action set then the predicate, $d_A(E)$, expressing the fact that E can perform an action in A, is defined recursively as follows:*

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

Definition A.17 *Let W be a set of weights $\{w_i\}$ then $max_\omega(W)$ is the maximum power of ω occuring in W.*

Definition A.18 *We define a projection from on weights as follows,*

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

The major difference when we extend our weight set to have many infinities is that the priority operator will now distribute over multiplication. The following equation now holds:

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

this permits much greater freedom in the use of priority and ensures that it more closely matches with our intuitions.

[5]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.