Exploiting strong attractors to slaughter monsters - taming 10^{1500} states and beyond

Chris Tofts
Trusted Systems Laboratory
HP Laboratories Bristol
HPL-2006-121
September 5, 2006*

process algebra, approximation, graph building, model checking, BDD, probability

The `holy grail' of automated state based model checking is to build precisely the states needed to validate a property and no more. In this paper we present a natural filter on automata which exploits the nature of the design of concurrent systems to order the state construction. We demonstrate that this can provide a sequence of approximating models which permit us to both address infinite systems and large finite systems (a 500 component 10^{1500} state system being effectively modelled). The models are optimal, in the number of states used, for the parameters (state occupancy probabilities and consequent rewards) they attempt to approximate. The filter on the states we deduce is an interesting variant on the near decomposable class of systems presented by Simon and Ando (1961), but one where the small value terms **do not** necessarily dominate the long run behaviour. The semantic nature of the decomposition avoids the need to instantiate any values, or introduce arbitrary numerical bounds during the automata construction, enabling us to derive bounds on permitted behaviour without an expensive rebuild of the automata. Whilst the main focus of the work is the evaluation of properties of probabilistic systems, probabilities are not required for the construction, consequently the approach can be exploited for qualitative arguments of system correctness. Seven examples of layering of various systems are presented. One of the most powerful properties of the approach is that it is reasonable to argue that either the approximation technique works, or the system is inherently badly designed.

* Internal Accession Date Only
Approved for External Publication

Exploiting strong attractors to slaughter monsters - taming 10^{1500} states and beyond.

C. Tofts
HP Research Laboratories Bristol,
Filton Road, Stoke Gifford,
Bristol, BS34 8QZ,
chris_tofts@hp.com

August 22, 2006

Abstract

The 'holy grail' of automated state based model checking is to build precisely the states needed to validate a property and no more. In this paper we present a natural filter on automata which exploits the nature of the design of concurrent systems to order the state construction. We demonstrate that this can provide a sequence of approximating models which permit us to both address infinite systems and large finite systems (a 500 component 10^{1500} state system being effectively modelled). The models are optimal, in the number of states used, for the parameters (state occupancy probabilities and consequent rewards) they attempt to approximate. The filter on the states we deduce is an interesting variant on the near decomposable class of systems presented by Simon and Ando (1961), but one where the small value terms **do not** necessarily dominate the long run behaviour. The semantic nature of the decomposition avoids the need to instantiate any values, or introduce arbitrary numerical bounds during the automata construction, enabling us to derive bounds on permitted behaviour without an expensive rebuild of the automata. Whilst the main focus of the work is the evaluation of properties of probabilistic systems, probabilities are not required for the construction, consequently the approach can be exploited for qualitative arguments of system correctness. Seven examples of layering of various systems are presented. One of the most powerful properties of the approach is that it is reasonable to argue that either the approximation technique works, or the system is inherently badly designed.

1 Introduction

One of the major problems of understanding concurrent systems is that they have lots of potential states. In fact the rate of growth in the raw number of states in the system is at least exponential in the number of components in the system. In some cases even a relatively small system, containing spawning elements can give rise to an infinite number of states. In this context the ability to 'pick' the most important states from the system so as to only focus effort on building an analysing these states is critical. Some 'on the fly' approaches to model checking do limit the requirement for state building to only those necessary to establish the validity of the formula, but even these approaches do not take advantage of the relative importance of states.

Concurrent (distributed) systems[1] can be largely regarded in functioning in the following manner. There is a core behaviour which obtains when everything functions correctly: for instance all

[1]There are concurrent systems that do not fit this view such as switches, multipliers and other low level hardware, but the distinction of distributed vs concurrent is not a well agreed one.

messages are correctly transmitted in order and on time, all sub processes function correctly and on time, none of the components of a system fail... The behaviour within this setting can be relatively straightforward. The full system consist of this behaviour combined with a collection of activities which ameliorate the effects of the 'problems' when they occur. It is reasonable to believe that the core describes an 'optimal' good behaviour, and is consequently the best that can be expected from the system. Equally importantly this behaviour, or at least one state one it, should be known. As this is the **designed** system behaviour it is reasonable to expect that the designer is aware of it. Interestingly random coupled systems usually exhibit very small 'core' behaviour [6], so we should hope that in general the core will be small.

With this view of a concurrent system we can see the state space as being constructed in the following manner, the is a core, ground state, behaviour, and a succession of increasingly complex behaviours that are a consequence of the system being 'excited' by the occurrence of 'bad' events: more than 1 object desiring to use a single resource, message loss, message corruption, unsuccessful termination, untimely termination, overrun in time... The perceived behaviour of the system will be given by how much time it spends in the core. In this context the critical parameters to evaluate are:

- the properties of the core;
- the probability of staying in the core;
- if significant time is spent away from the core, the properties of the 'near orbits' of the core.

A further way of viewing the system is that each of the orbits can be regarded as representing the number of occupants in a queue (cf the Ph/Ph/1 queue [11]). Arrivals in the queue are given by 'bad' events pushing the orbit out, and departures from the queue are the system 'dissipating' the effects of the bad event and dropping into a lower orbit. This view of the effects of 'bad' events is important for understanding what would be meant by stability in such systems and could be used as a strategy for evaluating the properties of the system. One aim of such a view is to achieve a general relationship between processes and effective approaches to calculation of queues such as phase distributions [10]. If the excitation and absorbing behaviour of the system are sufficiently uniform then it may be describable as such a queueing system and consequently relatively trivial to calculate upon.

As a consequence of understanding our system this way way we have a natural approach to coping with infinite state systems. From the queueing view there are only essentially two outcomes, the system is stable in which case spends 'most' of its time near zero; or it is unstable and heads off towards infinity. Clearly, in the second case it is hard to regard this as a well functioning system. This observation was exploited to analyse the behaviour of spawning systems by Christodolou [3]. In this approach we hope that this method can apply to arbitrary systems not just spawning ones. From the perspective of automated analysis anything beyond 10^{11} states (1 bit per state at around 4Gb) could be regarded as infinite and consequently a finite valid approximation will be useful.

For large systems it is inevitable the that the graph building process will be computationally expensive. So for our approach to be effective we will desire that two goals are achieved, firstly that the construction can function 'on the fly', that is that we can build each of our desired approximations successively from the last, and that we can complete each level of approximation without work that is unnecessary. Secondly, it will be desirable to not have to use actual values, as in conventional numerical truncation techniques [12, 5]. It is important to retain the free values whilst constructing the approximation, as one of the fundamental properties we will wish to be able to evaluate is that of stability against the 'bad' event rate. The fundamental question of the

upper bound to error that a particular system design can tolerate. Whilst the symbolic (BDD) approaches to analysing large systems [1] they have the problem of the difficulty of obtaining an action order(optimality is NP hard[2] and there are practical systems (multipliers) for which there is no effective variable ordering) of choosing an order over the variables[2] (actions) in the system. Interestingly these methods could be combined with our approach to generate the successive approximations. The contrast with the BDD approach is that the 'extra' decoration we require should be essentially semantic, and follows immediately from the system designer's intent.

The remainder of this paper presents a definition of a layer filtering of automata, and how to generate such a filter on automata 'on the fly'. Present how to calculate over such filtrations and in particular estimate whether the system can be stable. A discussion on the interaction of the construction of a viable layering and the likelihood of the system design being 'good' is presented. Several examples of filtered automata built on the fly are presented, with 3D visualisations of the systems. In the examples we demonstrate how two infinite state systems can be layered, and give a large finite state system example of 500 interacting processes each of which has 1000 sequential states.

2 Filtering Automata

Consider a synchronous[3] process [8, 13, 9] defined with the triple $(\mathcal{A}, \mathcal{S}, \mathcal{R})$:

- with action $\mathcal{A}$ defined by free group over alphabet A
- states in $\mathcal{S}$
- and transitions in $\mathcal{R} \subseteq S \times A \times S$.

Definition 2.1 *An action projection $Prj(a, b)$ returns the power of action b in action a. For example $Prj(in\#live\#small, small) = 1$.*

Definition 2.2 *A path $p(S, S')$ between states S and S' as a list of pairs (a_i, S_i) $1 \leq i \leq n$such that $S \xrightarrow{a_1} S_1 \xrightarrow{a_2} S_2 \ldots \xrightarrow{a_n} S_n \cong S'$*

Definition 2.3 *A bounded projection $bprj(a, b)$ such that $bprj(a, b) = 0$ iff $Prj(a, b) = 0$, $bprj(a, b) = 1$ iff $Prg(a, b) = 1$ and $bprg(a, b) = \infty$ otherwise.*

Definition 2.4 *The depth of a path $depth(p(S, S')) = \sum_i bprj(a_i, small)$*

Definition 2.5 *Given an initial state S_0 then we define the level of an arbitrary state S' reachable from S_0 as the $Level(S)_{S_0} = min_{all\ paths\ p(S_0, S')}(depth(p(S_0, S))$, or the minimum number of transitions with $Prj(a, small) = 1$ required on a path from S_0 to S.*

[2]We are aware that there are many heuristic techniques for selecting an order, and 'on the fly' methods for re-ordering, which can ameliorate this difficulty. However, the BDD approach remains fundamentally exposed to the order problem.

[3]We exploit a synchronous view for several reasons:

1. we can add extra information to the transitions without disturbing the underlying system;
2. we can express asynchronous systems within the synchronous one;
3. we can express exact minimum times.

Definition 2.6 *We can now form the level automata* $\mathcal{S}_0$, $\mathcal{S}_1, \ldots$ *where all states* $S \in \mathcal{S}_i$ *are at level* i *with respect to an initial state* S_0.

Out intention in the sequel is to regard the transitions labelled with a *small* action as being in some sense unlikely. We do not insist at this point that any *particular* probability is associated with the transition, merely that it is labelled in this way.

We can visualise the effect of the layering as producing a system as presented in Figure 1.

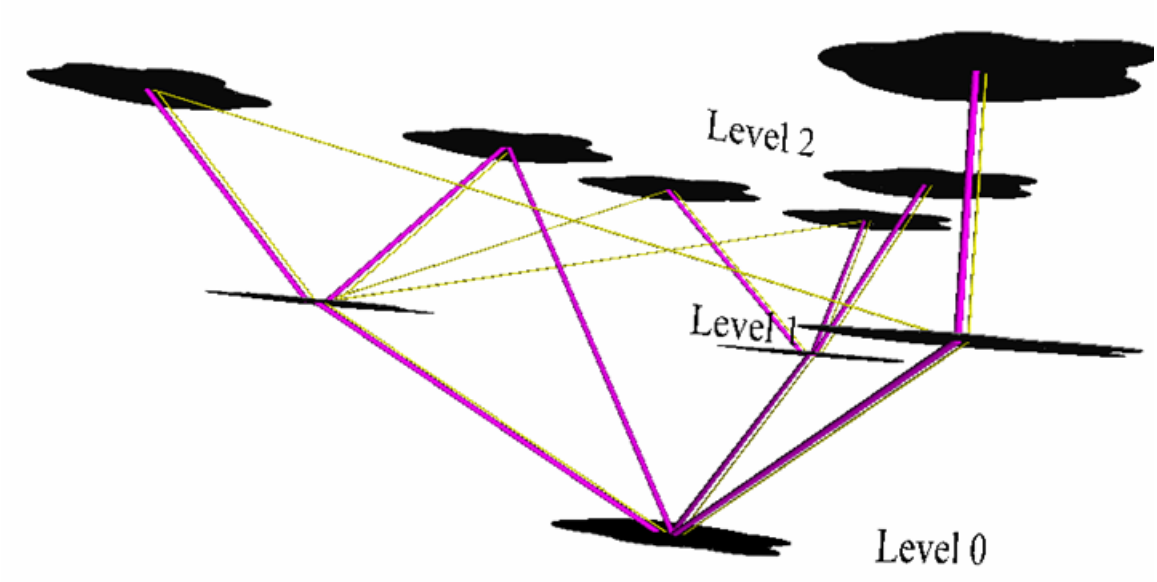

Figure 1: An automata after layering. The clouds represent states at the same level with non-small labelled transitions between them. The purple lines are the transitions down wards between layers, and do not have small markings. The yellow lines represent the small transitions upwards between the layers.

After stratification we can get essentially three outcomes (see Figure 2): a good stratification, a bad one or an unclear one. The representations of these outcomes is simplified in Figure 2, by considering that there is only one cloud of strongly connected states at each layer. Whilst we do not necessarily get an outcome as simple as the ones presented here, nonetheless, as the examples presented in the sequel demonstrate we do usually obtain a usable (good) stratification. Even when the stratification can be unclear, sometimes this is limited to a layer, and then subsequent layers have strong return likelihoods, for instance the structure of the alternating bit example[7, 9].

This stratification of our system gives rise to a transition matrix of the following form

$$A = \left\| \begin{array}{cccc} P_{00} & U_{10} & \ldots & U_{n0} \\ L_{01} & P_{11} & \ldots & U_{n1} \\ \vdots & \vdots & \ldots & \vdots \\ L_{0n} & L_{1n} & \ldots & P_{nn} \end{array} \right\|$$

alternatively $A = P + U + \epsilon C$, with C representing the lower values. Note that this is different to the nearly decomposable [12] systems of Simon and Ando. This difference is mainly important in that systems of our form will 'tend' to try and return to the states in P_{00} and hence the long run system behaviour will essentially be that of P_{00} for sufficiently small ϵ. The main aim of the 'nearly

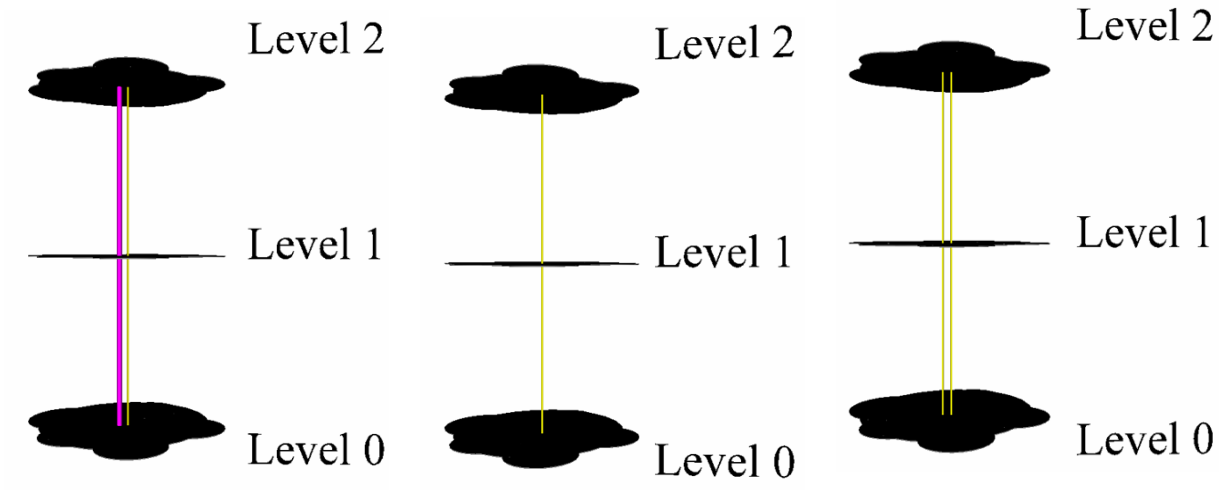

Figure 2: Three outcomes of layering. From left to right: a good outcome, low probability transitions upwards, high one downward; bad outcome, low probability outward throughout (note layering guarantees high probability is within layer or downwards **only**); unclear outcome, low probability transitions in both directions.

decomposable' systems was to demonstrate the effectiveness of indices to represent the behaviour of an economy. In that case the duration of the separate stability for each of the one diagonal square matrices is important. In our instance the important factor is that the system will largely be defined by the first of the matrices, namely P_{00}. If we can generate P_{00} with a limited amount of work and show that the system has the structure required when constructed up to some bounding point, say P_{ii}, then we can conclude **long run** properties of the system, without having to build the complete state space.

3 Generation 'on the fly'

Consider a (W)SCCS [8, 9, 14, 15] system constructed as follows:

$$P_1 \times P_2 \ldots P_n \lceil Perm$$

which we choose as S_0. We can then construct the systems state space in the following fashion:

1. take $fr = \emptyset$;
2. construct the set of one transition reachable states[4] $S_O \xrightarrow{a_i} Q_i$ from the current state;
3. partition the Q_i into those reachable with $Prj(a_i, small) = 0$[5], states $\mathcal{Q}_0$;
   those with $Prj(a_i, small) = 1$, states $\mathcal{Q}_1$;
   and those with $Prj(a_i, small) > 1$, states $\mathcal{Q}_2$;
   treat each of these state types as follows:

[4]It is worth exploiting AC equivalence to keep this set of states as small as possible [17, 20]. This exploits the simple lumpability (bisimilarity) in the system automatically.

[5]I chose the name *small* as this work started from a relative rate view. Perhaps a better choice of label would be *unwanted* or *unlikely*, implying less about numerical values. However, *small* is now embodied in the tool support, so...

4. store states and transitions $\mathcal{Q}_0$ and use as starting points for further state construction;
5. store states and transitions $fr := fr \cup \mathcal{Q}_1 - \mathcal{Q}_0$, save states as frontier for construction of next level;
6. discard states $\mathcal{Q}_2$;
7. continue from 2, until no new states reachable with $Prj(a_i, small) = 0$;
8. terminate noting the union of all the one reachable states, fr, found.

At this point we have the collection of 'zero' reachable states from S_0, and a frontier fr of 'one' reachable states. So we have $\mathcal{S}_0$, initiating the above algorithm with *all* states drawn from fr, we then construct $\mathcal{S}_1$ and a new 'frontier'. This process can be continued for as many layers as desired.

3.1 Approximate complexity

Given our system is defined as follows:

$$P_1 \times P_2 \ldots P_n \lceil Perm$$

and that each of the automata P_i has a maximum number of states σ_i then if we have σ_0 states in layer 0, that is the ground state machine has this many states. If introducing an error into the system has a bounded affect, it has no 'knock on' effects propagating disorder through the system. Then if we have t distinguishable processes amongst the P_i we would expect to require $tmax(\sigma_i)\sigma_0$ states to capture this level. This follows as only 1 as a time of the processes P_i can be disturbed by a *small* action, as a consequence we can construct a new set of states given by its maximum state space and the size of the underlying stable space for the remainder of the system. Obviously, if each of the distinguishable processes can itself by impacted by a 'bad' event then we gain a new orbit for each of these processes separately. Following the same line of reasoning we would expect to have $tmax(\sigma_i)^i\sigma_0$ states in layer i of the automata. Hence the important observation that useful calculation can be undertaken with only layer 0 and 1 constructed.

As a counter observation, if the introduction of 1 error causes a 'large disturbance' to the system then it could be reasonable to believe that the system is poorly designed. If one subscribes to this view then well designed systems will be analysable, and if the system cannot be effectively analysed it is inherently poorly designed! This phenomena can be seen in the respective examples in subsections 5.5 and 5.4.

4 Calculating with layered automata

Primarily we an interested in calculating properties of systems. Evidently for these properties to be meaningful the system must be 'stable'. That is the probability of it being outwith the approximating layered automata must be small. Further, we then need to be able to calculate arbitrary properties of the systems, often referred to as rewards.

4.1 Stability

The primary question is whether the system spend most of its time in the core, where presumably it is both functioning correctly and stably. A first approximation to this can be obtained from the first two layers $\mathcal{S}_0$ and $\mathcal{S}_1$ of the graph. Using standard numerical or symbolic approximation techniques

we can calculate P_0 and P_1, the overall probabilities of being at layer 0 or 1 respectively. If $P_0 > P_1$[6] then the system can be stable, otherwise it will not be. As we did not require that any variables were bound at graph construction time then for the cost of re-evaluating the stable distributions we can explore the various variables in the system to establish the level of error tolerance of the system. A potential general condition for when to stop building the layered system is when the first 'strong' (not *small*) transition back into layer 0 is observed. If within a relatively few layers no such transition can be found then the system is unlikely to be stable.

Theorem 4.1 *The set of states and internal transitions $\mathcal{S}_0$ and $\mathcal{S}_1$ is minimal for establishing the approximates P_0 and P_1 to establish their relative values.*

> **Proof:** The condition is certainly sufficient, can calculate the stable distribution for the joint state space and then simple arithmetic establishes the required result. It is necessary, without the complete construction of $\mathcal{S}_1$ there is no reason to exclude the possibility of an overlooked absorbing state within this layer, consequently any approximation derived from this smaller model can not be guaranteed to give the correct approximate relationship.

Corollary 4.2 *The sets of states $\mathcal{S}_0,\ \mathcal{S}_1,\ \ldots,\ \mathcal{S}_n$ will be minimal for evaluating the approximations $P_0,\ P_1,\ \ldots,\ P_n$ for the level probabilities.*

4.2 Rewards

For any reward R (usually denoted as a filtered on action output, e.g. the expected number of *success* actions observed) in a synchronous process algebra, we can calculate the expected reward for the layered automata in the obvious manner. Simply calculate the stable probabilities for each state and the reward at each state and calculate the reward accordingly.

5 Examples

Our examples are presented in the input syntax (see Appendix B) of the tool[7] which was used to generate the layered automata. This is an 'ascii-ized' version of the syntax of WSCCS [17].

5.1 Direct Example

This sequential automaton is presented to outline how these systems can be visualised, as the layers can be directly determined from its definition:

```
actions small
variables p

bs L01 1.small:L11 + 1.t:L02
bs L02 p.t:L02 + (1-p).t:L03
bs L03 1.t:L01
```

[6] Clearly we can stop building at any layer level n and then the same dominance of zero condition would apply for potential stability.

[7] This tool was used as I had easy access to the source, in principle the technique could be applied to any graph building system. Although labelling the transitions within an asynchronous system may prove more challenging.

```
bs L11 p.t:L12 + (1-p).t:L11
bs L12 1.small:L21 + 1.t:L11 + 1.t:L03

bs L21 1.t:L22
bs L22 1.small:L31 + 1.t:L23
bs L23 1.t:L24
bs L24 1.t:L12

bs L31 1.t:L31 + 1.t:L21

basi P small

btr SYS L01/P
```

After the automata `SYS` has been layered, the various layers are visualised using a 3D VRML viewer[8]. A slice through this view is presented in Figure 3. In this view transitions within a layer are black, small transitions are yellow (they can be either upwards or downwards), transitions between layers that are not small are purple (these will **always** be downwards). The layers of the automata are coloured green, blue, red in rotating order starting at layer zero.

5.2 Simple Queue

In this example a spawner generates queue items. These use a server, denoted as a resource with availability 1 (see [20] for an explanation of this approach to representing semaphores). The arrival of a new queue item is marked as a perturbing event to the system.

```
res get 1

actions get,done,small
variables pq,p1q,p,q,r

bs QIW q@1.get:T + 1-q@1.get:QIR + 1.t:QIW
bs QIR q.get:T + (1-q).get:QIR

bpa SP QIW|SPER
bpa SP1 QIR|SPER
bs SPER pq.get#small:SPER + p1q.get#small:SP1 + p.small:SP + (1-p).t:SPER

basi P get,small

btr SYS SPER/P
```

The layered view of the system unfolds as one would expect (and the first 6 layers of this infinite systems are presented in Figure 4), apart from a lone state in layer 2 representing the simultaneous take up of the service and arrival of the next item in the queue. Also importantly this approach has controlled the infinite state nature of this system. Used in this way, the layer approach can be considered as the 'action dual' to the state based approach to controlling infinite state systems of

[8]The full worlds for all of the examples are available at www-uk.hpl.hp.com/people/chrof

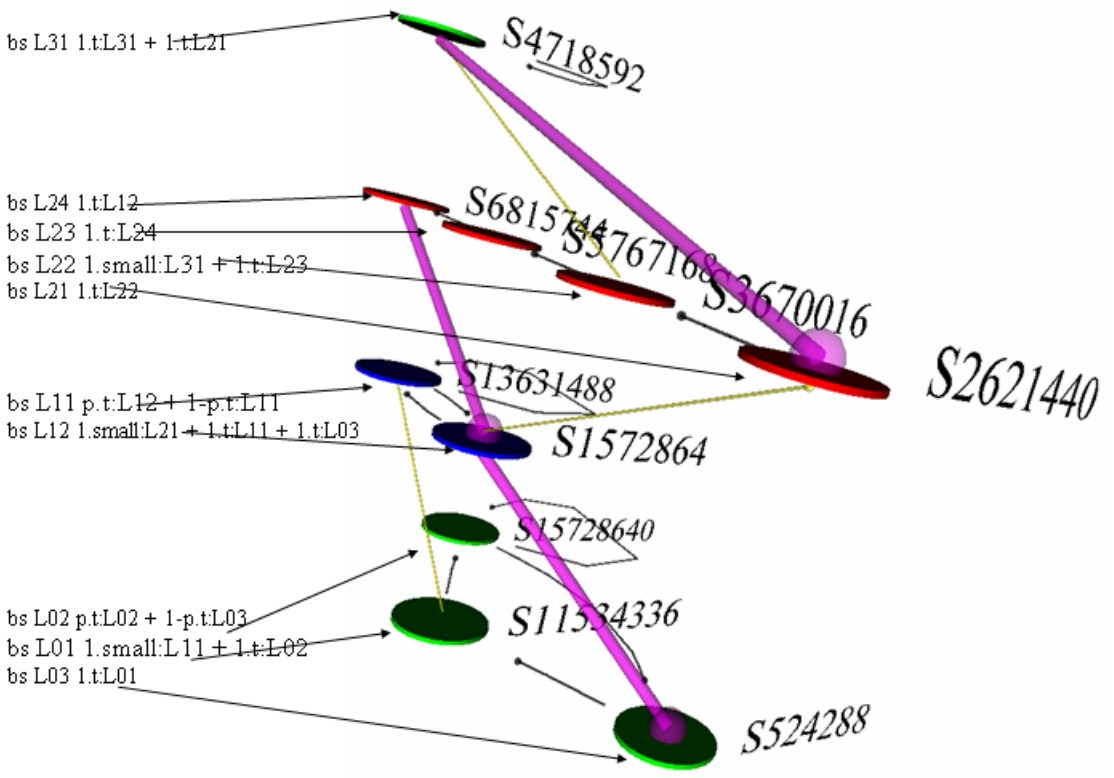

Figure 3: The layer view of the simple automata above, with each of the state definitions pointing to its representation in the layered system.

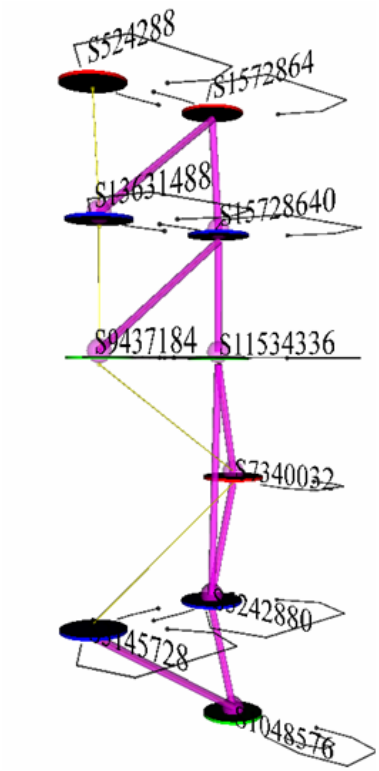

Figure 4: The layer view of the simple queue system defined above.

Christodoulou [3]. To keep the labels short the internally generated state numbers are used to label the states. The colour indicating the state level tends to give sufficient information to comprehend the systems.

5.3 Phase Queue[11]

An extension of the simple queue, which permits the representation of more complex distributions for arrivals and departures is the phase distribution based queue. A process view of this queue system is presented below.

```
res get 1

actions get,done,small
variables p00,p01,p02,p10,p11,p12,q01,q00,q02,q11,q10,q12,r

bs QIW q02@1.get:T + q00@1.get:QIR + q01@1.get:QIR  + 1.t:QIW
bs QIR q02.get:T + q00.get:QIR + q01.get:QIR1
bs QIR1 q12.get:T + q10.get:QIR + q11.get:QIR1

bpa SP QIW|SPER
bs SPER p02.small:SP + p00.t:SPER + p01.t:SPER1
bs SPER1 p11.t:SPER1 + p10.t:SPER + p12.small:SP

basi P get,small

btr SYS SPER/P
```

In Figure 5 the layers represent the product automata for arrivals and departures as the queue grows in length.

5.4 Pipe asynchrony 1

This problem is of interleaved competition for a resource (a memory bus) for processes of a functional nature. They use the bus to get data, compute on the data and then return the results. The computation period is long enough that all of the other processes *can* access the bus in the 'quiet' period. In this example we examine the effect of a perturbation in the running time of the compute phase. One (or more) of the processes can be late in returning their results. Note that in this simple example, as the computation window must be large enough to permit all of the other accesses to take place, the number of states of each entity is a function of the size of the system.

```
res get 1

actions get,done,big,small
variables p

bs STG 1@1.get:ST0 + 1.t:STG
bs ST0 1.t:ST1
bs ST1 1.t:ST2
bs ST2 1.t:ST3
```

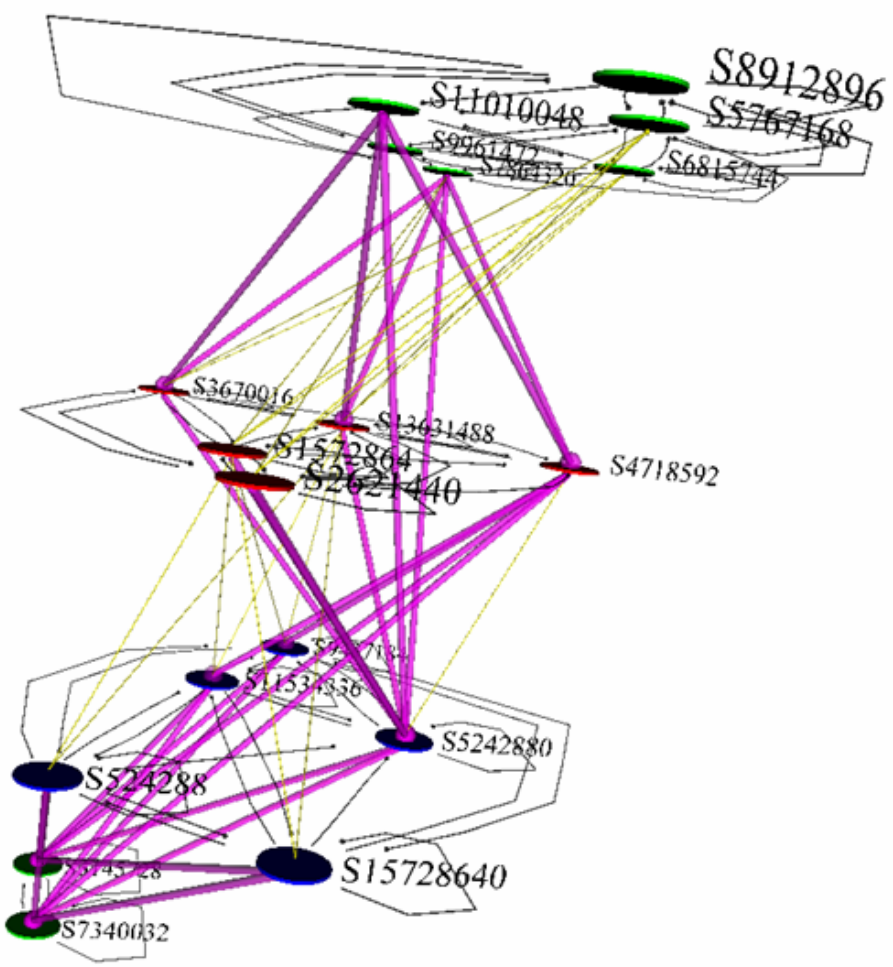

Figure 5: The first four layers of the phase queue system.

```
bs ST3 1-p.t:STP + p.small:ST3
bs STP 1@1.get#done:STG +1.t:STP

basi P get,done,big,small

btr SYS STG|ST1|ST3/P
```

The complete state space is expanded in layered form in Figure 6.

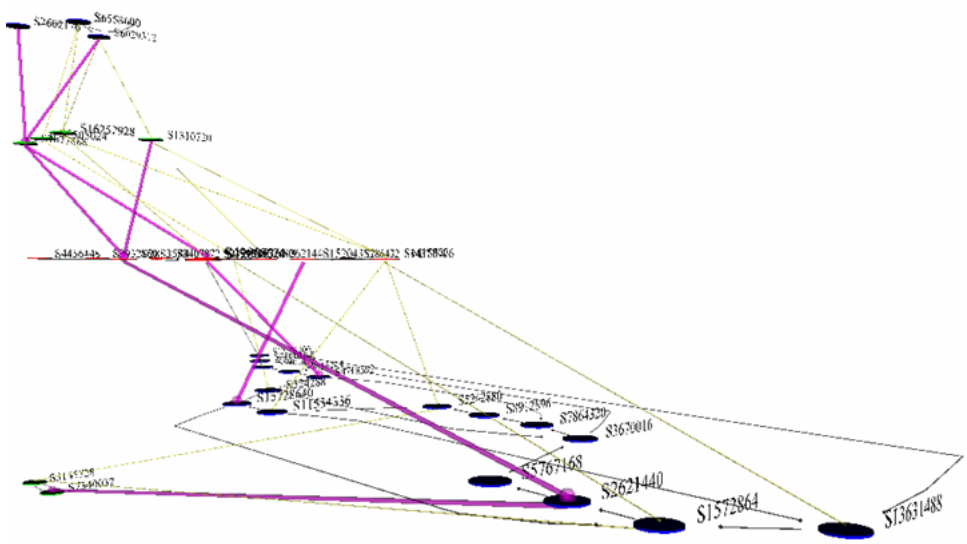

Figure 6: A layered system with knock on error effects.

5.5 Pipe asynchrony 2

Changing the system so the 'write back - read next' accesses are tied together has a dramatic effect on the behaviour of the system.

```
res get 1

actions get,done,big,small
variables p

bs STG 1@1.get:ST0          ******NO ASYNC HERE*********************
bs ST0 1.t:ST1
bs ST1 1.t:ST2
bs ST2 1.t:ST3
bs ST3 1-p.t:STP + p.small:ST3
bs STP 1@1.get#done:STG +1.t:STP

basi P get,done,big,small
```

```
btr SYS STG|ST1|ST3/P
```

In the layer view (Figure 7) we can see a deterministic run of four states (at the first level, blue) which lead inevitably back to the synchronised lower states.

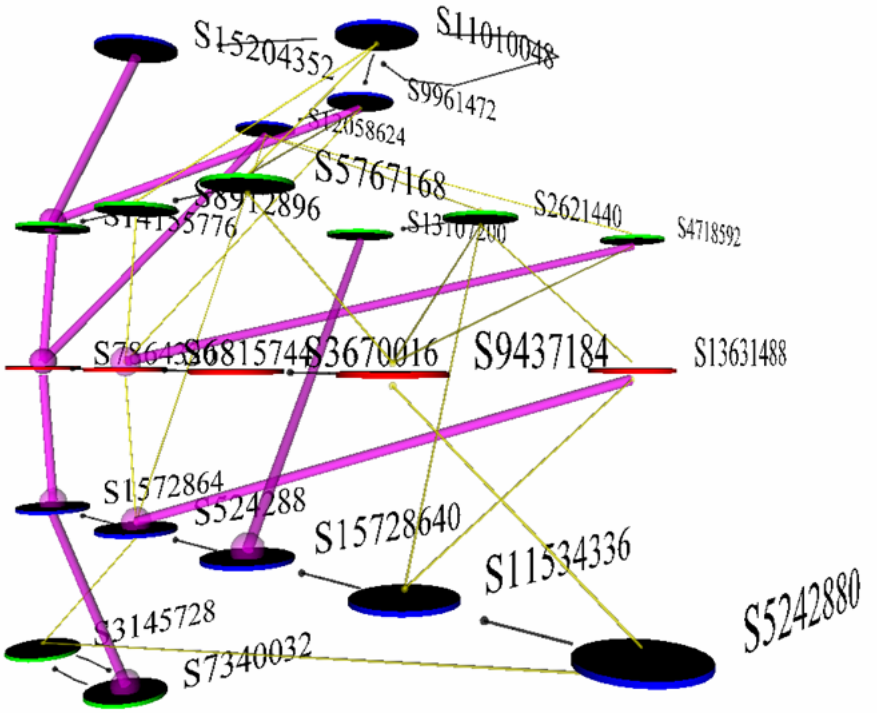

Figure 7: The complete asynchrony system without knock on error effects.

Since this example scales well it was chose for timing. The version with 500 objects competing for the resource was constructed (see Appendix A). The ground level (zero) was constructed in 32000 micro seconds. On 2.8Ghz dual processor machine with 4Gb of memory. It took 9.614 seconds to build the first layer of the approximation (about 999 states, another 1500 next approximation states were found). The next layer took 18.156 seconds (producing about 1496 new states, with about 2000 on the frontier). Importance of the construction is shown by the fact that the third level left a 4Gb machine unable to allocate sufficient memory. This system has raw 10^{1500} states, and up to AC, all of these are actually reachable, which means that despite appearances it is a far from being simple. Interestingly, whilst the core of our system is small, its reachability is the complete state space. This is a consequence of the constraints on the system actually being very weak when compared with those of [6]. For systems with greater interaction this construction may grow more slowly as the number of reachable states grow more slowly.

5.6 Ant synchrony

This is a model of ant nests synchronising (see [16]) for a full explanation. This version has the possibility of ants waking up too early and consequently dropping out of the cycle.

```
*Ant synchrony model with potential for error
```

```
*Chris Tofts

actions a,wake,small
variables p,q


bs Ant 1.a:Ant1
bs Ant1 1.a:Ant2
bs Ant2 1.t:Ant3
bs Ant3 1.t:Ant4
bs Ant4 1.t:Ant5
bs Ant5 1.t:Ant6
bs Ant6 1.t:Ant7
bs Ant7 1.t:Ant8
bs Ant8 1.t:Ant9
bs Ant9 1.t:Ant10
bs Ant10 1.t:Ant11
bs Ant11 1.t:Ant12
bs Ant12 1.t:Ant13
bs Ant13 1.t:Ant14
bs Ant14 1.t:Ant15
bs Ant15 1.t:Ant16
bs Ant16 1.t:Ant17
bs Ant17 1.t:Ant18 + 1.small:ACW
bs Ant18 1.t:Ant19
bs Ant19 1.t:ACW

bs ACW p.t:AAW + (1-p).t:ANW
bs ANW 1.t:ACW + 1.wake^-1:Ant

bs AAW  1@15.wake^15:Ant \
      + 1@14.wake^14:Ant + 1@13.wake^13:Ant \
      + 1@12.wake^12:Ant + 1@11.wake^11:Ant \
      + 1@10.wake^10:Ant + 1@9.wake^9:Ant \
      + 1@8.wake^8:Ant + 1@7.wake^7:Ant \
      + 1@6.wake^6:Ant + 1@5.wake^5:Ant \
      + 1@4.wake^4:Ant + 1@3.wake^3:Ant \
      + 1@2.wake^2:Ant + 1@1.wake^1:Ant \
      + 1.t:Ant

basi Allow a,small

btr SSys Ant6|Ant6|Ant6|Ant6|Ant6|Ant6|Ant6|Ant6/Allow
```

In the layer diagram (Figure 8) the green states are the base cycle, the blue states represent 9 different versions of perturbed cycles as a consequence of the first perturbation, this represent the various subgroups of synchronised ants which can form before returning to the main cycle.

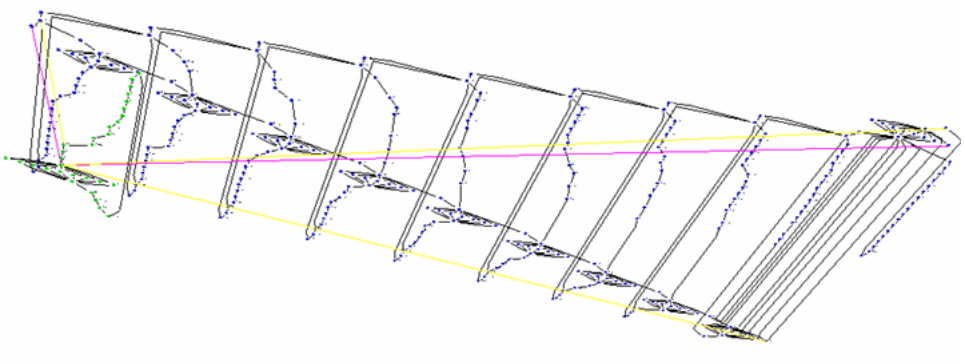

Figure 8: The layer view of the ant synchrony model defined above.

5.7 Alternating bit protocol

In this version of the alternating bit protocol, we regard transmission errors, and retries as 'bad' events and decorate the arcs accordingly.

```
*Probabilistic alternating bit protocol
*
*Chris Tofts 9/8/94 after CWB versions,
*very abstract version
*

actions s0,s1,r0,r1,rack0,rack1,sack0,sack1,send,receive,succ,small
variables rt,rte,err,ere

*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#small:Sa1 + rt-rte.small: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#small:S11 + rt-rte.small:S11 + 1-rt-rte.t:S12
bs Sas 1.succ:Sa

*the reciever for all of our endevours....

bs Ra  1.r0:Rar1+1.r1#small:Ra2 + rt-rte.small: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#small:R11 + rt-rte.small: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.small:Ml1

bs Ml1a 1@1.r0^-1:Ml1 + 1.t:Ml1a

bs Ml110 1-err-ere.t:Ml11 + err-ere.small:Ml1

bs Ml11 1@1.r1^-1:Ml1 + 1.t:Ml11

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

bs Ml2 1.sack0:Ml2a1 + 1.sack1:Ml211 + 1.t:Ml2

bs Ml2a1 1-err-ere.t:Ml2a + err-ere.small:Ml2

bs Ml2a 1@1.rack0^-1:Ml2 + 1.t:Ml2a

bs Ml211 1-err-ere.t:Ml21 + err-ere.small:Ml2

bs Ml21 1@1.rack1^-1:Ml2 + 1.t:Ml21
```

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

basi Allow send, receive, succ, small

*this is the complete system

btr ABP Ra|Ml1|Ml2|Sa/Allow

*cle();rf"Examples/abpVAP.pro";
```

In the layer diagram (Figure 9) we can see the base cycle of the succesful transmission of the signal, followed by layers caused by errors, which then need to wait for further 'bad' things, retries, to occur before they are corrected.

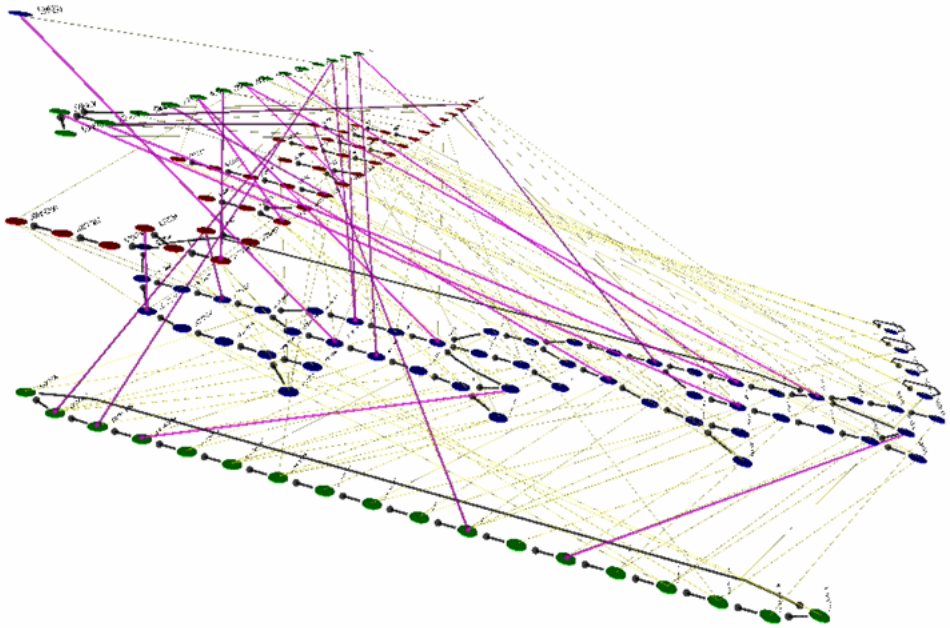

Figure 9: The layer view of the alternating bit protocol defined above.

6 Conclusions

The layered approach to synchronous automata presented appears to be calculationally effective, generating the first three layers of our large example within a total of 30 seconds. The demands it places on the system analyser, of indentifying a 'core' state, and labelling some transitions as having the property small do not appear to be too onerous. It offers the prospect of system analysis that scales by the square of the individual process size, not by the number of components in the system.

Interestingly when such scaling does not occur, it seems reasonable to immediately conclude that the system design is poor and that a better one should be sought, obviating the need for any further analysis. The analogy with excitation systems and queue systems seems to be a strong one, and this could lead to further calculational improvements. It appears to give rise to an interesting class of systems, which avoid the long term dominance of the small seen in the near decomposable class [12].

The layering can be achieved without invoking any numerical relationship at all, this avoiding chosing bounds for which transitions are in or out [5]. This has the pleasant property of correctly layering systems even when the values on the transitions could be comparable (all though evidently ordered) such as the stable simple queueing example. Used in this way it can also help build finite approximations to infinite systems in a more general fashion than that of [3].

For future work an interesting problem would be the location of a small core by simulation techniques [6], followed up by the efficeint automated placement of the *small* transitions to preserve the core. Estimating the difficulty of this task as compares with the order problem for BDDs [2] might be an interesting point of comparison for these techniques. Further, tool development whereby a user can indicate the placement of the *small* actions whilst traversing the core might also simplify usage of this approximation technique.

Acknowledgements

Contributions made by Matthew Collinson, Melanie Hatcher, Brian Monahan, David Pym, Richard Taylor and Mike Yearworth immensely improved this paper, any remaining mistakes are mine, all mine!

References

[1] J. Burch, E. Clarke, K. McMillan, D. Dill, L. Hwang, Symbolic model checking 10^{20} states and beyond, LICS' 90, 428-439, 1990.

[2] R.E. Byant, On the complexity of VLSI implementations and graph representations of boolean functions with application to integer multiplication, IEEE Transactions on Computers, 40(2):205-213, February 1991.

[3] A. Christodolou, Formal Solutions to Large Scale Performance Problems, PhD Thesis, University of Leeds, 1999.

[4] R. Cleaveland, J. Parrow, and B. Steffen. The Concurrency Workbench: A Semantics-Based Tool for the Verification of Concurrent Systems. *ACM Transactions on Programming Languages and Systems*, 15(1), 1993.

[5] H, Hermanns, U. Herzog, U. Klehmet, V. Mersiotakis, M. Siegle, Compositional performance modelling with the TIPPtool, Performance Evaluation 39:5-35, 2000.

[6] S. Kaufmann, Homeostasis and Differentiation in Random Genetic Networks, Nature 224:177-178, October 1969.

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

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

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

[10] M.F. Neuts, Matrix-Geometric solutions in stochastic models, John Hopkins University Press, 1981.

[11] M.F. Neuts, Structured Stochastic Matrices of M/G/1 Type and their Applications, volume 4 of Probability: Pure and Applied. Marcel Dekker, New York, 1989

[12] H.Simon & A. Ando, Aggregation of variables in dynamic systems, Econometrica 29:111-138, 1961

[13] R. de Simone, Higher-level synchronising devices in Meije-SCCS. *Transactions in Computer Science* **37**, 245-267, 1985.

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

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

[16] C. Tofts, Using Process Algebra to Describe Social Insect Behaviour, Transactions of the Society for Computer Simulation, 227-283, 1992.

[17] C. Tofts, Analytic and locally approximate solutions to properties of probabilistic processes, Proceedings TACAS '95, LNCS 1019, 1995.

[18] C, Tofts, Exact, analytic, and locally approximate solutions to discrete event-simulation problems. Simulation Practice and Theory, pp. 721-759(39), Volume 6, Number 8, 15 December 1998.

[19] C. Tofts, Symbolic approaches to probability distributions in process algebra, BCS FACS 12:392-415, 2000.

[20] C. Tofts, Efficiently Modelling Resource in a Process Algebra, Laboratory report number HPL-2003-181, 2003

A Large example

The process definition for the very large example is included for completeness.

```
res get 1

actions get,done,big,small
variables p

bs STG 1@1.get:ST0
bs ST0 1.t:ST1
bs ST1 1.t:ST2
.
.
.
bs ST996 1.t:ST997
bs ST997 1-p.t:STP + p.small:ST997
bs STP 1@1.get#done:STG +1.t:STP
```

```
basi P get,done,big,small

btr SYS STG|ST1|ST3|ST5|ST7|ST9|ST11|ST13|ST15|ST17|ST19|ST21|ST23|ST25|ST27|ST29|ST31|ST33|ST35|ST37|\
ST39|ST41|ST43|ST45|ST47|ST49|ST51|ST53|ST55|ST57|ST59|ST61|ST63|ST65|ST67|ST69|ST71|ST73|ST75|\
ST77|ST79|ST81|ST83|ST85|ST87|ST89|ST91|ST93|ST95|ST97|ST99|ST101|ST103|ST105|ST107|ST109|ST111|\
ST113|ST115|ST117|ST119|ST121|ST123|ST125|ST127|ST129|ST131|ST133|ST135|ST137|ST139|ST141|ST143|\
ST145|ST147|ST149|ST151|ST153|ST155|ST157|ST159|ST161|ST163|ST165|ST167|ST169|ST171|ST173|ST175|\
ST177|ST179|ST181|ST183|ST185|ST187|ST189|ST191|ST193|ST195|ST197|ST199|ST201|ST203|ST205|ST207|\
ST209|ST211|ST213|ST215|ST217|ST219|ST221|ST223|ST225|ST227|ST229|ST231|ST233|ST235|ST237|ST239|\
ST241|ST243|ST245|ST247|ST249|ST251|ST253|ST255|ST257|ST259|ST261|ST263|ST265|ST267|ST269|ST271|\
ST273|ST275|ST277|ST279|ST281|ST283|ST285|ST287|ST289|ST291|ST293|ST295|ST297|ST299|ST301|ST303|\
ST305|ST307|ST309|ST311|ST313|ST315|ST317|ST319|ST321|ST323|ST325|ST327|ST329|ST331|ST333|ST335|\
ST337|ST339|ST341|ST343|ST345|ST347|ST349|ST351|ST353|ST355|ST357|ST359|ST361|ST363|ST365|ST367|\
ST369|ST371|ST373|ST375|ST377|ST379|ST381|ST383|ST385|ST387|ST389|ST391|ST393|ST395|ST397|ST399|\
ST401|ST403|ST405|ST407|ST409|ST411|ST413|ST415|ST417|ST419|ST421|ST423|ST425|ST427|ST429|ST431|\
ST433|ST435|ST437|ST439|ST441|ST443|ST445|ST447|ST449|ST451|ST453|ST455|ST457|ST459|ST461|ST463|\
ST465|ST467|ST469|ST471|ST473|ST475|ST477|ST479|ST481|ST483|ST485|ST487|ST489|ST491|ST493|ST495|\
ST497|ST499|ST501|ST503|ST505|ST507|ST509|ST511|ST513|ST515|ST517|ST519|ST521|ST523|ST525|ST527|\
ST529|ST531|ST533|ST535|ST537|ST539|ST541|ST543|ST545|ST547|ST549|ST551|ST553|ST555|ST557|ST559|\
ST561|ST563|ST565|ST567|ST569|ST571|ST573|ST575|ST577|ST579|ST581|ST583|ST585|ST587|ST589|ST591|\
ST593|ST595|ST597|ST599|ST601|ST603|ST605|ST607|ST609|ST611|ST613|ST615|ST617|ST619|ST621|ST623|\
ST625|ST627|ST629|ST631|ST633|ST635|ST637|ST639|ST641|ST643|ST645|ST647|ST649|ST651|ST653|ST655|\
ST657|ST659|ST661|ST663|ST665|ST667|ST669|ST671|ST673|ST675|ST677|ST679|ST681|ST683|ST685|ST687|\
ST689|ST691|ST693|ST695|ST697|ST699|ST701|ST703|ST705|ST707|ST709|ST711|ST713|ST715|ST717|ST719|\
ST721|ST723|ST725|ST727|ST729|ST731|ST733|ST735|ST737|ST739|ST741|ST743|ST745|ST747|ST749|ST751|\
ST753|ST755|ST757|ST759|ST761|ST763|ST765|ST767|ST769|ST771|ST773|ST775|ST777|ST779|ST781|ST783|\
ST785|ST787|ST789|ST791|ST793|ST795|ST797|ST799|ST801|ST803|ST805|ST807|ST809|ST811|ST813|ST815|\
ST817|ST819|ST821|ST823|ST825|ST827|ST829|ST831|ST833|ST835|ST837|ST839|ST841|ST843|ST845|ST847|\
ST849|ST851|ST853|ST855|ST857|ST859|ST861|ST863|ST865|ST867|ST869|ST871|ST873|ST875|ST877|ST879|\
ST881|ST883|ST885|ST887|ST889|ST891|ST893|ST895|ST897|ST899|ST901|ST903|ST905|ST907|ST909|ST911|\
ST913|ST915|ST917|ST919|ST921|ST923|ST925|ST927|ST929|ST931|ST933|ST935|ST937|ST939|ST941|ST943|\
ST945|ST947|ST949|ST951|ST953|ST955|ST957|ST959|ST961|ST963|ST965|ST967|ST969|ST971|ST973|ST975|\
ST977|ST979|ST981|ST983|ST985|ST987|ST989|ST991|ST993|ST995|ST997/P
```

B Probabilistic Workbench (prwb)

This is a brief introduction to the syntax of process presentation in prwb [17, 18].

B.1 Weights

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

$$\begin{array}{rcl} e & ::= & n|string|(e') \\ e' & ::= & n|string|decimal|e'+e'|e'-e'|e'*e'|e^n|(e') \end{array}$$

So the following are weights:

```
5
(0.005)
(1-p * q)
5@2
(1-p)@3
```

the last two being weights at priority level 2 and 3 respectively. Note that we do not allow symbolic priorities, as this would actually affect the computational structure.

B.2 Actions

Actions are defined as products of powers of strings;

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

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`.

B.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 absorption of equivalent state in parallel compositions:

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`

There are two special process names `N` denoting a process which undertakes no actions, and `T` which denotes a process which will only undertake *tick* actions.