A Model Checking Algorithm for Stochastic Systems

Jeremy Bryans, Howard Bowman and John Derrick

1 Introduction

In this report we present an algorithm for model-checking stochastic automata with respect to a probabilistic temporal logic. We consider the stochastic automata presented in [D'A99]. In particular, the algorithm is novel in that it allows any (continuous) probability density functions to be used in the automaton (not just exponential ones).

This report is structured as follows. In Section 2 we introduce stochastic automata, which forms the system description language for the model checking algorithm. In Section 3 we introduce and explain the simple probabilistic temporal logic which forms the query language in the model checking algorithm. In Section 4 we present an overview of the algorithm, together with the data structures and variables used. In Section 5 we consider the twin questions of correctness and convergence: does the result pass (fail) imply that the automaton models (does not model) the formula? and secondly for any automaton and bounded until formula, is it possible to run the algorithm with a small enough timestep so that the result undecided is given only in arbitrarily few cases?

2 Stochastic Automata

As defined in [D'A99], a *stochastic automaton* is a structure $SA = (\mathcal{S}, \mathcal{C}, \mathcal{A}, \twoheadrightarrow, \kappa)$ where

- $\mathcal{S}$ is a set of *locations.*
- $\mathcal{C}$ is a set of *random clocks.* Each $x \in \mathcal{C}$ is a random variable with *distribution function* F_x.
- A is a set of *actions.*
- $\twoheadrightarrow \subseteq \mathcal{S} \times (\mathcal{A} \times \mathbb{P}_{fin}(\mathcal{C})) \times \mathcal{S}$ is the set of *edges.*

- $\kappa : \mathcal{S} \to \mathbb{P}_{fin}(\mathcal{C})$ is the *clock setting function.*

We will denote $(s, a, C, s') \in \twoheadrightarrow$ by $s \xrightarrow{a,C} s'$.

For a full explanation of Stochastic Automata, see [D'A99].

In this report, we will use a slightly simplified form of the stochastic automata, simplified in the following ways.

- each clock has a lower bound on the range to which it may be set[1]. The minimum of these is the maximum permissible value of time step.
- each clock has an upper bound on the value to which it may be set; for a clock c this is given by $ub(c)$.
- clocks are used only on transitions emanating from the states in which they have been set.
- all clocks set in a state must be consumed by at least one transition from that state[2].
- there is one clock on each transition.

3 A Probabilistic Real-Time Logic

In this section, we introduce a simple probabilistic temporal logic. The purpose of the logic is to express properties that we wish to check the stochastic automaton against. The logic we define allows us to check a range of such properties.

We will use adversaries to resolve the nondeterministic (as opposed to probabilistic) choices in the automaton.(see for example [BK98]). An adversary of a stochastic automaton can be thought of as a scheduler, which resolves any nondeterministic choices which the stochastic automaton must make. An adversary may vary it's behaviour according to the previous behaviour of the automaton.

We assume that when we wish to model check a property against an automaton, we are also given an adversary to resolve the nondeterminism

[1]A beneficial consequence of this assumption is that it (in a rather strong way) ensures time guardedness of the automaton, and thus prevents zeno behaviour.

[2]In the case where the same clock is consumed by more than one transition, then we resolve the non-deterministic choice that arises using adversaries.

within the automaton. We can now, for example, answer such questions as "Given a stochastic automaton and an adversary, is the probability of a *success* event greater than 0.8?".

The syntax of our logic is

$$\psi ::= \mathsf{tt} \mid \mathsf{ap} \mid \neg\, \psi \mid \psi_1 \wedge \psi_2 \mid [\phi_1\, \mathcal{U}_{\sim c}\, \phi_2] \simeq p$$

$$\phi ::= \mathsf{tt} \mid \mathsf{ap} \mid \neg\, \phi \mid \phi_1 \wedge \phi_2$$

where $[\phi_1\, \mathcal{U}_{\sim c}\, \phi_2] \simeq p$ is a *path formula.* The path formulae can only be used at the outermost level — they cannot be nested. This is because the model checking algorithm we give can only evaluate path formulae from the initial state.

Further: $c \in \mathcal{N}$ (natural numbers), a is an atomic proposition, $\mathsf{p} \in [0, 1]$ is a probability value, $\simeq \in \{<, >, \leq, \geq\}$ and $\sim \in \{<, \leq\}$.

With this syntax, an example of a valid formula that we can check would be $[\mathsf{tt}\ \mathcal{U}_{<10}\, \mathsf{success}] > 0.8$ which says that the probability of reaching a $\mathsf{success}$ event within 10 time units is greater than 0.8.

4 Overview of algorithm

In this section we present an overview of the algorithm, together with discriptions of the data structures which will be used. The model checking algorithm takes a stochastic automaton SA, together with a bounded until temporal logic formula TL, a time step parameter δ and an adversary *pick*. For convenience we will present only the case where TL is of the form $[a_0\, \mathcal{U}_{\leqslant time}\, a_1] > p$. Minor modifications to the algorithm would allow any of $\geqslant p$, $\leqslant p$ or $< p$. We use the atomic propositions a_0 and a_1 as part of the formula because anything more complex can be reduced to these by standard model-checking techniques. Using $\leq$ *time* guarantees that the algorithm will terminate.

A single iteration of the algorithm will return one of three results: true, false or undecided. If it returns true, then the automaton models the formula. If it returns false, then the automaton does not model the formula. If it returns undecided, then the algorithm was unable to determine whether the automaton models the formula. In this case, the algorithm can be re-applied with a smaller value for the time step δ. The question of convergence to the correct answer as δ tends to zero is discussed in section 5. For the remainder of this section we assume δ to be fixed.

A stochastic automaton has a finite number of clocks each with a probability distribution function (pdf). For each state, the set of clocks has an (arbitrary) order, and the algorithm makes use of this ordering[3]. As discussed above, we assume that each clock has non-zero lower and upper bounds on the values to which it can be set. This has been done so that δ can be initially chosen to be less than the minimum of all these lower bounds.

The algorithm works by creating a snapshot of the automaton at each time point $n\delta$ ($n \in \mathbb{N}$)[4] and extracting some global informaton about the probability of the formula $[a_0 \, \mathcal{U}_{\leqslant time} \, a_1]$ being satisfied at this point.[5] To build the next snapshot, the algorithm picks out at each time point $n\delta$ the transitions that the automaton is capable of during the next interval of length δ. Because δ is less than the minimum of all the clock lower bounds, a maximum of one transition per path[6] can occur in each interval. Recording all possible states of the automaton at each time point is therefore enough to record all the possible transitions.

The algorithm stops when either enough information has been gathered to determine the truth or falsity of the formula, or enough time has passed so that $n\delta > time$, and allowing time to pass further will make no difference to the information we already have. In this case the result undecided is returned.

4.1 Data structures

The principal data structures used by the algorithm are matrices. For each state s in the stochastic automaton we derive a matrix for a given time t (which is by definition $n\delta$), denoted $matrix(s, t)$, which is a record of the probabilities of the various combinations of clock values in state s at time t.

Each matrix $matrix(s, t)$ will have $\#\kappa(s)$ dimensions. Each dimension is associated with a particular clock, and the ordering of the dimensions

[3]However, the choice of ordering is arbitrary and does not carry any meaning. Any ordering will be sufficient.

[4]We will speak of the time instants generated by $n\delta$ ($n \in \mathbb{N}$) as time points.

[5]We also require that $\exists\, n \bullet n\delta = time$, which ensures that one of the snapshots will be at exactly time *time*.

[6]A *path* is a route through the Probabilistic Transition System which forms the semantic model of the Stochastic Automaton. For further details on Probabilistic Transition Systems see [DKB98].

corresponds to the ordering of the clocks. The dimension associated with a clock c will have $\lceil \frac{ub(c)}{\delta} \rceil$ entries, where $ub(c)$ is the largest value to which the clock c can be set, and $\lceil \frac{ub(c)}{\delta} \rceil$ is the smallest integer greater than or equal to $\frac{ub(c)}{\delta}$. For a clock c_i, we will abbreviate $\lceil \frac{ub(c_i)}{\delta} \rceil$ by N_i.

The valuation function v gives the value of a particular clock: $v(c_i)$ is the value of clock c_i.

Each entry in the matrix $\mathit{matrix}(s,t)$ is the probability that at time t, the automaton is in state s, and each clock is within a particular time range. Thus, the value $\mathit{matrix}(s,t)[k_1 \ldots k_n]$ is the probability that at time t, the automaton is in state s, and $v(c_i) \in (\delta(k_i - 1), \delta k_i]$ for each clock c_i.

A further data structure we shall need is $\mathit{live}(t)$, which is the set of states "live" at time t (i.e. their matrices at time t contain at least one non-zero entry, and the formula is still undecided). In order to get an accurate picture of the automaton at time $t + \delta$, we must take into account all states live at time t.

A *snapshot* of the automaton at time t is the set of all matrices $\mathit{matrix}(s,t)$ where s is in $\mathit{live}(t)$.

Let $pr(c_i \in (\delta(k_i - 1), \delta k_i])$ be the probability that clock c_i is initially set to a value in the range $(\delta(k_i - 1), \delta k_i]$. Before the algorithm proper begins, we calculate all these values from the clock probability distribution functions, which are entered into the algorithm as part of the stochastic automaton.

4.2 Variables

The algorithm also uses a number of auxillary variables.

$\mathit{prob}(s,t)$ is the probability of entering state s during the time range $(t - \delta, t]$, and is defined for states s live at time $t - \delta$, and s' live at time t.

$\mathit{new\_states}(s,t)$ is the set of states which can be reached from a state s during a time range $(t - \delta, t]$.

$\mathit{total\_pass}$ is a probability value. It is incremented at each iteration. The iterations of the algorithm correspond to the time points, and $\mathit{total\_pass}$ records the probability of the automaton having passed the formula at that time. $\mathit{total\_fail}$ is also a probability value; it records the probability of the automaton having failed the formula as the algorithm progresses.

error is an upper bound on the possible errors of $\mathit{total\_pass}$ and $\mathit{total\_fail}$. after an iteration, we know that the actual probability of the automaton having passed the formula is in the range $[\mathit{total\_pass}, \mathit{total\_pass} + \mathit{error}]$, and similarly for *total_fail*.

4.3 The algorithm

The matrix algorithm is given in detail in the appendix. We begin here with a pseudocode description.

```
initialise variables
build $matrix(s_0, 0)$
check formula against $s_0$ and $t = 0$          → pass
                                                  → fail
            ↓ undecided
repeat
  increment $t$
  forall locations in $live(t - \delta)$
    call procedure $new\_time\_matrix$:    (record possible new locations)
                                           (increment probability of entering new locations)
                                           (increment error)
    update $live(t)$
  forall locations in $live(t)$
    check formula against location:
      if pass then add probability to pass
      if fail then add probability to fail
      if undecided then call procedure $new\_state\_matrix$
until (formula has passed, or
      formula has failed, or
      $t$ has reached the limit set by the formula)
set all locations undecided at last iteration to false
if $pass > formulaprobability$ then output pass
elseif $fail > 1 - formulaprobability$ then output fail
else output undecided
```

We now describe the algorithm in overview, outlining the procedures involved. It begins by calculating $matrix(s_0, 0)$, where s_0 is the initial state of the stochastic automaton. If there are n clocks in state s_0, then $matrix(s_0, 0)$ is calculated using the probability distribution functions of the clocks in state s_0 as follows:

$$\begin{aligned}
&\forall\, 1 \leqslant k_1 \leqslant N_1 \\
&\qquad\vdots \\
&\forall\, 1 \leqslant k_n \leqslant N_n \bullet \mathit{matrix}(s_0,0)[k_1 \ldots k_n] := \prod_{l=1}^{n} pr(v(c_l) \in (\delta(k_l - 1), \delta k_l])
\end{aligned}$$

$\mathit{live}(0)$ will either be $\{s_0\}$ or the empty set, according to whether the formula TL is made true or false by state s_0, or whether we cannot yet decide. This is determined as follows. If state s_0 models proposition a_1, then the formula TL is immediately true and $\mathit{live}(0)$ is the empty set. Otherwise, if s_0 models a_0 we cannot yet decide, and so $\mathit{live}(0)$ contains s_0. If the state models neither proposition then the formula TL is immediately false, and $\mathit{live}(0)$ is the empty set.

If the initial step does not determine whether the formula is true or false, we perform a number of iterations. Each iteration builds the snapshot at time $t + \delta$, based upon the snapshot at time t. The sequence of snapshots build progressively more information as to whether the stochastic automaton has passed or failed the formula.

In the case of a bounded until formula with a $\leqslant t$ subscript[7], the number of iterations is finite (i.e. the algorithm always terminates) because the iterations terminate either when sufficient information has been extracted to determine whether the formula passes or fails, or after the $\frac{\mathit{time}}{\delta}$th iteration, since the formula cannot become true after time *time*.

If the information at time t is not enough to determine the truth or falsity of the formula, we build the snapshot for time $t + \delta$. We now describe an individual iteration.

An iteration consists of two sections. In the first, we consider all of the states which are currently undecided. These are all the states in $\mathit{live}(t)$. For each state we create the matrices at time $t+\delta$, update $\mathit{live}(t+\delta)$ and calculate $\mathit{prob}(s', t+\delta)$ for states s' which can be reached in the interval $(t, t+\delta]$. In the second, we look at all states which can be reached in the interval $(t, t+\delta]$, and consider them with respect to the temporal logic formula. We then either update the global probabilities, if the states cause the formula to pass or fail, otherwise we update the respective matrices.

[7]i.e. $[a_0\, \mathcal{U}_{\leqslant \mathit{time}}\, a_1] > p$, which is the only one we consider in this paper.

Note that in this algorithm a matrix is updated at most twice. Once within procedure *new_time_matrix*, if the state was live at the previous time, and once within the procedure *new_state_matrix*, if the state is reachable via a transition in the previous interval.

4.3.1 Creating and updating matrices

We begin with some necessary notation. Let us assume δ is a fixed rational number greater than zero.

Definition 1 If $c_1, \ldots, c_n$ are the clocks on state s, a *valuation*[8] is the vector of results of the valuation function $v(c_i)$ from clocks to $\mathbb{R}$ which gives the values of each of the n clocks.

Two valuations v and v' are $(\delta-)$ equivalent if

$$\forall\, c_i.\, \exists\, k_l.v(c_i) \in (\delta(k_l-1), k_l] \wedge v'(c_i) \in (\delta(k_l-1), k_l]$$

A *valuation equivalence class* (or clock configuration) is a maximal set of equivalent valuations. □

If δ is understood, we can abbreviate this configuration as $(k_1, \ldots, k_n)$. For a state s and a time t, the probability $\prod_{l=1}^{n} pr(v(c_l) \in (\delta(k_l-1), \delta k_l])$ is an (s,t)-*clock configuration probability* (or just a clock configuration probability when s and t are understood).

There are two different procedures for updating a matrix. The first (encapsulated in the procedure *new_time_matrix*) corresponds to the situation within the stochastic automaton where time passes, but the state remains unchanged. In this case we must shift the clock configuration probabilities in the previous matrix down by one index step (which corresponds to δ time passing) and add the result to the matrix we are updating.

We also at this stage determine the new states which can be reached from the current state during the δ time passing, and the probability of entering these states. We do this by looking at all the clock configurations where at least one of the indices has the value one. If the clocks are set within such a configuration then we know that at least one clock will expire during the ensuing δ time step.

[8]We overload the definition of valuation here.

If only one index in the configuration has the value one then only one clock can expire, and only one state can be entered from this clock configuration, and so that state is added to the set of states which can be entered from the current state at the current time.

If more than one index in the configuration has the value one, then we simply do not go any further into the automaton and the configuration probability is added to error.

The second way to update a matrix corresponds to a transition from one state to another within the automaton. It is described in the procedure *new_state_matrix*. For each matrix entry we calculate the clock configuration probability, multiply it by the probability of moving into this state at this time, and add it to the matrix entry we are updating.

4.3.2 Terminaton of an iteration

When the iteration terminates, it will output one of three results: true, false or undecided. true means that the automaton models the temporal formula, i.e. $\mathtt{SA} \models [a_0 \, \mathcal{U}_{\leqslant time} a_1] > p$. false means that $\mathtt{SA} \not\models [a_0 \, \mathcal{U}_{\leqslant time} a_1] > p$, and undecided means that the algorithm could not accumulate enough information to decide whether or not the automaton modeled the formula.

The algorithm makes the output decision based on the three global variables *total_pass*, *total_fail* and *error*.

total_pass is a lower bound on the probability that the stochastic automaton models the formula, and *total_fail* is a lower bound on the probability that the stochastic automaton does not model the formula. *error* is the largest amount by which *total_fail* or *total_pass* may be wrong. In a sense, it records the size of the uncertainty introduced by the choice of δ.

If neither of these situations holds then the errors introduced by the algorithm are too large to determine an answer with this value of δ. In this case, we can rerun the algorithm with a smaller δ, and in section 5 we show that the sum of the errors tends to zero as δ tends to zero. Note, however, that in the case where the probability that SA models $[a_0 \, \mathcal{U}_{<t} a_1]$ is exactly p, we cannot guarantee that there will be a δ small enough to allow the algorithm to generate a true or a false.

5 Correctness and convergence

For a single run with fixed δ, we wish to prove two things: that the algorithm terminating with pass implies that the automaton models the formula, and that the algorithm terminating with fail imples that the automaton does not model the formula.

If the algorithm outputs pass then the variable *total_pass* must be greater than p (where p is taken from the temporal formula $[a_0 \, \mathcal{U}_{\leqslant t} a_1] > p$). The only place where *total_pass* gets incremented is line 14 of section C. If the current state q models a_1 (and all previous states in the path model a_0) we add the probability of entering the state q at time ct. If the sum of these probabilities is greater than p then the algorithm outputs pass.

We will consider the case when the algorithm outputs pass. Consider the initial state. Note that for any clock configuration, the probability of all paths which commence with the clocks being set somewhere within this configuration is equal to the clock configuration probability. Furthermore, for an arbitrary state s and time t and configuration, the probability of all paths which go through this configuration at this time is the probability of the configuration multiplied by the probability of reaching that state at that time.

The probability of reaching state s at time t is the second parameter passed to the procedure *new_state_matrix*[9].

If every valuation in a configuration corresponds to the same automaton transition, and this transition is the final one in a path which models the formula, then we add the clock configuration probability (multiplied by the probability of reaching that state at that time) to *total_pass*.

This is the only way in which the algorithm adds to the variable *total_pass*. Since the algorithm only outputs pass if *total_pass* is greater than the formula probability p, it is clear that the algorithm will only output pass if the automaton models the formula.

If more than one clock in the configuration is in the range $(0, \delta]$ then more than one of the clocks will have reached time 0 in the interval we are considering, and so the clock configuration probability is added to *error* (line 12 of procedure *new_time_matrix*).

A similar argument applies in the case where the algorithm outputs fail.

[9]In fact, it is greater than or equal to this sum, because some routes through the transition system may have passed or failed the formula already, and therefore would be considered no further by the algorithm.

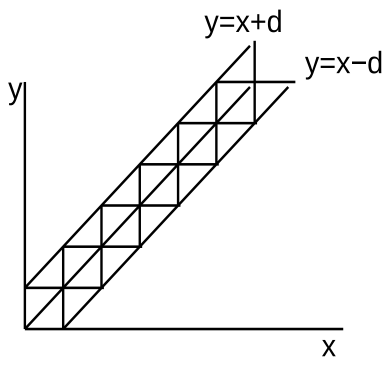

Figure 1: Upper bound on error with clocks x and y.

Therefore the algorithm is sound in the sense that if we are given a definitive answer, this answer is correct. There remains, of course, the question of convergence to the correct answer, and the following theorem summarises the situation.

Theorem 1 For every automaton SA and propositions a_0 and a_1 it is the case that if SA models $[a_0\, \mathcal{U}_{\leqslant t}\, a_1]$ with probability p, then for any error e greater than zero, there is a timestep δ greater than zero such that for the formula $[a_0\, \mathcal{U}_{\leqslant t}\, a_1] > q$, the algorithm will only return undecided if $q \in [p-e, p+e]$.

First note that n independent single variable continuous probability distribution functions $f_1 \ldots f_n$ can always be combined to give a single n variable probability distribution function which is continuous in all dimensions: $f(x_1 \ldots x_n) = f_1(x_1) \times \cdots \times f_n(x_n)$.

For convenience, consider a location with two outgoing transitions and two clocks x and y with distribution functions f_x and f_y. Because f_x and f_y are both continuous, if we set $f(x, y) = f_x(x) \times f_y(y)$ we can (by the note above) say that

$$\forall\, \epsilon > 0.\, \exists\, \delta > 0. f(x, x+\delta) - f(x, x-\delta) < \epsilon$$

We will show that for any desired size of error we can choose a suitably small timestep.

Now, $\int_0^m f(x, x+\delta) - f(x, x-\delta)dx$ [10] (the probability of the clock valuation falling between the two 45 degree lines in Figure 1) is greater than the sum of all contributions to the error variables (represented by the squares in the figure). Since the number of locations in the stochastic automaton is finite (say N_s) and (for bounded until formulas with less than subscripts) the maximum number of visits to any location is finite (say N_v) for any desired error e we must ensure that, for every location, for the multivariate function associated with that location, we choose ϵ such that $\epsilon < \frac{e}{N_s \times N_v}$. If the timestep is set to the smallest δ necessary to ensure that every location provides errors less than $\frac{e}{N_s \times N_v}$, then total error provided by one location (over all time) will be less than $\frac{e}{N_s}$ and the total error provided by all locations will be less than e.

6 Example

In this section we consider a simple automaton and a bounded until formula, and use these to work through the algorithm and illustrate the key points.

In the example automaton in Fig 2, functions F and G are *probability distribution functions*, i.e. $\lim_{x\to\infty} I(x) = 1$ for $x \in \{v, w\}, I \in \{F, G\}$. Functions f and g are the corresponding *probability density functions*, i.e. $f = F'$ and $g = G'$.

The function H, that maps states to propositions, is $H(s_0) = a_0$, $H(s_1) = a_1$, and the formula we are trying to verify is $[a_0 \, \mathcal{U}_{\leqslant 2} a_1] > \frac{1}{2}$. Or, in words, is the probability of reaching state s_1 within 2 time units greater than 0.5?

We now illustrate this algorithm by applying it to the example specified in Section 6.

We begin with δ equal to one[11].

[10] $m = min\{ub(x), ub(y)\}$, where ub is the function which returns the largest value to which a clock can be set.

[11] The type of situation where the algorithm would do very badly is if one clock has a very small lower bound and all the rest have a very high lower bound. This is accenuated if the first clock is hardly used. It might even be that the state where the first clock is used is unreachable or has a very low probability of being reached. Thus a criterion for the algorithm to work efficiently is that all pdf lower bounds are similar.

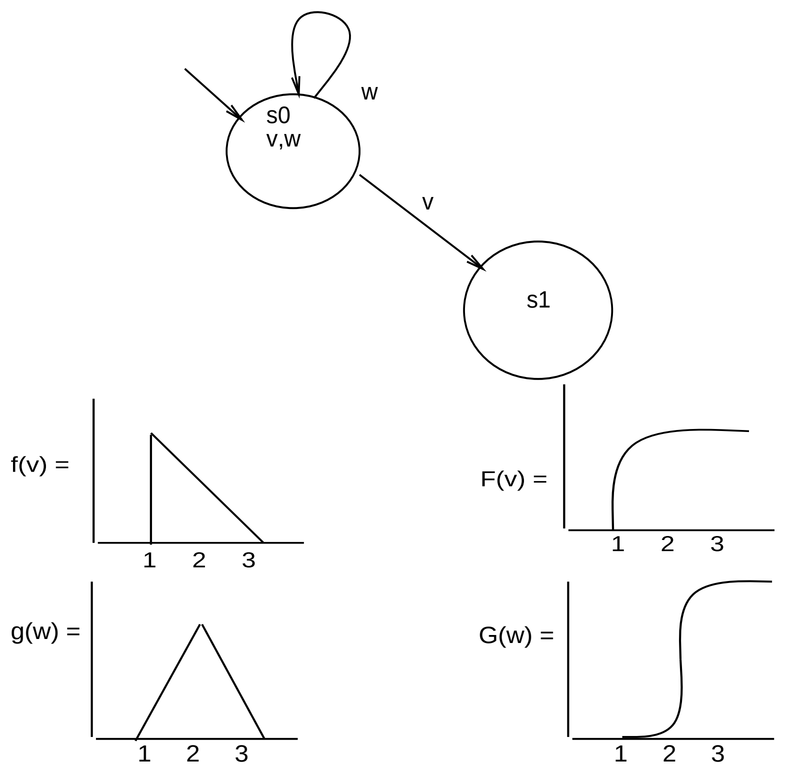

Figure 2: Example automaton and probability functions.

The algorithm

We illustrate the working of the algorithm by working through key sections of the algorithm. Sections A, B and C below correspond to the sections A,B and C in the algorithm description in Section A. Within Section C, line numbers correspond to the line numbers of the algorithm.

Section A

This section initialises all the variables to zero, and calculates all the probabilities of clocks falling in the ranges $(0, \delta], (\delta, 2\delta]$ etc. from the probability distribution functions entered as part of the stochastic automaton.

In our example, the probabilities that the clocks v and w are in the ranges $(0, \delta], (\delta, 2\delta]$ or $(2\delta, 3\delta]$ are given by

	v	w
$(0, \delta]$	0	0
$(\delta, 2\delta]$	$\frac{3}{4}$	$\frac{1}{2}$
$(2\delta, 3\delta]$	$\frac{1}{4}$	$\frac{1}{2}$

These are easy to obtain from the clock probability distribution functions.

Section B

The initial state s_0 does not model a_1, but it does model the proposition a_0, and so the procedure *init_matrix* is called. This returns $matrix(s_0, 0)$ which is as follows

$$\begin{array}{c|cccc} w & & & & \\ 3 & 0 & \frac{3}{8} & \frac{1}{8} & \\ 2 & 0 & \frac{3}{8} & \frac{1}{8} & \\ 1 & 0 & 0 & 0 & \\ \hline & 1 & 2 & 3 & v \end{array}$$

and is easily derivable from the probabilities above. The procedure also sets $live(0)$ to $\{s_0\}$.

If $N(v)$ is the upper bound of v, and $N(w)$ is the upper bound of w, there will be $\lceil N(v) \times \frac{1}{\delta} \rceil$ entries on the v axis, and $\lceil N(w) \times \frac{1}{\delta} \rceil$ entries on the w axis, so in this case we get a 3×3 matrix.

This matrix tells us e.g. that when the clocks in the initial state are first set, the probability of clock v being set within the range $(1, 2]$ and clock w being set within the range $(2, 3]$ is $\frac{3}{8}$. That is, for the clock configuration $\langle(1, 2], (2, 3]\rangle$, the clock configuration probability is $\frac{3}{8}$.

Section C

We now enter the iterative part of the algorithm, where each iteration corresponds to increasing the time by one time unit, and the snapshot produced at the end of iteration n corresponds to a view of the automaton at time $n\delta$. The three global probability values[12] are all still zero (lines 1-1a), so ct (current time) becomes δ. Only the state s_0 is live at time zero, so *new_time_matrix* is called (line 6) for $matrix(s_0, 1\delta)$. This returns a number of parameters: $matrix(s_0, 1\delta)$, $new\_states(s1, \delta), prob$ and $error$.

The procedure *new_time_matrix*.

This procedure will return the $matrix(s_0, 1\delta)$ as

$$\begin{array}{c|cccc}
w & & & & \\
3 & 0 & 0 & 0 & \\
2 & \frac{3}{8} & \frac{1}{8} & 0 & \\
1 & \frac{3}{8} & \frac{1}{8} & 0 & \\
\hline
 & 1 & 2 & 3 & v
\end{array}$$

where each clock has advanced one time unit from $matrix(s_0, 0)$. So, at time 1, the probability of clock v being within the range $(0, 1]$ and clock w being within the range $(1, 2]$ is $\frac{3}{8}$.

The probability of staying in state s_0 for at least one time unit is 1; this follows from the fact that no clock can be set to less than δ (1 time unit). Thus $prob(s_0, s_0, 1\delta) = 1$.

None of the edge values (those with at least one clock in the range $(0, 1]$) of the previous time matrix ($matrix(s_0, 0)$) is non-zero (so there is no possibility of any clock reaching zero and causing a transition to fire). The second half of the procedure (lines 10-23, which would determine the new states reached from state s_0) is therefore not executed and the global probability values

[12] These are the probability values that are updated throughout the algorithm: *total_pass*, *total_fail* and *error*.

(*total_pass*, *total_fail* and *error*) are all still zero. *new_states*(s_0, δ) will be returned as $\{\}$, since no new states can be reached at time δ.

The next step (lines 7-11 of section C) is to calculate the live states at time δ, and since $prob(s_0, s_0, 1\delta) = 1$ we include s_0.

Since there are no states which can be reached from state s_0 in the time interval $(0, \delta]$, lines 12-22 of section C are not executed.

All of the global probability values are still zero, (i.e. we don't have enough information to decide the truth or falsity of the formula at this stage, lines 1-1a of Section C), and $2\delta \leqslant 2$ (we have more time in which to gain more information, lines 2-3 of Section C), so we begin a second iteration.

On the second iteration of the while loop, *ct* is set to 2δ. Only s_0 was live at the last iteration ($live(\delta) = \{s_0\}$), so at line 6 we call *new_time_matrix* for $matrix(s_0, 2\delta)$.

The procedure *new_time_matrix*.

This again returns a number of parameters, e.g. $matrix(s_0, 2\delta)$ becomes

$$\begin{array}{c|cccc}
w & & & & \\
3 & 0 & 0 & 0 & \\
2 & 0 & 0 & 0 & \\
1 & \frac{1}{8} & 0 & 0 & \\
\hline
 & 1 & 2 & 3 & v
\end{array}$$

where the entry $matrix(s_0, 2\delta)(1, 1)$ is taken from the clock configuration $(\delta, 2\delta], (\delta, 2\delta]$ in the previous time matrix $matrix(s_0, \delta)$ and thus the probability of staying in state s_0 in the interval $(\delta, 2\delta]$ is $\frac{1}{8}$. This is not the final version of $matrix(s_0, 2\delta)$, because some of the clock configurations lead to transitions which lead back to state s_0.

All the other clock configurations ($(1, 1)$, $(1, 2)$ and $(2, 1)$) in $matrix(s_0, \delta)$ lead to transitions. Lines 10-22 of procedure *new_time_matrix* are executed for each of these three configurations.

For clock configuration $(1, 1)$, clock v is (arbitrarily) chosen to fire, leading to state s_1. Line 13a adds state s_1 to *new_states*$(s_0, 2\delta)$, and $prob(s_0, s_1, 2\delta)$ becomes $\frac{3}{8}$. Clock configuration $(1, 1)$ is one where some error may be introduced into the algorithm result. Choosing clock v meant that we go to a state where the formula *TL* becomes true, but choosing the other clock may not lead to such a state. We therefore allow for the possible error introduced here by adding the clock configuration probability to *error*, which becomes $\frac{3}{8}$.

For clock configuration $(1, 2)$ in $matrix(s_0, \delta)$ clock v will fire during the time interval $(\delta, 2\delta]$, again leading to state s_1. The probability of moving from s_0 to s_1 in this interval is now $\frac{6}{8}$. In this configuration only one of the clocks (clock v) could fire, and so no changes are made to *error*.(lines 11a-22).

For clock configuration $(2, 1)$ in $matrix(s_0, \delta)$ clock w will fire during the time interval $(\delta, 2\delta]$, this time leading back to state s_0, so state s_0 is included in $new\_states(s_0, 2\delta)$ and $prob(s_0, s_0, 2\delta)$ is set to $\frac{1}{8}$. Again, no changes are made to *error*.

Now, the *new_time_matrix* procedure is finished, and lines 7-11 of Section C determine the value of $live(2\delta)$ which is $\{s_0, s_1\}$, because at time 2δ the automaton may be in either state.

Lines 12-22 of section C consider each new state that can be reached in time interval $(\delta, 2\delta]$. State s_0 still allows the temporal logic formula to be true, and so procedure *new_state_matrix* is called (line 17), and this updates $matrix(s_0, 2\delta)$ to

$$\begin{array}{c|cccc}
w & & & & \\
3 & 0 & \frac{3}{64} & \frac{1}{64} & \\
2 & 0 & \frac{3}{64} & \frac{1}{64} & \\
1 & \frac{1}{8} & 0 & 0 & \\
\hline
 & 1 & 2 & 3 & v
\end{array}$$

Since there is only a $\frac{1}{8}$ probability of returning to state s_0 at this time each of the added clock configuration probabilities is divided by 8.

State s_1 makes the temporal logic formula true (line 13) and so *total_pass* is increased to $\frac{6}{8}$.

In the next iteration, ct becomes 3δ, which is greater than 2 (line 3). This means that we have no more time left, and so all states undecided after this time are simply false. *total_fail* becomes $\frac{2}{8}$.

With the iterative part of the algorithm over, we have that $total\_pass = \frac{6}{8}$, $total\_fail = \frac{2}{8}$ and $error = \frac{3}{8}$.

The iterations stopped because the value of time became too large — not because the global probabilities contained enough information to make a decision. This means that *total_pass* $(\frac{6}{8})$ is a maximum possible probability value of the formula $[a_0 \, \mathcal{U}_{\leqslant 2} \, a_1]$ (with any clock ordering) and $total\_pass - error$ $(\frac{3}{8})$ is a minimum possible probability value.

Thus, since we wish to determine whether the actual probability value is greater than $\frac{1}{2}$, the algorithm will output undecided.

Decreased timestep: $\delta = \frac{1}{2}$

We now run through the same example, but take snapshots of the automaton every $\frac{1}{2}$ time unit, i.e. $\delta = \frac{1}{2}$.

First, the clock configuration probabilities for the clocks must be recalculated:

	v	w
$(0, \delta]$	0	0
$(\delta, 2\delta]$	0	0
$(2\delta, 3\delta]$	$\frac{7}{16}$	$\frac{1}{8}$
$(3\delta, 4\delta]$	$\frac{5}{16}$	$\frac{3}{8}$
$(4\delta, 5\delta]$	$\frac{3}{16}$	$\frac{3}{8}$
$(5\delta, 6\delta]$	$\frac{1}{16}$	$\frac{1}{8}$

and the initial matrix (multiplied by 128 for convenience) is

w							
6	0	0	7	5	3	1	
5	0	0	21	15	9	3	
4	0	0	21	15	9	3	
3	0	0	7	5	3	1	
2	0	0	0	0	0	0	
1	0	0	0	0	0	0	
	1	2	3	4	5	6	v

The single lines show how the representation of the probabilistic infomation changes with a smaller time step. The top right square, for example, was summarised by the single value $\frac{1}{8}$ when δ was 1.

Section C

After one iteration of the while loop, $matrix(s_0, \delta)$ is

w							
6	0	0	0	0	0	0	
5	0	7	5	3	1	0	
4	0	21	15	9	3	0	
3	0	21	15	9	3	0	
2	0	7	5	3	1	0	
1	0	0	0	0	0	0	
	1	2	3	4	5	6	v

and after the second iteration, $matrix(s_0, 2\delta)$ is

w							
6	0	0	0	0	0	0	
5	0	0	0	0	0	0	
4	7	5	3	1	0	0	
3	21	15	9	3	0	0	
2	21	15	9	3	0	0	
1	7	5	3	1	0	0	
	1	2	3	4	5	6	v

On the third iteration, with $ct = 3\delta = \frac{3}{2}$, procedure *new_time_matrix* returns $matrix(s_0, 3\delta)$ as

w							
6	0	0	0	0	0	0	
5	0	0	0	0	0	0	
4	0	0	0	0	0	0	
3	5	3	1	0	0	0	
2	15	9	3	0	0	0	
1	15	9	3	0	0	0	
	1	2	3	4	5	6	v

This is not the final version of $matrix(s_0, 3\delta)$, because some of the transitions from the state s_0 at this time lead back to state s_0. In particular, clock configurations $(2, 1), (3, 1)$ and $(4, 1)$ of $matrix(s_0, 2\delta)$ will lead back to state s_0, and $prob(s_0, s_0, 3)$ will be $\frac{9}{128}$. $new\_state(s_0, 3\delta) := \{s_0\}$.

The clock configuration $(1, 1)$ of $matrix(s_0, 2\delta)$ leads to state s_1, because the clock v is preferred, so $new\_state(s_0, 3\delta) := \{s_0, s_1\}$, but because we have to choose between clocks we must also add the clock configuration probability ($\frac{7}{128}$) to the variable *error*, which becomes $\frac{7}{128}$.

$live(3\delta) := \{s_0, s_1\}$ (line 10 of section C).

The clock configurations $(1,2)$, $(1,3)$ and $(1,4)$ also lead to state s_1, so $prob(s_0, s_1, 3\delta) = \frac{56}{128}$.

lines 12-23 of section C determine the rest of $matrix(s_0, 3\delta)$. s_0 models a_0 but not a_1, so procedure *new_state_matrix* is called with probability parameter $\frac{9}{128}$, and $matrix(s_0, 3\delta)$ becomes (multiplied this time by $128^2 = 16384$)

w							
6	0	0	189	45	27	9	
5	0	0	189	135	81	27	
4	0	0	189	135	81	27	
3	640	384	191	45	27	9	
2	1920	1152	384	0	0	0	
1	1920	1152	384	0	0	0	
	1	2	3	4	5	6	v

State s_1 models a_1, so *total_pass* becomes $\frac{56}{128}$.

The forth iteration will initially produce the matrix $matrix(s_0, 4\delta)$

w							
6	0	0	0	0	0	0	
5	0	189	45	27	9	0	
4	0	189	135	81	27	0	
3	0	189	135	81	27	0	
2	384	191	45	27	9	0	
1	1152	384	0	0	0	0	
	1	2	3	4	5	6	v

$prob(s_0, s_1, 4\delta)$ will be equal to the sum of the clock configuration probabilities $(1,1)$, $(1,2)$ and $(1,3)$ of $matrix(s_0, 3\delta)$, which is $\frac{4480}{16384} \approx 0.2734$. $total\_pass := 0.4375 + 0.2734 = 0.7109$. The addition to *error* is $\frac{1920}{16384}$, so $error \approx 0.1719$.

Therefore, the probability of entering state s_1 at or before time 4δ is at least $total\_pass - error = 0.5390 > 0.5$.

The forth iteration will continue by building $matrix(s_1, 4\delta)$ and altering $matrix(s_0, 4\delta)$, but they won't be used, because (line 1 of section C), $total\_pass - error = 0.5390 > 0.5$, so lines 28-34 of section C will output pass.

7 Conclusions

In this report we have presented a model checking algorithm for stochastic systems. The principal novel feature in this algorithm is that the system description language (stochastic automata) can handle more than just exponential probability functions. Although we require the probability functions to be continuous, we believe this to be a worthwhile advance. Further work on this subject will include relaxing the restrictions imposed on the stochastic automata, particularly the ability to set and use clocks anywhere in the automaton. Being able to do this would allow parallel composition, and compositional model checking would be a very worthwhile goal indeed.

It would also be good to increase the expressiveness of the logic, allowing nested untils or "greater than" queries, and to extend the model checking algorithm itself to allow queries such as "what is the probability of $[a_0\, \mathcal{U}_{<t}]$ and receive a probability for an answer.

A The algorithm

In this section we present a detailed description of the algorithm. It is divided into Section A (which initialises variables), Section B (the initial part of the algorithm) and Section C (the iterative part). Procedures used are described at the end.

The lines of code are prefaced with numbers, and the comments are delimited with double stars.

```
  ** Section A**
$Model\_check(SA, Formula, \delta, pick)$
  ** note that the function $pick$ is the adversary, used in procedure $new\_time\_matrix$.**
  ** We are assuming a TL formula of the form $[a_0\, \mathcal{U}_{\leqslant t} a_1] \geqslant p$. **
  ** The $\geqslant p$ could easily be changed; the $\leqslant t$ is hardwired into the algorithm. **
  ** **
  ** We begin by initialising variables.**
  ** $ct$: (integer) current_time**
  $ct := 0$
  ** $total\_pass$ and $total\_fail$ are reals in $[0, 1]$. **
  ** At any point in the algorithm, $total\_pass$ is the accumulated **
  ** probability of all the passed paths and $total\_fail$ is the accumlated **
  ** probability of all the failed paths. We initialise them both to zero.**
```

```
$total\_pass := 0$
$total\_fail := 0$
** $error$ is a real in $[0, 1]$. It is the accumulated probability of all paths **
** which, because of the discretisation of the algorithm, we cannot determine exactly.**
** This is where the revised version of the algorithm differs from the initial one.**
** It is initialised to zero. **
** **
$error := 0$
** $prob(s, t)$ is the probability of moving (from anywhere) to location $s$ **
** at time $t$. (i.e. in interval $(t - \delta, t]$.)**
** For all combinations of locations and times, we initialise $prob$ **
** to zero. **
$\forall\, s \in S. \forall\, i \leqslant n.$
  $prob(s, \delta i) := 0$
** $remain(s, t)$ is a boolean which is true if the probability of remaining **
** in location $s$ during time interval $(t - \delta, t]$ is non-zero, false otherwise.**
** They are all initialised to false.**
$\forall\, s \in S. \forall\, i \leqslant n.$
  $remain(s, \delta i) := false$
** $live(t)$ is the set of locations "active" at the end of **
** interval $(t - \delta, t]$, which **
** we need for calculating the information for the next time interval. **
** For all time values, we initialise $live$ to the emptyset. **
$\forall\, i \leqslant n.$
  $live(\delta i) := \emptyset$
** We initialise all values in all matrices to zero.**
** The are $n_s$ clocks in location $s$.**
$\forall\, s \in S.$
  $\forall\, 0 \leqslant j \leqslant n.$
    $\forall\, 1 \leqslant i_1 \leqslant N_1$
        $\vdots$
    $\forall\, 1 \leqslant i_{n_s} \leqslant N_{n_s}.matrix(s, \delta j)[i_1 \ldots i_{n_s}] := 0$

** call procedure for calculating probabilities of clocks falling in the ranges **
** $(0, \delta], (\delta, 2\delta]$ etc. This comes directly from the clock PDFs, **
** and is only calculated once. It is needed for determining the clock**
**probabilities. **
```

```
**$C$ is the set of all clocks and $F$ is the set of clock probability functions**
** This procedure returns $pr$, which is needed in $new\_state\_matrix$ **
** and $init\_matrix$. **
$clock\_config\_probs(C, F, \delta, pr)$
** **

** Section B**
** Consider initial location of SA: $s\_0$ **
** If $s\_0 \models a\_1$ then formula is trivially true. **
if $s\_0 \models a_1$ then
   $total\_pass := 1$
** If $s\_0 \models a\_0$ then formula is undecided and we must **
** unfold SA further. **
elseif s_0 $\models a_0$ then
   ** Build the initial matrix, i.e. $matrix(s\_0, 0)$. **
   **This will then contain the probabilities **
   **of all the different clock settings for location $s\_0$ at time zero. **
   $init\_matrix(matrix(s\_0, 0))$
   ** The only location "live" at time zero will be $s\_0$. **
   $live(0) := \{s\_0\}$
** If $s\_0$ does not model $a\_0$ or $a\_1$ then formula is trivially false. **
else
   $total\_fail := 1$
end if

** Section C**
** Each iteration of the following loop unfolds the automaton by **
** one time step of $\delta$. States which cause the formula to **
** pass/fail are pruned from the tree, and their probabilities added to **
** $total\_pass/total\_fail$, while the undecided states are recorded **
** for the next iteration. **
** We continue while the values of $total\_pass$, $total\_fail$ and $error$ **
** are not enough to determine whether the formula is true or false **
1:   repeat
     ** Increment current_time **
2:   $ct := ct + \delta$
        ** for all states $s$ that were live at the last clock tick **
4:      $\forall\, s \in live(ct - \delta)$
```

```
            ** set current_state to $s$. **
5:          $cs := s$
            ** The procedure $new\_time\_matrix$ returns **
            ** $matrix(cs, ct)$: the matrix for the current state at the current time. **
            ** It also **
            ** updates the function $prob$ with the probability of remaining **
            ** in the current state at the current time and the probabilities of **
            ** moving to different states at the current time. **
            ** It also updates the value of $error$. **
6:          $new\_time\_matrix(matrix(cs, ct), new\_states(cs, ct), remain(cs, ct), prob, error)$
            ** If the probability of remaining in current state at current time is zero **
7:          if $remain(cs, ct) = false$ then
               ** current state is not live at current time and **
               ** only the states which can be reached from current state at current time **
               ** are added to those live at current time **
8:             $live(ct) := live(ct) \cup new\_states(cs, ct)$
9:          else ** $remain(cs, ct) = true$ **
               ** The current state, plus all states which may be reached from it at **
               ** the current time, must be added to the live states. **
10:            $live(ct) := live(ct) \cup \{cs\} \cup new\_states(cs, ct)$
11:         end if
11a:     end forall ** $\forall\, s \in live(ct - \delta)$ **
   ** Now, we have $live(ct)$ and $prob(cs, ct)$ for all $cs$ in $live(ct)$ **
   ** i.e. all the states we could be in at time $ct$, and the probability of **
   ** actually entering them in the previous time interval. **
   ** **
            ** For every state which can be reached at the current **
            ** time, we must see if it causes the formula to pass or fail, in **
            ** which cases we adjust the values for $total\_pass$ or **
            ** $total\_fail$ and remove the state from the $live$ set. If we cannot yet **
            ** tell whether the formula is true or false, we must build the state/time matrix. **
12:         $\forall\, q \in live(ct)$
               ** if $q \models a_1$, then formula is true **
13:            if $q \models a_1$ then
                  ** $total\_pass$ is incremented by the probability of entering $q$ **
                  ** from the current state at the current time **
14:               $total\_pass := total\_pass + prob(q, ct)$
                  ** State $q$ is removed from the live set **
```

```
15:             $live(ct) := live(ct) \setminus \{q\}$
              ** Otherwise, if $q \models a_0$ (and $q$ is not a terminating state) **
              ** then the formula may still be true, **
              ** so we must build $matrix(q, ct)$ and keep state $q$ in the $live(ct)$ set. **
16:           elseif $q \models a_0 \wedge q \notin terminating\_states$ then
                ** The procedure $new\_state\_matrix$ returns **
                ** $matrix(q, ct)$: the matrix for state $q$ at current time, and requires **
                ** $prob(q, ct)$: the probability of entering state $q$ from the current **
                ** state at the current time. **
17:             $new\_state\_matrix(matrix(q, ct), prob(q, ct))$
18:           else ** If $q$ does not model $a\_0$ or it is a terminating state and also **
                  ** it does not model $a\_1$ then the formula is false **
                  ** $total\_fail$ is incremented by the probability of entering $q$ **
                  ** from the current state at the current time **
19:             $total\_fail := total\_fail + prob(q, ct)$
                ** State $q$ is removed from the live set **
20:             $live(ct) := live(ct) \setminus \{q\}$
21:           end if
22:         end forall ** for all states in $live(ct)$ **
23:   until $total\_pass > p$ ** formula has passed **
24:         or
25:         $total\_fail \geqslant 1 - p$ ** formula has failed **
26:         or
27:         $(error \geqslant 1 - p \wedge error \geqslant p)$ ** no possibility of a pass or a fail **
28:         or
29:         ct = t ** time's up.**
30:   if $(ct = t)$ then
        ** All states undecided at the last iteration are now false, so **
        ** $total\_fail$ is set to $1 - total\_pass - error$ **
31:     $total\_fail := 1 - total\_pass - error$
32:   end if
  ****
  ** Output result, based on the values of**
  ** $total\_pass$, $total\_fail$ and $error$ **
33:   if $total\_pass > p$ then
      ** SA models formula **
34:   output pass
35:   elseif ** $total\_fail \geqslant 1 - p$ **
```

```
       ** SA does not model formula **
36:    output fail
37:    else ** errors are too large; cannot decide **
38:    output undecided
39:    end if
```

```
** This procedure builds the initial matrix. **
** We assume there are $n$ clocks associated with this state, **
** and $c_l^{s_0}$ is the $l$th clock. **
** We abbreviate $\lceil upper\_bound(c_l^{s_0}) \rceil . \frac{1}{\delta}$ by $N_l$. **

$procedure\ init\_matrix(matrix(s_0, 0))$
begin procedure
   $\forall\, 1 \leqslant i_1 \leqslant N_1$
        $\vdots$
   $\forall\, 1 \leqslant i_n \leqslant N_n . matrix(s_0, 0)[i_1 \ldots i_n] := \prod_{l=1}^{n} pr(c_l^{s_0} \in [i_l - \delta, i_l))$
end procedure
```

```
$procedure\ new\_time\_matrix(matrix(cs, ct), new\_states(cs, ct), remain(cs, ct), prob, error)$
   ** This procedure updates a matrix by incrementing time, not by **
   ** changing state. We can do this by considering the values in the previous time **
   ** matrix. It also updates the function $prob$,**
   ** and the variable $error$.**
   ** There are $n$ clocks in state $cs$.**
begin procedure
1: $\forall\, 1 \leqslant i_1 \leqslant N_1$
        $\vdots$
2: $\forall\, 1 \leqslant i_n \leqslant N_n .$
                    ** If one of the matrix indices is at its maximum value, then the **
                    ** probability value in this position must be zero. This is **
                    ** because this procedure is always the first to update a state/time matrix. **
                    ** **
                    ** **
3:                  if $\exists\, l \leqslant n \bullet i_l = N_l$ then
```

```
4:              $matrix(cs, ct)[i_1, \dots, i_n] := 0$
                ** otherwise the values in the matrix can be updated simply from the **
                ** values in the previous time matrix. **
5:            else ** all clocks $c_i$ are $\geqslant 1$ and $< N_i$ **
6:              $matrix(cs, ct)[i_1, \dots, i_n] :=$
7:                  $matrix(cs, ct)[i_1, \dots, i_n] + matrix(cs, ct - \delta)[i_1{+}1, \dots, i_n{+}1]$
                ** we record the fact that it is possible to remain in this state **
                ** at this time. **
8:              $remain(cs, ct) := true$
9:            end if
9a:end forall
  ** We now pick out the positions in the previous time matrix which, **
  ** when moved forward one unit in time, result in a new state. **
10:$\forall\, 1 \leqslant i_1 \leqslant N_1$
      $\vdots$
11:$\forall\, 1 \leqslant i_n \leqslant N_n$
  ** If more than one of the previous time matrix indices is one, we know that **
  ** more than one of the clocks will have reached zero by $ct$, and so we **
  ** add the probability to error. **
11a:          if $\#\{c_l \mid c_l = 1\} > 1$ then
12:             $error := error + matrix(cs, ct - \delta)[i_1, \dots, i_n]$
12a:          else if $\#\{c_l \mid c_l = 1\} = 1$
                ** Given the stochastic Automaton $SA$, the state $cs$ and the clock $cc$ **
                ** $s'$ is the resulting state. If the clock is associated with more than **
                ** one transition the function $pick$ (the adversary) chooses the **
                ** resulting state. Otherwise the state is the one determined by the **
                ** transition relation of the SA. **
13:             $s' := pick(SA, cs, c_l)$
13a:            $new\_states(cs, ct) := new\_states(cs, ct) \cup \{s'\}$
                ** the probability of entering $s'$ at time $ct$ **
                ** is incremented by the matrix probability **
14:             $prob(s', ct) := prob(s', ct) + matrix(cs, ct - \delta)[i_1, \dots, i_n]$
22:           end if **line 11**
23:         end forall
24:end procedure
```

** This procedure builds a new matrix, where the state is new rather than the time **

```
** We assume there are $n$ clocks associated with this state, **
** and $c^s_l$ is the $l$th clock. **
** We abbreviate $\lceil upper\_bound(c^s_l)\rceil.\frac{1}{\delta}$ by $N_l$. **
** The values in the matrix are calculated by multiplying the clock **
** probabilities by a factor of $p$, where $p$ is the probability of **
** entering the state, and adding this value to the value already in **
** the position. **

procedure new_state_matrix(matrix(cs, ct), p)
begin procedure
  $\forall\, 1 \leqslant i_1 \leqslant N_1$
      ⋮
  $\forall\, 1 \leqslant i_n \leqslant N_n . matrix(cs, ct)[i_1, \ldots, i_n] :=$
                  $matrix(cs, ct)[i_1, \ldots, i_n] + (p \times \prod_{l=1}^{n} pr(c^s_l \in [i_l - \delta, i_l))\ )$
end procedure
```

B Complexity measues

One obvious measure of the complexity of the algorithm is how δ relates to t, i.e. the value of n in $n\delta = t$ is an upper bound on the number of iterations that can occur when considering a TL formula of the form $[a\,\mathcal{U}_{\leqslant t} b] > p$. For formulae with a $\geqslant t$ subscript this isn't true.

Space complexity is exponential wrt the number of clocks consumed at states. The largest matrix is given by the formula $max\prod\{\lceil\frac{upper\_bound(c)}{\delta} \mid c \in \kappa(s)\}$ where s is a state in the automaton.

References

[BK98] Christel Baier and Marta Kwiatkowska. Model checking for a probabilistic branching time logic with fairness. *Distributed Computing*, May 1998.

[D'A99] Pedro D'Argenio. *Algebras and automata for timed and stochastic systems.* PhD thesis, University of Twente, November 1999.

[DKB98] Pedro R. D'Argenio, Joost-Pieter Katoen, and Ed Brinksma. An algebraic approach to the specification of stochastic systems (extended abstract). In D. Gries and W.-P. de Roever, editors, *Proceedings of the IFIP Working Conference on Programming Concepts and Methods, PROCOMET'98*, pages 126–147. Chapman & Hall, 1998.