Temporal Logic in a Stochastic Environment

B. Strulo* **P.G. Harrison†** **D. Gabbay**
Department of Computing
Imperial College
LONDON SW7 2BZ

Abstract

Temporal logic has proved to be a useful tool for the specification of computer systems interacting with their environment. Meanwhile a common approach to the analysis of the performance of such systems has been based on stochastic process theory and in particular Markov chains. We describe an approach to modelling a temporal logic system in a random environment and how Markov Chain techniques can be applied. We survey some other approaches.

1 Introduction

Temporal logic has been used to good effect in specifying systems interacting with their environment [Pnu86, Hai82, Cha89, AH91]. These systems, often called reactive systems, are typically concurrent programs or real-time control systems whose job is to monitor and react to external events. Temporal logic is good at describing the structure in time required of the actions of the system. Then standard methods of proof or synthesis allow us to show that the system does have those properties. An example is an operating system where a resource is to be allocated to a requester. We might require that after any request eventually the resource is allocated. Temporal logic is a natural formalism for expressing this requirement and the hope is that appropriate proof techniques would allow a demonstration that a given operating system has the required property.

For information about the performance of a real-time system we may wish to describe the environment in a probabilistic manner. Then numerical techniques are available to predict the long-term behaviour of such a stochastic system. Thus suppose that, in our example, we know the probabilities for a transition from a state with no request to a state with one pending and vice versa. Then we may predict the probability of finding the system, in the long term, in either state.

Temporal logic has a large and interesting literature both as a topic in itself and in relation to computer science, [RU71]. It emerged from study in philosophy and linguistics but its importance to computer science is now clear. A wide ranging survey showing this

*Supported by a grant from the SERC
†Supported by the CEC as part of the ESPRIT-BRA QMIPS project 7269

movement and discussing the many different themes and perspectives on this central idea but from a theoretical point of view is provided by [vB89].

Much work relevant to computer science is on the use of temporal logic in specification particularly for reactive systems of the type mentioned already. An introduction to this stream of work is [MP81] which uses a fairly standard linear temporal logic and clearly demonstrates its utility for proving important properties of concurrent programs. A complete example of a complex program and proof is in [Hai82] which clearly and indisputably shows the importance of these ideas.

The present work investigates the particular logic USF and the temporal logic specification/programming style of [Gab89]. This is the heart of a relatively new approach in which the specification is treated as a program and executed directly. The above paper introduces the idea that the past component of a specification can be read as a querying of the past while the future component is an imperative command that the future must meet.

Our use of stochastic process theory is limited to standard results taken from a well established literature. We have used in particular [Mit87] and also [Bar78]. We have used standard definitions of processes as random variables using conventional notions of probability and conditionals. We also use the usual ideas for classifying states by the accessibility relation and finally we use the steady-state theorem to find the steady-state solution.

In this paper we first of all explain what we mean by a temporal logic system. We then show that, for a wide class of such systems, there is a finite-state strategy that they may use to execute. Thus we show that we may regard that system as a finite state automaton. Then we show that such an automaton in an environment modelled by a finite-state Markov chain is itself such a Markov chain. Thus we show that the natural model of a temporal logic system in a stochastic environment is a finite-state Markov chain. Once we know the system is a finite-state Markov chain further analysis becomes possible. For example, we might be able to determine the probability that a specification will be satisfied in a given stochastic environment.

2 Fundamentals

What is a temporal logic system? We consider we have a system made up of an environment and what we will call a program (though it may in fact be some complex combination of programs and hardware). We model this system in a discrete time with a starting point. Thus w.l.o.g. we assume time to be the natural numbers.

To enable us to use a logical formalism we will describe the behaviour of the system in terms of the valuation of proposition letters. Thus we assume the environment and the program to have a finite number of controls each and represent these controls by the state of some sets of proposition letters.

We will use a synchronous model. Thus we assume the environment and then the program take turns to assert their behaviour (by setting and resetting their proposition letters). Then our temporal logic comes in as some supplied specification of desired behaviour. The specification is a temporal formula in the proposition letters and we say a behaviour meets its specification if it is a model of the formula. A program meets the specification if all its possible behaviours do. We formalise this model below.

2.1 The model

We assume our propositions are drawn from some set *Prop*. We will assume this infinitely extensible so none of our constructions exhaust it. Given a set of propositions $P \subset Prop$ we say $h : P \to 2^{\mathbb{N}}$ is a P-structure *i.e.* a function assigning a set of times at which it is true to each proposition. We write the set of P-structures $S(P)$.

A propositional temporal logic L over such structures then assigns to a set of propositions a tuple $\langle L(P), \models \rangle$ where $L(P)$ is the set of well-formed P-formulae and $\models$ a relation from $S(P) \times L(P)$ which, given $h \in S(P)$, gives the true formulae $A \in L(P)$ (we write $h \models A$ in that case).

Much work has gone into comparing the expressiveness of such propositional temporal logics. Typically this is concerned with the number of subsets of the natural numbers that may be "named" with formulae of the logic. This sort of definition distinguishes many different logics. In the context of specification we will use a different idea of expressiveness and show that this makes many propositional temporal logics equivalent.

How do we envisage formulae of our logic to be used as specifications? We imagine our implementor being given a formula $A_1 \in L_1(P)$ for some logic $\langle L_1, \models_1 \rangle$. In this formula some of the propositions represent behaviours of the environment while others represent actions of the program. Our implementor is then obliged to build his program so that any of its behaviours, when considered as a P-structure h say, have the property $h \models_1 A_1$.

Given h_1, a P_1-structure, and h_2, a P_2-structure, and some $P \subset P_1 \cap P_2$ we say

$$h_1 =_P h_2 \quad \text{iff} \quad \forall p \in P \ \ h_1(p) = h_2(p)$$

which is to say that they agree on the propositions in P. Note that the demand we have placed on our implementor is that the P-structures he produces give appropriate interpretations to the propositions in P. Now let us say that our implementor produces a behaviour which assigns additional propositions (from P' say). These have no direct external interpretation and so we may make no restriction on the value of these propositions. Thus the demand on our implementor is that whatever behaviour he produces, say a $(P \cup P')$-structure h_2, then for any P-structure h_1 with $h_1 =_P h_2$ then $h_1 \models_1 A_1$.

So now let us say that our implementor is given some alternative formula A_2 in a different logic $\langle L_2(P \cup P'), \models_2 \rangle$. This will be equally good considered as a specification if it requires him to produce exactly the same behaviours. So we define

Definition 1 *A logic L_2 is* as specificationally expressive *as a logic L_1 iff $\forall P \subset Prop$, $\forall A_1 \in L_1(P)$, $\exists P' \subset Prop$ and $A_2 \in L_2(P \cup P')$ so that $\forall h \in S(P)$*

$$h \models_1 A_1 \quad \textit{iff} \quad \exists h' \in S(P \cup P') \ \ h' \models_2 A_2 \ \textit{and} \ h =_P h'$$

We will take a very simple future time logic we will call T which has only a tomorrow and always operator and show that it is as specificationally expressive as a large and powerful logic - the USF of [Gab89]. This is because by treating formulae as specifications we are giving our specifier additional power - essentially that of existentially quantifying over propositions.

2.2 The logic T

For a set of propositions P, the well-formed formulae $L_2(P)$ are the least set including

$$p \quad \neg A \quad A \wedge B \quad \circ A \quad \widehat{\Box} A$$

where $p \in P$ and A, B are well formed formulae.

We define the semantics first by extending h to a valuation for all formulae. The valuation of a formula A in a P-structure, h, is A^h defined recursively by:

1. If $A \in P$ then $A^h = h(A)$
2. $(\neg A)^h = \mathbb{N} \backslash (A^h)$
3. $(A \wedge B)^h = A^h \cap B^h$
4. $(\circ A)^h = \{n : n + 1 \in A^h\}$
5. $(\widehat{\Box} A)^h = \{n : \forall m \geq n \; m \in p^h\}$

As usual we can define the additional logical connectives $\vee$, $\mathbb{T}$, $\mathbb{F}$, etc. in terms of $\neg$ and $\wedge$. We will also define the additional temporal connective $\widehat{\Diamond} A = \neg \widehat{\Box} \neg A$ meaning that A occurs at sometime in the future. Note that $\widehat{\Box}$ is "reflexive" *i.e.* checks the present as well as the future.

To use these formulae as specifications we have a choice. If the specification is A, does this state that A is true at all times or at time 0? In other words is the requirement on the model h that $A^h = \mathbb{N}$ or that $0 \in A^h$? The choice between these "floating" and "anchored" versions is discussed in [MP89]. For our simple logic we will choose the anchored version and say $h \models_2 A$ iff $0 \in A^h$.

2.3 The logic USF

For a set of propositions P, the well-formed formulae $L_1(P)$ are the least set including

$$p \quad \neg A \quad A \wedge B \quad A \mathcal{U} B \quad A \mathcal{S} B \quad \varphi p A$$

where $p \in P$ and A, B are well formed formulae. We define pure past formulae to be boolean combinations of formulae of the form $A \mathcal{S} B$ where A and B are formulae not including an $\mathcal{U}$. We then add a syntactic restriction that for a fixed point formula $\varphi p A$ to be well formed, A must be pure past.

As before the semantics is defined recursively:

1. If $A \in P$ then $A^h = h(A)$
2. $(\neg A)^h = \mathbb{N} \backslash (A^h)$
3. $(A \wedge B)^h = A^h \cap B^h$
4. $(A \mathcal{S} B)^h = \{n : \exists m < n \; m \in A^h \wedge \forall l (m < l < n \rightarrow l \in B^h)\}$
5. $(A \mathcal{U} B)^h = \{n : \exists m > n \; m \in A^h \wedge \forall l (n < l < m \rightarrow l \in B^h)\}$
6. $(\varphi p A)^h$ is the unique $T \subseteq \mathbb{N}$ such that $T = A^{h(p/T)}$

Here $h(p/T)$ is h with the original valuation of p replaced by T.

Thus the definition of $A\mathcal{S}B$ is the set of all times where B is true at some time in the past and A is true at all times (if any) strictly in between. Similarly the definition of $A\mathcal{U}B$ is all times where B is true in the future and A true at all times (if any) strictly in between.

The definition of φ, as usual for fixed point definitions, requires justification. Rigorous proof is provided in [Hod89] while an explanation is in [Gab89]. To see intuitively why the fixed point φpA exists uniquely note that A is pure past (by the definition of well-formedness). Given h we can consider A as an operator taking any valuation of p, say T, to $A^h(p/T)$. So to construct the fixed point start with p completely unknown. If we apply the A operator to such a valuation we don't know what we will get at *most* time points but $A(p)$ *is* uniquely well defined at time 0 since A is pure past. If we now apply the A operator again and consider $A(A(r))$ the value at 0 will not change. The value at 1 will now also be uniquely well defined however. At each step of reapplying A the values defined so far cannot change but a new defined value is added. Thus the limit of this process (under the obvious metric) is the unique well defined fixed point we require.

As before we can define the additional logical connectives in terms of $\neg$ and $\wedge$. We can also define additional temporal connectives as follows:

Connective	Definition	Meaning
$\Diamond A$	$A\mathcal{U}\mathbb{T}$	A at sometime in the future
$\blacklozenge A$	$A\mathcal{S}\mathbb{T}$	A at sometime in the past
$\Box A$	$\neg\Diamond\neg A$	A always in the future
$\blacksquare A$	$\neg\blacklozenge\neg A$	A always in the past
$\circ A$	$A\mathcal{U}\mathbb{F}$	A "tomorrow" (at the next time)
$\bullet A$	$A\mathcal{S}\mathbb{F}$	A "yesterday" (at the previous time)
$\widehat{\Box} A$	$\Box A \wedge A$	reflexive always
$\widehat{\Diamond} A$	$\neg\widehat{\Box}\neg A$	reflexive sometime

Note that we have defined the usual non-reflexive $\Box$ and $\Diamond$ from USF but also our reflexive operators from logic T. It should be plain that this valuation function is consistent with that defined for our logic T in that it gives the same valuations to formulae that are syntactically in both languages.

Finally we define our semantic relation, this time using the floating version *i.e.* $h \models_1 A$ iff $A^h = \mathbb{N}$. This choice is usual for USF and gives us the opportunity to compare the two.

3 Specificational equivalence

We will show that, as far as specifications go, it doesn't matter which logic we work in. First note that USF is certainly as specificationally expressive as T. Assume we are given some formula A in T. Then plainly A is a formula of USF and given a P-structure h then A^h is the same in both logics. Now consider $\neg\bullet\mathbb{T}$ in USF. This is only true at 0 and so $h \models_1 \neg\bullet\mathbb{T} \rightarrow A$ iff $h \models_2 A$ as required.

Now we show the result in the other direction. We prove first that we can simulate the evaluation process of USF in our simple logic T. We prove that

Theorem 1 *Given $A \in L_1(P)$, a formula in USF, a formula $A^*(q) \in L_2(P \cup P' \cup \{q\})$ where $q \notin P \cup P'$ can be constructed, with*
$\forall h' \in S(P \cup P' \cup \{q\})$

$$h' \models_2 A^*(q) \quad \textit{implies} \quad (\forall h \in S(P) \;\; h =_P h' \textit{ implies } h'(q) = A^h)$$

while

$$\forall h \in S(P) \;\; \exists h' \in S(P \cup P' \cup \{q\}) \;\; h' \models_2 A^*(q) \textit{ and } h =_P h'$$

Thus we construct a new formula $A^*(q)$ from T which includes the necessary specifications to make q the same value as A. Further, it will work whatever P-structure we give it.

Then consider A_2 defined by $A^*(q) \wedge \widehat{\Box} q$. Say $h' \models_2 A_2$. By requiring $A^*(q)$ we obtain $h'(q) = h(A)$ and by requiring $\widehat{\Box} q$ we obtain $h \models_1 A$. Conversely if $h \models_1 A$ then we know we can find $h' \models_2 A^*(q)$ with $h' =_P h$ and $h'(q) = A^h = \mathbb{N}$. Hence $h' \models_2 A_2$ as required and this A_2 is the formula we need to show T as specificationally expressive as USF.

The proof is constructive in that we show an algorithm to construct A^* inductively from A.

Proof
We proceed by induction over the structure of A.

- A is $p \in P$
  Define $A^*(q) = \widehat{\Box}(p \leftrightarrow q)$.

  First say $h' \models_2 A^*(q)$. Then $h'(q) = h'(p)$ and so $h' =_P h$ implies $h'(q) = A^h$ as required.

  Alternatively given h define $h'(q) = h(q)$ and $h' =_P h$. Then $h' \models_2 A^*(q)$ as required.

- A is $\neg B$
  Define $A^*(q) = B^*(\neg q)$.

  First say $h' \models_2 A^*(q)$. Thus $h' \models_2 B^*(\neg q)$ and so by induction $h' =_P h$ implies $h'(\neg q) = B^h$. Thus $h'(q) = A^h$ as required.

  Alternatively given h we know inductively we can find h'' with $h'' =_P h$ and $h'' \models_2 B^*(q)$ so since $q \notin P$ we can define $h' = h''$ but with $h'(q) = \mathbb{N} \setminus h''(q)$ and we have $h' \models_2 A^*(q)$ as required.

- A is $B \wedge C$
  Define $A^*(q) = \widehat{\Box}(q \leftrightarrow r \wedge s) \wedge B^*(r) \wedge C^*(s)$ where r, s are new propositions from P'.

  Now if $h' \models_2 A^*(q)$ then $h' \models_2 B^*(r)$ and $h' \models_2 C^*(s)$ so by induction $h' =_P h$ implies $h'(r) = B^h$ and $h'(s) = C^h$. Thus $h'(q) = A^h$ as required.

  Alternatively given h we know inductively we can find h'_1 with $h'_1 =_P h$ and h'_2 with $h'_2 =_P h$ and $h'_1 \models_2 B^*(r)$ and $h'_2 \models_2 C^*(s)$. Thus by appropriate choice of propositions from P' we can define h' but with $h'(q) = h'_1(r) \cap h'_2(s)$ and we have $h' \models_2 A^*(q)$ as required.

- A is $C\mathcal{S}B$
  Define $A^*(q) = \neg q \wedge \widehat{\Box}(\circ q \leftrightarrow (r \vee (s \wedge q)) \wedge B^*(r) \wedge C^*(s)$.

  Now if $h' \models_2 A^*(q)$ then $h' \models_2 B^*(r)$ and $h' \models_2 C^*(s)$ so by induction $h' =_P h$ implies $h'(r) = B^h$ and $h'(s) = C^h$. Now plainly at 0 we have $\neg q$ as required. Further assume by induction that at n, q correctly has the value of $C\mathcal{S}B$. Then q will be true at $n+1$ iff we have B at n or $C\mathcal{S}B$ at n and C at n giving $h'(q) = A^h$ as required.

  Alternatively given h we know inductively we can find h'_1 with $h'_1 =_P h$ and h'_2 with $h'_2 =_P h$ and $h'_1 \models_2 B^*(r)$ and $h'_2 \models_2 C^*(s)$. Thus by appropriate choice of propositions from P' we can define h' but with
  $h'(q) = \{n : \exists m < n\ m \in h'_2(s) \wedge \forall l(m < l < n \rightarrow l \in h'_1(r))\}$. Then we plainly obtain $h' \models_2 A^*(q)$ as required.

- A is $C\mathcal{U}B$
  Define $A^*(q) = \widehat{\Box}(q \leftrightarrow \circ(r \vee (s \wedge q)) \wedge \widehat{\Box}(q \leftarrow \Diamond r)) \wedge B^*(r) \wedge C^*(s)$.

  Now if $h' \models_2 A^*(q)$ then $h' \models_2 B^*(r)$ and $h' \models_2 C^*(s)$ so by induction $h' =_P h$ implies $h'(r) = B^h$ and $h'(s) = C^h$. Now assume q has the correct value at time n. Then plainly it has the correct value at all times less than n (proof by induction on n). Now consider if r is true at n. Then plainly q is correct at $n-1$ and hence all $m < n$. Now either $h'(r)$ is infinite or finite, if the former then for all m there is such a larger n. Otherwise at some point we have $\neg\Diamond r$ and hence $\neg q$ making q correct for all n as required.

  Alternatively given h we know inductively we can find h'_1 with $h'_1 =_P h$ and h'_2 with $h'_2 =_P h$ and $h'_1 \models_2 B^*(r)$ and $h'_2 \models_2 C^*(s)$. Thus by appropriate choice of propositions from P' we can define h' but with
  $h'(q) = \{n : \exists m > n\ m \in h'_2(s) \wedge \forall l(n < l < m \rightarrow l \in h'_1(r))\}$. Then we plainly obtain $h' \models_2 A^*(q)$ as required.

- A is $\varphi p B(p)$
  Note here that $B(p)$ has a free variable p which it retains after transformation. We will write $B^*(q;p)$ where the second variable is this p and then use $B^*(q;q)$ to indicate that we wish to transform B and then substitute q for the free p in B^*. We define $A^*(q) = B^*(q;q)$.

  Now if $h' \models_2 A^*(q)$ then $h' \models_2 B^*(q;q)$ so by induction $h' =_P h$ implies $h'(q) = B^h$. But $B^h = B^{h'}$ and so by the fixed point semantics $h'(q) = A^h$

  Alternatively given h we know inductively we can find h'' with $h'' =_P h$ and $h'' \models_2 B^*(q;q)$. If we define h' as h'' but with $h'(p) = h'(q)$ then plainly $h' \models_2 A^*(q)$ as required.

This completes the induction and gives the theorem. □

Thus we have shown that in the context of specification we might as well work in our simple T logic as with all the power of USF. The technique used to show this theorem is powerful. It shows that to obtain the required expressiveness as far as specifications are concerned we need only three mechanisms:

1. The ability to assign initial values to propositions. This is available in USF because of the yesterday operators which distinguish the initial time, while in T it is available because of the anchored interpretation of specification.

2. The ability to constrain the valuation of formulae tomorrow by boolean operators given their value today. This single step unfolding is essentially a (non-deterministic) finite-state automaton.

3. The ability to impose some restriction on the eventual satisfaction of any commitments propagated in 2). We achieve this through the $\Box$ or $\widehat{\Box}$ operators.

The relationship to automata (though excluding the past fragment) is discussed clearly in [Wol89].

4 Showing the program to be finite state

This study was motivated by considering the MetateM specification language described in [Gab89]. However we have shown that that logic and a wide variety of other propositional temporal logic specification languages are no more expressive as specifications than our simple logic T. We now wish to show that we can reasonably assume that our program has only a finite number of states.

How does our implementor create the program to meet its specification? In general there may be many answers to this question. We will just show that if our specification is implementable then there is some implementation which is finite state and assume that the program we are dealing with is such an implementation.

We use directly the results from [PR89]. Firstly note that formulae of our logic T are path formulae in their logic. Their operators are reflexive but we have shown how to use these only. Their logic is future only and they use the anchored interpretation exactly as in T. Thus it is straightforward to take our formula A_2 as their ϕ - the linear temporal logic formula to be satisfied. Then their theorem 2 implies that, since we assume our specification implementable, there exists a "deterministic transducer" which, given the environment actions, gives the next program action. They show this transducer is finite-state. Thus we have the result we require.

5 The main theorem

Having established a Markov-like property for the program and assuming it for the environment we now pull it together and show that the system as a whole is a Markov chain in some state space.

5.1 The model in Markov formalism

We describe the external state by a set of environment propositions whose values are encoded by a random variable value E_t : the value of the environment propositions at time t.

The actions of the MetateM interpreter are similarly encoded by A_t : the value of the actions at time t.

To model incomplete information we make the interpreter see not the external state but a visible representation of it encoded by V_t : the value of the visible environment at time t.

To model the uncertainty we make the visible environment dependent randomly on the real environment at that time but nothing else:

Assumption 1

$$\begin{aligned}
&\Pr(V_t = i \mid E_t = j) = \\
&\qquad \Pr(V_t = i \mid E_t = j, E_{t-1} = j_{t-1}, \ldots, E_1 = j_1, A_t = k_t, \ldots, A_1 = k_1)
\end{aligned}$$

This assumption is not restrictive since on the one hand the program has access to this visible environment at any point in the history. On the other, if we wish to have the visible environment dependent on more history we can carry that dependence in the real environment (it is only the instantaneous uncertainty that the real environment cannot model).

As stated above we assume a Markov type property for the real environment:

Assumption 2

$$\begin{aligned}
&\Pr(E_t = i \mid E_{t-1} = j, A_{t-1} = j') = \\
&\quad \Pr(E_t = i \mid \quad E_{t-1} = j, E_{t-2} = j_{t-2}, \ldots, E_1 = j_1, \\
&\qquad\qquad\qquad\quad A_{t-1} = j', A_{t-2} = j'_{t-2}, \ldots, A_1 = {j'}_1)
\end{aligned}$$

As already remarked, although the environment need only look at its last state, in this discrete time framework we could transform any chain which required more information about its past into one which did not, by embedding the history into the state space. The only disadvantage here is that then the state space would be countably infinite.

As shown above we can encode all the information the program needs to make its decision into a finite state space - the state space of the deterministic transducer along with the values of V_t and A_{t-1} (though note that the actions in A may need to be augmented by finitely many new ones).

We will assume that the encoding of A_t, E_t and V_t includes that information. Thus we may take it that the strategy gives A_t as a deterministic function of V_t and A_{t-1}. Thus:

Assumption 3

$$\begin{aligned}
&\Pr(A_t = i' \mid V_t = v, A_{t-1} = j') = \\
&\quad \Pr(A_t = i' \mid \quad V_t = v, V_{t-1} = v_{t-1}, \ldots, V_1 = v_1, \\
&\qquad\qquad\qquad\quad A_{t-1} = j', A_{t-2} = {j'}_{t-2}, \ldots, A_1 = {j'}_1, \\
&\qquad\qquad\qquad\quad E_t = i_t, \ldots, E_1 = i_1)
\end{aligned}$$

Thus the possible dependence is diagrammatically:

$$\begin{tikzcd}
E \cdots \arrow[r] & E_{t-1} \arrow[rr] \arrow[d] & & E_t \arrow[d] & \\
V \cdots & V_{t-1} \arrow[dr] & & V_t \arrow[dr] & \\
A \cdots \arrow[rr] & & A_{t-1} \arrow[uur] \arrow[rr] & & A_t
\end{tikzcd}$$

The proof of the Markov property for the complete process with state encoded by E_t and A_t proceeds by expanding over all possible visible environments to eliminate any dependency there. Then we can combine the properties for E and A to obtain the required overall property. Thus we have

Theorem 2 *The stochastic process $\{(E_t, A_t) \mid t \geq 0\}$ has the Markov property*

$$\begin{aligned}&\Pr(E_t = i, A_t = i' \mid (E_{t-1} = j, A_{t-1} = j', \ldots, A_1 = j'_1)\\ &\quad = \Pr(E_t = i, A_t = i' \mid E_{t-1} = j, A_{t-1} = j')\end{aligned}$$

Proof

Let Π denote the history condition $E_{t-1} = j$, $A_{t-1} = j'$, $E_{t-1} = j_{t-1}$, …, $A_1 = {j'}_1$. Writing VA for the set of visible actions.

$$\Pr(E_t = i, A_t = i' \mid \Pi)$$

$$= \Pr(A_t = i' \mid E_t = i, \Pi) \times \Pr(E_t = i \mid \Pi)$$

by expanding the conditional

$$\begin{aligned}= \sum_{v \in \mathrm{VA}} \quad & \Pr \quad (A_t = i' \mid V_t = v, E_t = i, \Pi)\\ \times \quad & \Pr \quad (V_t = v \mid E_t = i, \Pi)\\ \times \quad & \Pr \quad (E_t = i \mid \Pi)\end{aligned}$$

by total probability

$$\begin{aligned}= \sum_{v \in \mathrm{VA}} \quad & \Pr \quad (A_t = i' \mid V_t = v, E_t = i, \Pi)\\ \times \quad & \Pr \quad (V_t = v \mid E_t = i, A_{t-1} = j')\\ \times \quad & \Pr \quad (E_t = i \mid \Pi)\end{aligned}$$

since V is independent of the history in Π

$$\begin{aligned}= \sum_{v \in \mathrm{VA}} \quad & \Pr \quad (A_t = i' \mid V_t = v, E_t = i, A_{t-1} = j')\\ \times \quad & \Pr \quad (V_t = v \mid E_t = i, A_{t-1} = j')\\ \times \quad & \Pr \quad (E_t = i \mid \Pi)\end{aligned}$$

since A is independent of the history in Π

$$= \Pr(A_t = i' \mid E_t = i, A_{t-1} = j') \times \Pr(E_t = i \mid \Pi)$$

by total probability

$$= \Pr(A_t = i' \mid E_t = i, A_{t-1} = j') \times \Pr(E_t = i \mid E_{t-1} = j, A_{t-1} = j')$$

using the Markov property of E_t to remove the dependence on Π.

$$= \Pr(E_t = i, A_t = i' \mid E_{t-1} = j, A_{t-1} = j')$$

$\square$

5.2 Discussion

We have shown that the system behaves as a Markov chain with a specific state space. This consists of the action propositions (possibly augmented by our transformation into T), the state space of the deterministic transducer, and whatever information is needed by the environment. Using [PR89] and assuming a finite state space for the environment we obtain a finite state space overall.

5.3 An example

We use an example from [BFG+90] : the resource manager. We translate to the connectives used in this paper as follows.

The specification describes a process which manages a resource to be allocated to a maximum of one at any one time of two requesters.

The constraints are taken as:

1. If the resource is requested by a process then it must eventually be allocated to that process.
2. If the resource is not requested then it should not be allocated.
3. At any time, the resource should be allocated to at most one process.

Restricting the example to just two processes use r_1 and r_2 to name the occurrence of requests from the two requesters. Then we use a_1 and a_2 to name the corresponding allocations. Note that the r_i are environment propositions (in E_t) while the a_i are actions in A_t. We will not use V_t in this example since we will assume the interpreter has complete information *i.e.* that $V_t = A_t$.

Then the floating specification in USF is:

1. $\bullet r_1 \rightarrow \Diamond a_1$
2. $\bullet r_2 \rightarrow \Diamond a_2$
3. $(\neg r_1)\,\mathcal{Z}(a_1 \wedge \neg r_1) \rightarrow \neg a_1$
4. $(\neg r_2)\,\mathcal{Z}(a_2 \wedge \neg r_2) \rightarrow \neg a_2$
5. $\neg a_1 \vee \neg a_2$

We are using here the weak form of Since (Zince) which is true at time zero. Note that our yesterday operator is strong (unlike the treatment in Barringer) *i.e.* false at time zero.

We exhibit a transformation into logic T. This gives a new specification:

1. $\widehat{\Box}\left(r_1 \rightarrow \circ\widehat{\Diamond} a_1\right)$
2. $\widehat{\Box}\left(r_1 \rightarrow \circ\widehat{\Diamond} a_1\right)$
3. $\widehat{\Box}\,(p_1 \rightarrow \neg a_1)$
4. $\widehat{\Box}\,(p_2 \rightarrow \neg a_2)$

5. $\widehat{\Box}(\neg a_1 \vee \neg a_2)$

6. $p_1 \wedge \widehat{\Box}(\circ p_1 \leftrightarrow (a_1 \wedge \neg r_1) \vee (\neg r_1 \wedge p_1))$

7. $p_2 \wedge \widehat{\Box}(\circ p_2 \leftrightarrow (a_2 \wedge \neg r_2) \vee (\neg r_2 \wedge p_2))$

Here the p_i hold the current values of the relevant $\mathcal{Z}$. We are using a version of the transformation which translates the weak $\mathcal{Z}$ analogously to $\mathcal{S}$ (the proof is similar and straightforward). We have also eliminated unnecessary propositions.

Working from this specification in T to a deterministic transducer using the Pnueli and Rosner approach has a doubly exponential complexity and is far too long to do manually. Instead we will use a more straightforward approach modelled on the strategy adopted by the MetateM interpreter.

We use states that value these propositions and also our commitment to eventually allocate *i.e.* $\Diamond a_i$. Thus there are 256 different states. Some of these are expressly forbidden by the specifications 3,4, and 5.

We use a notation in which we describe a state by the successive values (0 for false and 1 for true) of: $r_1\ r_2\ a_1\ a_2\ \Diamond a_1\ \Diamond a_2\ p_1\ p_2$.

Assuming the simplest strategy for the interpreter of always giving priority to the requester 1 the possible states and transitions are:

00 00 00 $p_1p_2 \rightarrow r_1r_2$ 00 00 p_1p_2	
00 01 01 $p_1 0 \rightarrow r_1r_2$ 00 00 $p_1 1$	
00 10 10 $0p_2 \rightarrow r_1r_2$ 00 00 $1p_2$	Here we have assumed the interpreter makes no allocation since there is no request.
00 10 11 00 $\rightarrow r_1r_2$ 01 01 10	Here the interpreter fulfils the outstanding allocation.
01 00 00 $p_1p_2 \rightarrow r_1r_2$ 01 01 $p_1 0$	
01 01 01 $p_1 0 \rightarrow r_1r_2$ 01 01 $p_1 0$	
01 10 10 $0p_2 \rightarrow r_1r_2$ 01 01 10	
01 10 11 00 $\rightarrow r_1r_2$ 01 01 10	Here the interpreter makes an immediate allocation to 2 since 1 cannot be waiting (it has priority).
10 00 00 $p_1p_2 \rightarrow r_1r_2$ 10 10 $0p_2$	
10 01 01 $p_1 0 \rightarrow r_1r_2$ 10 10 01	
10 10 10 $0p_2 \rightarrow r_1r_2$ 10 10 $0p_2$	
10 10 11 00 $\rightarrow r_1r_2$ 10 11 00	Here the interpreter makes an immediate allocation to 1 since it has priority.
11 00 00 $p_1p_2 \rightarrow r_1r_2$ 10 11 00	
11 01 01 $p_1 0 \rightarrow r_1r_2$ 10 11 00	
11 10 10 $0p_2 \rightarrow r_1r_2$ 10 11 00	
11 10 11 00 $\rightarrow r_1r_2$ 10 11 00	If both are requesting then we allocate to 1 but ensure we propagate a commitment to allocate to 2.

Note that there are no states of the form xx 00 01 xx (outstanding request with no allocation) since any reasonable interpreter strategy would have satisfied them. Further with this strategy no state of the form xx 01 11 xx (allocation to 2 with outstanding request from 1) is possible since the interpreter would have satisfied the first requester by preference.

Now we may perform analysis on this chain. There are 64 probabilities chosen by the environment for the new state of r_i given the previous r_i and a_i. We will assume them constant so that this chain is time-homogeneous and call them a_{ijklmn} where i, j, k, l are the previous values of the r_i and a_i and m, n are the new r_i values.

To start our analysis we will assume the a_{ijklmn} are all non-zero. Then we may analyze the states using the transition table above as follows. We need to find the irreducible part of the chain to find a steady-state solution.

Consider the sequence of states starting with the potential start state 00 00 00 11:

00 00 00 11	$\rightarrow$	01 00 00 11	$\rightarrow$	00 01 01 10	$\rightarrow$
10 00 00 11	$\rightarrow$	00 10 10 01	$\rightarrow$	11 00 00 11	$\rightarrow$
00 10 11 00	$\rightarrow$	01 01 01 10	$\rightarrow$	10 01 01 10	$\rightarrow$
01 10 10 01	$\rightarrow$	11 01 01 10	$\rightarrow$	01 10 11 00	$\rightarrow$
11 01 01 10	$\rightarrow$	10 10 11 00	$\rightarrow$	11 10 11 00	$\rightarrow$
00 10 11 00	$\rightarrow$	00 01 01 10	$\rightarrow$	00 00 00 11	

which takes us back to the first state. The existence of this cycle shows that no smaller set of states can be closed. To show the irreducibility of this set we look at the other states and see if they are accessible. If we find that they can be reached from a possible start state then they are accessible otherwise inaccessible. We write "i" and "j" to indicate 0 or 1 and use (loop) to indicate a path which is not finding any more states and (inac) for states we have already shown to be inaccessible.

ij 00 00 00	$\leftarrow$	can only be reached from 00 00 00 00 which is not a start state. Thus this state is inaccessible.					
ij 00 00 01	$\leftarrow$	00 00 00 01		(loop)			
	$\leftarrow$	00 01 01 00	$\leftarrow$	01 00 00 0j	$\leftarrow$	00 00 00 00	(inac)
					$\leftarrow$	00 00 00 01	(loop)
					$\leftarrow$	00 01 01 00	(loop)
			$\leftarrow$	01 01 01 00	$\leftarrow$	01 00 00 0j	(loop)
					$\leftarrow$	01 01 01 00	(loop)
ij 00 00 10	$\leftarrow$	00 00 00 10		(loop)			
	$\leftarrow$	00 10 10 00	$\leftarrow$	10 00 00 00	$\leftarrow$	00 00 00 00	(inac)
			$\leftarrow$	10 10 10 00	$\leftarrow$	10 00 00 00	(loop)
					$\leftarrow$	10 00 00 10	...
		continued	$\leftarrow$	10 00 00 10	$\leftarrow$	00 00 00 10	(loop)
					$\leftarrow$	00 10 10 00	(loop)
ij 01 01 00		(inac)					
ij 10 10 10		(inac)					

None of these states are starting states and so none are accessible and part of the irreducible chain.

Thus the originally listed cycle is the irreducible set of states and because of the finite state space it must be recurrent non-null.

We could now solve for steady-state probabilities using the steady-state theorem: Writing e.g. $p_{00000011}$ for the probability of state 00000011 in the steady-state we have balance equations such as: (we use $p_{000000ij}$ to indicate summation over possible i,j values)

$$p_{00000011} = a_{000000} \times p_{00000011} + a_{000100} \times p_{00010110} + a_{001000} \times p_{00101001}$$

$$p_{00010110} = a_{001000} \times p_{00101100} + a_{010000} \times p_{0100001i} + a_{010100} \times p_{01010110} + a_{011000} \times (p_{0110100i} + p_{01101100})$$

etc.

Solution of these equations would enable us to find the proportion of time spent in any one state or the mean time to move from one state to another or various other properties of the steady-state behaviour.

We may however make predictions about its behaviour even without solving these:

Note that the system will return infinitely often to 00 00 00 11. This shows that the system will meet its specifications since any commitment $\Diamond a_i$ will be eventually satisfied.

However consider the system with different environmental transition probabilities so that $a_{ijkl0m} = 0$ in other words that the first requester never stops requesting. Then less transitions are possible so our cycle is no longer valid.

The maximal cycle is now

10 10 11 11 $\rightarrow$ 11 10 11 11 $\rightarrow$ 10 10 11 11

which takes us back to the first state. Thus this set of states are all part of the steady-state recurrent solution. As before we can show that all other states are not part of the irreducible chain. Thus with these environment transitions the system no longer successfully meets its commitments with this strategy.

6 Conclusion

We have considered a natural model to use in describing a system executing temporal logic specifications in an environment which is described randomly. We have shown that a USF specification need not use any past formulae (including fixed points) as long as it has some means of initialising its propositions at the time origin. We have seen that if the specification is implementable then it has a finite-state implementation.

This implies a Markov property for the actions of the program in a suitably but finitely augmented state space. We then assumed such a Markov property for the environment and showed that in conjunction with the property for the interpreter this gives us the Markov property for the system with combined state-space.

Although this did enable us to obtain some results for a very simple example it seems that the computation involved in any reasonable size problem is very large. Instead of providing this sort of numerical answer directly we hope these ideas will provide a jumping off point for the examination of new interpreter execution strategies or for the analysis of probabilistic algorithms.

An important value to find for our system is the probability of successfully satisfying a given specification for a given strategy in a given environment. Our approach, unfortunately, does not naturally lead to this sort of answer. If we consider a system in its steady-state it will either be certain to satisfy its specification or certain not to. The interesting behaviour controlling satisfaction is intimately linked with the transient or initial behaviour of the chain before it reaches its steady-state. Markov chain theory does not include easy solutions to this sort of behaviour though some approximation techniques are available. The applicability of these is still to be investigated.

7 Alternative approaches

We have considered a program built without regard to probability. Our implementations have the property that they will succeed in meeting their specification for all possible behaviours of the environment. But in our example we have seen a simple program which is not an implementation in this sense. However for most random environments (any with certain transitions non-zero probability) our simple program succeeds in meeting its specification with probability one.

There has been considerable work on this sort of verification *i.e.* testing if a given program and system will meet a given specification with probability 1 *e.g.* [ACD91, HS83, LS83, HS84]. Essentially they work with logics which reason about probabilistic systems themselves. In [LS83] a logic containing a straightforward logic of linear time has a new modality added to it. This new modality is read "certainly" and indicates that its argument happens with probability 1. Three different axiomatisations are also provided covering general, finite and bounded models. [HS84] gives logics (over branching models) to reason directly about what can or must happen in Markov systems. Thus with these logics we could take our system with known behaviour and then show it was certain to meet its specifications.

In [CY90] a different problem is attacked. It is assumed that some specification is given. The formalism used is ω-regular sets but this is equivalent to Büchi automata and thus our temporal logic (see *e.g.* [Wol89]). The problem tackled is to find an implementation (program) which maximises the probability of meeting that specification. They give an algorithm based on Markov Decision Process theory to find such an implementation. Their implementation is finite state and, like the non-probabilistic techniques, doubly-exponential in the size of specification.

References

[ACD91] R. Alur, C. Courcoubetis, and D. Dill. Model-checking for probabilistic real-time systems. In *Proceedings of the Eighteenth Colloquium on Automata Languages and Programming*, volume 510 of *LNCS*, pages 115–126, 1991.

[AH91] Rajeev Alur and Thomas A. Henzinger. Logics and models of real time: A survey. In *Real Time: Theory in Practice*, volume 600 of *LNCS*, pages 74–106, 1991.

[Bar78] M. S. Bartlett. *An Introduction to Stochastic Processes, Third Edition.* Cambridge University Press, 1978.

[BBP89] B. Banieqbal, H. Barringer, and A. Pnueli, editors. *Proceedings of Colloquium on Temporal Logic in specification, Altrincham, 1987*, volume 398 of *LNCS*, Altrincham, 1987, 1989.

[BFG+90] H. Barringer, M. Fisher, D. Gabbay, G. Gough, and R. Owens. MetateM: A framework for programming in temporal logic. In *REX Workshop on Stepwise Refinement of Distributed Systems: Models, Formalisms, Correctness.*, volume 430 of *LNCS*, Mook, Netherlands, June 1989, 1990.

[Cha89] Zhou Chaochen. Specifying communicating systems with temporal logic. In Banieqbal et al. [BBP89].

[CY90] C. Courcobetis and M. Yannakakis. Markov decision processes and regular events. In *Proceedings of the 17th Colloquium on Automata Languages and Programming*, number 443 in LNCS, pages 336–349, 1990.

[dBdRR89] J.W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors. *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, volume 354 of *LNCS*. 1989.

[Gab89] D. Gabbay. Declarative past and imperative future: Executable temporal logic for interactive systems. In Banieqbal et al. [BBP89].

[Hai82] B.T. Hailpern. *Verifying Concurrent Processes Using Temporal Logic*, volume 129 of *LNCS*. Springer-Verlag, 1982.

[Hod89] I. Hodkinson. Elimination of fixed point operators in the temporal logic yf. Technical report, Imperial College, 1989.

[HS83] S. Hart and M. Sharir. Termination of probabilistic concurrent programs. *ACM Transactions on Programming Languages and Systems*, 5(3):356–380, 1983.

[HS84] S. Hart and M. Sharir. Probabilistic temporal logics for finite and bounded models. In *Proceedings of the Sixteenth ACM Symposium on the Theory of Computing*, pages 1–13. ACM, 1984.

[LS83] D. Lehmann and S. Shelah. Reasoning with time and chance. *Information and Control*, 53:165–198, 1983.

[Mit87] I. Mitrani. *Modelling of computer and communication systems*. Cambridge University Press, 1987.

[MP81] Z. Manna and A. Pnueli. Temporal verification of concurrent programs. In R. S. Boyer and J. S. Moore, editors, *The Correctness Problem in Computer Science*. Academic Press, 1981.

[MP89] Z. Manna and A. Pnueli. The anchored version of the temporal framework. In de Bakker et al. [dBdRR89].

[Pnu86] A. Pnueli. Applications of temporal logic to the specification and verification of reactive systems: a survey of current trends. In *Current trends in concurrency*, number 224 in LNCS. 1986.

[PR89] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Conference Record of the Sixteenth Annual ACM Symposium on Principles of Programming Languages*. ACM, 1989.

[RU71] N. Rescher and A. Urquhart. *Temporal Logic*, volume 3 of *LEP*. Springer-Verlag, 1971.

[vB89] J. van Bentham. Time, logic and computation. In de Bakker et al. [dBdRR89].

[Wol89] P. Wolper. On the relation of programs and computations to models of temporal logic. In Banieqbal et al. [BBP89].