A Logic for Reasoning about Time and Reliability *

Hans Hansson and Bengt Jonsson

Swedish Institute of Computer Science, Box 1263, S-164 28 Kista, SWEDEN
E-mail: hansh@sics.se, bengt@sics.se

and

Department of Computer Systems, Uppsala University

SICS Research Report SICS/R90013

December 5, 1994

Abstract

We present a logic for stating properties such as, "after a request for service there is at least a 98% probability that the service will be carried out within 2 seconds". The logic extends the temporal logic CTL by Emerson, Clarke and Sistla with time and probabilities. Formulas are interpreted over discrete time Markov chains. We give algorithms for checking that a given Markov chain satisfies a formula in the logic. The algorithms require a polynomial number of arithmetic operations, in size of both the formula and

*This research report is a revised and extended version of a paper that has appeared under the title "A Framework for Reasoning about Time and Reliability" in the Proceeding of the 10^{th} IEEE Real-time Systems Symposium, Santa Monica CA, December 1989. This work was partially supported by the Swedish Board for Technical Development (STU) as part of Esprit BRA Project SPEC, and by the Swedish Telecommunication Administration.

the Markov chain. A simple example is included to illustrate the algorithms.

Keywords: Branching time temporal logic, Markov chains, Model checking, Real-time, Reliability

1 Introduction

Research on formal methods for specification and verification of computer systems has to a large extent focussed on correctness of computed values and qualitative ordering of events, while ignoring aspects that deal with real-time properties such as bounds on response times. For many systems, such as control systems, timing behavior is an important aspect of the correctness of the system, and the interest for research on these aspects of formal methods seems to be increasing at the moment (see e.g. [Jos88]).

For some systems, it is very important that certain time bounds on their behavior are always met. Examples are flight control systems and many process control systems. Methods for reasoning about such *hard deadlines* can be obtained by adding time to existing methods. One can add time as an explicit (virtual) variable, and use standard verification techniques [PH88, SL87, OW87]. Logics that deal explicitly with time quantities have been designed [BH81, JM86, KVdR83, EMSS89].

For some systems, one is interested in the overall average performance, such as throughput, average response times, etc. Methods for analyzing such properties usually employ Markov analysis. Often the systems are described by different variants of timed or stochastic Petri nets [Mol82, ABC86, Zub85, RP84, HV87b].

In this paper, we shall investigate methods for reasoning about properties such as "after a request for a service, there is at least a 98 percent probability that the service will be carried out within 2 seconds". We call such properties *soft deadlines*. Soft deadlines are interesting in systems in which a bound on the response time is important, but the failure to meet the response time does not result in a disaster, loss of lives, etc. Examples of systems for which soft deadlines are relevant are telephone switching networks and computer networks.

We present a logic for stating soft deadlines. The logic is based on Emerson, Clarke, and Sistla's Computation Tree Logic (CTL) [CES83]. CTL is a modal (temporal) logic for reasoning about qualitative program correctness. Typical properties expressible in CTL are: p will eventually hold on all future execution paths (AFp), q will always hold on all future execution paths (AGq), and r will hold continuously on some future execution path (EGr). Recently, and independently of the work presented here, Emerson, Mok, Sistla, and Srinivasan [EMSS89] have extended CTL to deal with quantitative time. Examples of properties expressible in the extended logic ($RTCTL$) are: p will become true within 50 time

units ($AF^{\leq 50}p$) and q will continuously hold for 20 time units ($AG^{\leq 20}q$). $RTCTL$ is suited for specification and verification of hard deadlines.

In our logic, we have equipped temporal operators with time bounds in the same way as in $RTCTL$, i.e., time is discrete and one *time unit* corresponds to one transition along an execution path. In addition, to enable reasoning about soft deadlines we have replaced path quantifiers with probabilities. Examples of properties expressible in our logic are: with at least 50% probability p will hold within 20 time units ($F^{\leq 20}_{\geq 0.5}\ p$) and, with at least 99% probability q will hold continuously for 20 time units ($G^{\leq 20}_{\geq 0.99}\ q$). We interpret formulas in our logic over structures that are discrete time Markov chains. This relates our work to probabilistic temporal logics (e.g., [HS84, HS84]) and temporal logics with probabilistic models (e.g., [CVW86, CY88]) However, these works only deal with properties that either hold with probability one or with a non-zero probability.

A related research area is the work on Timed (and stochastic) Petri Nets (TPN) [Mol82, ABC86, Zub85, RP84, HV87b]. Much effort in the TPN research goes into generating Markov chains from TPN's, and that work could probably be integrated into our framework. The main difference between the TPN approach and ours is the class of properties that are analyzed for Markov Chains. In the TPN tradition, one analyzes properties such as mean utilization, mean response time, and average throughput.

In Section 2, we define our logic, *Probabilistic real time Computation Tree Logic* (PCTL) and in Section 3 we provide examples of properties that can be expressed in PCTL. In Section 4, we present and discuss algorithms for checking if a given structure is a model of a PCTL-formula. Section 5 presents a verification of a simple communication protocol. In Section 6, we discuss related work. In Section 7, we summarize the results and propose directions for further work. Proofs of some theorems and claims are found in Appendix A. Finally, Appendix B contains details of some extra algorithms.

2 Probabilistic real time CTL

In this section, we define a logic, called *Probabilistic real time Computation Tree Logic* (PCTL), for expressing real-time and probability in systems.

Assume a finite set A of *atomic propositions*. We use a, a_1, etc. to

denote atomic propositions. Formulas in PCTL are built from atomic propositions, propositional logic connectives and operators for expressing time and probabilities. The set of PCTL formulas is divided into *path formulas* and *state formulas*. Their syntax is defined inductively as follows:

- Each atomic proposition is a state formula,
- If f_1 and f_2 are state formulas, then so are $\neg f_1$, $(f_1 \wedge f_2)$, $(f_1 \vee f_2)$, $(f_1 \rightarrow f_2)$,
- If f_1 and f_2 are state formulas and t is a nonnegative integer or ∞, then $(f_1\ U^{\leq t}\ f_2)$ and $(f_1\ \mathcal{U}^{\leq t}\ f_2)$ are path formulas,
- If f is a path formula and p is a real number with $0 \leq p \leq 1$, then $[f]_{\geq p}$ and $[f]_{>p}$ are state formulas.

We shall use f, f_1, etc. to range over PCTL formulas. Intuitively, state formulas represent properties of states and path formulas represent properties of paths (i.e., sequences of states). The propositional connectives $\neg$, $\vee$, $\wedge$ and $\rightarrow$ have their usual meanings. The operator U is the (strong) until operator, and $\mathcal{U}$ is the unless (or weak until) operator. For a given state s, the formulas $[f]_{\geq p}$ and $[f]_{>p}$ express that f holds for a path from s with a probability of at least p and greater than p, respectively.

We shall use $f_1\ U^{\leq t}_{\geq p}\ f_2$ as a shorthand for $\left[f_1\ U^{\leq t}\ f_2\right]_{\geq p}$, and $f_1\ \mathcal{U}^{\leq t}_{\geq p}\ f_2$ as a shorthand for $\left[f_1\ \mathcal{U}^{\leq t}\ f_2\right]_{\geq p}$. Intuitively, $f_1\ U^{\leq t}_{\geq p}\ f_2$ means that there is at least a probability p that both f_2 will become true within t time units and that f_1 will be true from now on until f_2 becomes true. Intuitively, $f_1\ \mathcal{U}^{\leq t}_{\geq p}\ f_2$ means that there is at least a probability p that either f_1 will remain true for at least t time units, or that both f_2 will become true within t time units and that f_1 will be true from now on until f_2 becomes true. We will also use $f_1\ U^{\leq t}_{>p}\ f_2$ and $f_1\ \mathcal{U}^{\leq t}_{>p}\ f_2$, with the analogous meaning.

PCTL formulas are interpreted over structures that are discrete time Markov chains. A specified initial state is associated with the structure. In addition, for each state there is an assignment of truth values to atomic propositions appearing in a given formula. Formally, a *structure* is a quadruple $\langle S, s^i, \mathcal{T}, L\rangle$, where

S is a finite set of *states*, ranged over by s, s_1, etc.,

$s^i \in S$ is an *initial state*,

$\mathcal{T}$ is a *transition probability function*, $\mathcal{T} : S \times S \rightarrow [0,1]$, such that for all s in S we have

$$\sum_{s' \in S} \mathcal{T}(s,s') = 1 \ ,$$

L is a labeling function assigning atomic propositions to states, i.e., $L : S \rightarrow 2^A$.

Intuitively, each transition is considered to require one *time unit*. We will display structures as transition diagrams, where states (circles) are labeled with atomic propositions and transitions with non-zero probability are represented as arrows labeled with their probabilities (e.g., the arrow going from state s_k to state s_l is labelled with $\mathcal{T}(s_k, s_l)$). The initial state (s^i) is indicated with an extra arrow. For example, figure 1 shows a structure with 4 states and 5 transitions with non-zero probability. The state labeled with a_1, a_2 is the initial state.

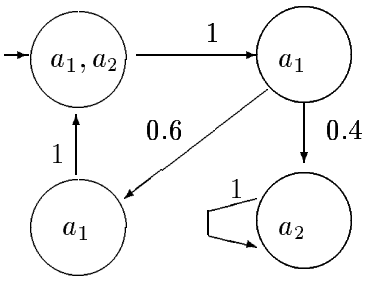

Figure 1: An example of a structure

A *path* σ from a state s_0 in a structure is an infinite sequence

$$s_0 \rightarrow s_1 \rightarrow \ \cdots \rightarrow s_n \ \rightarrow \ \cdots$$

of states with s_0 as the first state. The n:th state (s_n) of σ is denoted $\sigma[n]$, and the prefix of σ of length n is denoted $\sigma{\uparrow}n$, i.e.,

$$\sigma{\uparrow}n \ = \ s_0 \rightarrow s_1 \rightarrow \ \cdots \rightarrow s_n \ .$$

For each structure and state s_0 we define a probability measure μ_m on the set of paths from s_0. Following measure theory [Coh80, KSK76], μ_m

is defined on the probability space $\langle X, \mathcal{A} \rangle$, where X is the set of paths starting in s_0 and $\mathcal{A}$ is a sigma-algebra on X generated by sets

$$\{\sigma \in X : \ \sigma\uparrow n = \ s_0 \rightarrow s_1 \rightarrow \ \cdots \rightarrow s_n\}$$

of paths with a common finite prefix $s_0 \rightarrow s_1 \rightarrow \ \cdots \rightarrow s_n$. The measure μ_m is defined as follows: for each finite sequence $s_0 \rightarrow s_1 \rightarrow \ \cdots \rightarrow s_n$ of states,

$$\mu_m(\{\sigma \in X : \ \sigma\uparrow n = \ s_0 \rightarrow s_1 \rightarrow \ \cdots \rightarrow s_n\}) = \mathcal{T}(s_0, s_1) \times \cdots \times \mathcal{T}(s_{n-1}, s_n) \ ,$$

i.e., the measure of the set of paths σ for which $\sigma\uparrow n = \ s_0 \rightarrow s_1 \rightarrow \ \cdots \rightarrow s_n$ is equal to the product $\mathcal{T}(s_0, s_1) \times \cdots \times \mathcal{T}(s_{n-1}, s_n)$. For $n = 0$ we define $\mu_m(\{\sigma \in X : \ \sigma\uparrow 0 = \ s_0\}) = 1$. This uniquely defines the measure μ_m on all sets of paths in the sigma-algebra $\mathcal{A}$.

We define the truth of PCTL-formulas for a structure K by a satisfaction relation

$$s \models_K f$$

which intuitively means that the state formula f is true at state s in the structure K. In order to define the satisfaction relation for states, it is helpful to use another relation

$$\sigma \models\!\!\models_K f$$

which intuitively means that the path σ satisfies the path formula f in K. The relations $s \models_K f$ and $\sigma \models\!\!\models_K f$ are inductively defined as follows:

$s \models_K a$	iff $a \in L(s)$
$s \models_K \neg f$	iff not $s \models_K f$
$s \models_K f_1 \ \wedge \ f_2$	iff $s \models_K f_1$ and $s \models_K f_2$
$s \models_K f_1 \ \vee \ f_2$	iff $s \models_K f_1$ or $s \models_K f_2$
$s \models_K f_1 \rightarrow f_2$	iff $s \models_K \neg f_1$ or $s \models_K f_2$
$\sigma \models\!\!\models_K f_1 \ U^{\leq t} \ f_2$	iff there exists an $i \leq t$ such that $\sigma[i] \models_K f_2$ and $\forall j : \ 0 \leq j < i : \ (\sigma[j] \models_K f_1)$
$\sigma \models\!\!\models_K f_1 \ \mathcal{U}^{\leq t} \ f_2$	iff $\sigma \models\!\!\models_K f_1 \ U^{\leq t} \ f_2$ or $\forall j : \ 0 \leq j \leq t : \ (\sigma[j] \models_K f_1)$
$s \models_K [f]_{\geq p}$	iff the μ_m-measure of the set of paths σ starting in s for which $\sigma \models\!\!\models_K f$ is at least p.
$s \models_K [f]_{> p}$	iff the μ_m-measure of the set of paths σ starting in s for which $\sigma \models\!\!\models_K f$ is greater than p.

We define

$$\models_K f \quad \equiv \quad s^i \models_K f$$

where s^i is the initial state of K.

3 Properties expressible in PCTL

In this section we present examples of properties that can be expressed in PCTL. First, we discuss some of the facilities of PCTL which makes it suitable for specification of soft and hard deadlines.

The main difference between PCTL and branching time temporal logics such as CTL, is the quantification over paths and the ability to specify quantitative time. CTL allows universal (Af) and existential (Ef) quantification over paths, i.e., one can state that a property should hold for all computations (paths) or that it should hold for some computations (paths). It is not possible to state that a property should hold for a certain portion of the computations, e.g. for at least 50% of the computations. In PCTL, on the other hand, arbitrary probabilities can be assigned to path formulas, thus obtaining a more general quantification over paths. An analogy to universal and existential quantification can in PCTL be defined as:

$$\begin{aligned} Af &\equiv [f]_{\geq 1} \\ Ef &\equiv [f]_{>0} \end{aligned}$$

Quantitative time allows us to specify time-critical properties that relate the occurrence of events of a system in real-time. This is very important for programs that operate in distributed and real-time environments, e.g., communication protocols and industrial control systems. In PCTL it is possible to state that a property will hold continuously during a specific time interval, or that a property will hold sometime during a time interval. Combining this with the above quantification we can define

$$\begin{aligned} G^{\leq t}_{\geq p}\ f &\equiv f\ \mathcal{U}^{\leq t}_{\geq p}\ \mathit{false} \\ F^{\leq t}_{\geq p}\ f &\equiv \mathit{true}\ U^{\leq t}_{\geq p}\ f \end{aligned}$$

Intuitively, $G^{\leq t}_{\geq p}\ f$ means that the formula f holds continuously for t time units with a probability of at least p, and $F^{\leq t}_{\geq p}\ f$ means that the formula f holds within t time units with a probability of at least p.

An important requirement on most real-time and distributed systems is that they should be continuously operating, e.g., every time the controller receives an alarm signal from a sensor the controller should take the appropriate action. We can express such requirements with the following PCTL operators:

$$AGf \equiv f\ \mathcal{U}^{\leq \infty}_{\geq 1}\ \mathit{false}$$

$$AFf \equiv \mathit{true}\ U^{\leq \infty}_{\geq 1}\ f$$

$$EGf \equiv f\ \mathcal{U}_{>0}^{\leq\infty}\ false$$
$$EFf \equiv true\ U_{>0}^{\leq\infty}\ f$$

Intuitively, AGf means that f is always *true* (in all states that can be reached with non-zero probability), AFf means that a state where f is *true* will eventually be reached with probability 1, EGf means that there is a non-zero probability for f to be continuously *true*, and EFf means that there exists a state where f holds which can be reached with non-zero probability. Owicki and Lamport [OL82] have defined a *leads-to* operator ($a \rightsquigarrow b$), with the intuitive meaning that whenever a becomes true, b will eventually hold. We can in PCTL define a quantified leads-to operator as:

$$f_1 \underset{\geq p}{\overset{\leq t}{\rightsquigarrow}} f_2 \equiv AG\left[\left(f_1 \rightarrow F_{\geq p}^{\leq t}\ f_2\right)\right]$$

Intuitively, $f_1 \underset{\geq t}{\overset{\leq p}{\rightsquigarrow}} f_2$ means that whenever f_1 holds there is a probability of at least p that f_2 will hold within t time units. Analogies to many modal operators can be derived from the basic PCTL operators. We can for instance define an operator that corresponds to the CTL [CES83] operator $A[f_1\ U\ f_2]$ as follows:

$$A[f_1\ U\ f_2] \equiv f_1\ U_{\geq 1}^{\leq\infty}\ f_2$$

As an example we will specify a mutual exclusion property. Consider two processes (P_1 and P_2) using the same critical section. The atomic propositions N_i, T_i, and C_i indicates that P_i is in its non-critical, trying, and critical regions, respectively. The mutual exclusion property can be expressed as:

$$AG\left[\neg(C_1\ \wedge\ C_2)\right]$$

This is not sufficient for most real-time systems since the property only states that "bad behavior" must be avoided (safety). To capture the real-time behavior we can specify that whenever P_1 enters its trying region, it will enter its critical region within 4 time units. This can in PCTL be expressed as:

$$T_1 \underset{\geq 1}{\overset{\leq 4}{\rightsquigarrow}} C_1$$

For some systems, it might be sufficient that the deadline is almost always met (e.g. in 99% of the cases). The relaxed property can be expressed as:

$$T_1 \underset{\geq 0.99}{\overset{\leq 4}{\rightsquigarrow}} C_1$$

Relaxing the timing requirement might enable a less costly implementation that still shows acceptable behavior. To be on the safe side we could add a strict upper limit to the relaxed property, combining the hard and soft deadlines above. If we assume that we want P_1 to always enter its critical region within 10 time units, and almost always within 4 time units we get the property:

$$(T_1 \overset{\leq 10}{\underset{\geq 1}{\rightsquigarrow}} C_1) \ \wedge \ (T_1 \overset{\leq 4}{\underset{\geq 0.99}{\rightsquigarrow}} C_1)$$

4 Model Checking in PCTL

In this section, we present a model checking algorithm, which given a structure $K = \langle S, s^i, \mathcal{T}, L \rangle$ and a PCTL formula f determines whether $\models_K f$. The algorithm is based on the algorithm for model checking in CTL [CES83]. It is designed so that when it finishes each state will be labeled with the set of subformulas of f that are *true* in the state. One can then conclude that $\models_K f$ if the initial state (s^i) is labeled with f.

For each state of the structure, the algorithm uses a variable $label(s)$ to indicate the subformulas that are *true* in state s. Initially, each state s is labeled with the atomic propositions that are *true* in s, i.e., $label(s) := L(s)$, $\forall s \in S$. The labeling is then performed starting with the smallest subformula of f that has not yet been labeled, and ending with labeling states with f itself. Composite formulas are labeled based on the labeling of their parts. Assuming that we have performed the labeling of f_1 and f_2, the labeling corresponding to negation ($\neg f_1$) and propositional connectives ($f_1 \ \wedge \ f_2$, $f_1 \ \vee \ f_2$ and $f_1 \rightarrow f_2$) is straightforward, i.e.,

$$\begin{array}{lcll}
label(s) & := & label(s) \cup \{\neg f_1\} & \text{if } f_1 \notin label(s), \\
label(s) & := & label(s) \cup \{f_1 \wedge f_2\} & \text{if } f_1, f_2 \in label(s), \\
label(s) & := & label(s) \cup \{f_1 \vee f_2\} & \text{if } f_1 \in label(s) \text{ or } f_2 \in label(s), \\
label(s) & := & label(s) \cup \{f_1 \rightarrow f_2\} & \text{if } f_1 \notin label(s) \text{ or } f_2 \in label(s),
\end{array}$$

where in addition the new formula must be a subformula of f.

In the sequel, we shall treat the modal operators. Section 4.1 presents two algorithms for labeling states with the modal subformulas of PCTL. In Section 4.2 we discuss labeling in cases with extreme parameter values (e.g. $p = 1$, $p = 0$, and $t = \infty$).

4.1 Labeling states with the modal subformulas of PCTL

We shall give an algorithm for the labeling of states for the formula $f_1\ U^{\leq t}_{\geq p}\ f_2$, assuming that we have done the labeling for formulas f_1 and f_2, and that $t \neq \infty$ (an algorithm for $t = \infty$ will be given in Section 4.2). Let us introduce the function $\mathcal{P}(t,s)$ for $s \in S$ and t an integer. If $t \geq 0$, we define $\mathcal{P}(t,s)$ to be the μ_m-measure for the set of paths σ starting in s for which $\sigma \models_K f_1\ U^{\leq t}\ f_2$. If $t < 0$, we define $\mathcal{P}(t,s) = 0$. In Appendix A we prove that $\mathcal{P}(t,s)$ for $t \geq 0$ satisfies the following recurrence equation:

$$\begin{aligned} \mathcal{P}(t,s) = \quad & \text{if } f_2 \in \mathit{label}(s) \text{ then } 1 \\ & \text{else} \quad \text{if } f_1 \notin \mathit{label}(s) \text{ then } 0 \\ & \qquad \text{else } \sum_{s' \in S} \mathcal{T}(s,s') \times \mathcal{P}(t-1,s') \end{aligned} \tag{1}$$

This recurrence equation gives an algorithm that labels the state s with $f_1\ U^{\leq t}_{\geq p}\ f_2$ if $\mathcal{P}(t,s) \geq p$.

The above recurrence equation can also be formulated in terms of matrix multiplication. Let $s_1, \ldots, s_N$ be the states in S. Partition S into three subsets, S_s, S_f, and S_i, as follows:

- S_s - the *success* states, are states labeled with f_2 (i.e., states for which $f_2 \in \mathit{label}(s)$).
- S_f - the *failure* states, are states which are not labeled with f_1 nor f_2 (i.e., states for which $f_1, f_2 \notin \mathit{label}(s)$).
- S_i - the *inconclusive* states, are states labeled with f_1 but not with f_2 (i.e., states for which $f_1 \in \mathit{label}(s)$ and $f_2 \notin \mathit{label}(s)$).

Define the $N \times N$-matrix M by

$$M[s_k, s_l] \ = \ \begin{cases} \mathcal{T}(s_k, s_l) & \text{if } s_k \in S_i \\ 1 & \text{if } s_k \notin S_i \ \wedge \ k = l \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

Let $\overline{\mathcal{P}}(t)$ be a column vector of size N whose ith element is $\overline{\mathcal{P}}(t)[s_i]$. Define $\overline{\mathcal{P}}(0)[s_i]$ to be 1 if $f_2 \in \mathit{label}(s_i)$, otherwise $\overline{\mathcal{P}}(0)[s_i]$ is 0. Then we have

$$\overline{\mathcal{P}}(t) \ = \ M^t * \overline{\mathcal{P}}(0) \tag{3}$$

for $t > 0$. A possible optimization is to collapse the S_s and S_f states into two representative states: s_s and s_f. This will reduce the size of M to $(|S_i| + 2) \times (|S_i| + 2)$.

In Appendix A we prove that equations (1) and (3) are equivalent.

For formulas of form $f_1\ U^{\leq t}_{>p}\ f_2$ we can use the same calculations as for $f_1\ U^{\leq t}_{\geq p}\ f_2$, but we will only label states for which $\mathcal{P}(t,s) > p$. Model checking for formulas of the form $f_1\ \mathcal{U}^{\leq t}_{\geq p}\ f_2$ and $f_1\ \mathcal{U}^{\leq t}_{>p}\ f_2$ can be done via the dual formulas:

$$f_1\ \mathcal{U}^{\leq t}_{\geq p}\ f_2 \equiv \neg\left[(\neg f_2)\ U^{\leq t}_{>(1-p)}\ (\neg f_1\ \wedge\ \neg f_2)\right]$$

$$f_1\ \mathcal{U}^{\leq t}_{>p}\ f_2 \equiv \neg\left[(\neg f_2)\ U^{\leq t}_{\geq(1-p)}\ (\neg f_1\ \wedge\ \neg f_2)\right]$$

Alternatively, we can define an analogy to $\mathcal{P}(t,s)$ for the Unless case and construct algorithms similar to algorithms 1 and 2 below. This is done in Appendix B.

Calculating $\mathcal{P}(t,s)$

We propose two algorithms for calculating $\mathcal{P}(t,s)$. The first algorithm is more or less directly derived from equation (1) and the second algorithm uses matrix multiplication and the matrix M as in equation (3).

Algorithm 1:

for $i := 0$ to t do
  for all $s \in S$ do
    if $f_2 \in label(s)$ then $\mathcal{P}(i,s) := 1$
    else begin
      $\mathcal{P}(i,s) := 0$;
      if $f_1 \in label(s)$ then for all $s' \in S$ do
        $\mathcal{P}(i,s) := \mathcal{P}(i,s) + \mathcal{T}(s,s') * \mathcal{P}(i-1,s')$
    end

Algorithm 2:

for all $s \in S$ do
  if $f_2 \in label(s)$ then $\overline{\mathcal{P}}(0)[s] := 1$
  else $\overline{\mathcal{P}}(0)[s] := 0$;
$\overline{\mathcal{P}}(t) = M^t * \overline{\mathcal{P}}(0)$

Algorithm 1 requires $\mathcal{O}(t * |S|^2)$ [1] arithmetical operations. Ignoring the zero-probability transitions in $\mathcal{T}$ we can reduce the number of arithmetical operations required to $\mathcal{O}(t * (|S| + |E|))$, where $|E|$ is the number of transitions in $\mathcal{T}$ with non-zero probability. For a fully connected structure these expressions coincide, since $|E| = |S|^2$. The matrix multiplication in Algorithm 2 can be performed with $\mathcal{O}(log(t) * |S|^3)$ arithmetical

[1] The actual worst case complexity can be reduced to $((t+1) * (|S|+1) * 2 * |S|)$, since the outermost loop will be run through $t+1$ times, the "for all" loops will be run through $|S|$ times, there is one assignment statement just before the innermost loop, and there are two arithmetical operations in the innermost assignment statement.

operations, since M^t can be calculated with $\mathcal{O}(log\ t)$ matrix multiplications, each requiring $|S|^3$ arithmetical operations. Let us define the *size* of a modal operator as $log(t)$, where t is the integer time parameter of the operator. The *size* $|f|$ of a PCTL formula f is defined as the number of propositional connectives and modal operators in f plus the sum of the sizes of the modal operators in f. Then the problem whether a structure satisfies a formula f can be decided using at most $\mathcal{O}(t_{max} * (|S| + |E|) * |f|)$ or $\mathcal{O}(|S|^3 * |f|)$ arithmetical operations, depending on the algorithm, where $|S|$ is the number of states, $|E|$ the number of transitions with non-zero probability, t_{max} is the maximum time parameter in a formula, and $|f|$ is the size of the formula. The second expression of complexity is polynomial in the size of the formula and the structure. In Section 5 we illustrate the use of both algorithms in the verification of a simple communication protocol.

4.2 Alternative algorithms for labeling states with modal subformulas

In this section we will discuss alternative algorithms for cases when the modal operator has extreme time (0 or ∞) or probability (1 or 0) parameter values. As in Section 4.1, we will only consider *Until* formulas, since the *Unless* case can be handled via the dual modal operators. To improve performance in an actual implementation, it will probably be desirable to use separate algorithms for the *Unless* case. Such algorithms are defined in Appendix B.

Table 1 gives a classification of possible combinations of p and t parameter values as well as complexities of performing the labeling. The three entries in the left column, corrsponding to $t = 0$ state that the labeling problem then collapses to the problem of labeling states with f_2. The general case in the middle entry has been considered in Subsection 4.1. In the following, we will present alternative algorithms for the remaining cases of the table: $f_1\ U^{\leq t}_{>0}\ f_2$, $f_1\ U^{\leq \infty}_{>0}\ f_2$, $f_1\ U^{\leq \infty}_{\geq p}\ f_2$, $f_1\ U^{\leq t}_{\geq 1}\ f_2$, and $f_1\ U^{\leq \infty}_{\geq 1}\ f_2$.

4.2.1 The case $f_1\ U^{\leq t}_{>0}\ f_2$

To label states with $f_1\ U^{\leq t}_{>0}\ f_2$ we will use the partitioning of states defined in Section 4.1, i.e., S_i, S_s, and S_f. The algorithm will (trivially) label states in S_s. States in S_i will be labeled if there exists a path which is shorter than $t + 1$ from the state to a state in S_s. Inspired by

PROBABILITY \ TIME	0	variable	∞
> 0	$s \models_K f_2$	$EF\left[f_1\ U^{\leq t}\ f_2\right]$ $\mathcal{O}(\|S\|)$	CTL $EF[f_1\ U\ f_2]$ $\mathcal{O}(\|S\|)$
variable	$s \models_K f_2$	The general case $\mathcal{O}(t*(\|S\|+\|E\|))$ or $\mathcal{O}(log\ t*\|S\|^3)$	"probabilistic CTL" $\mathcal{O}(\|S\|^{2.81})$
≥ 1	$s \models_K f_2$	$AF\left[f_1\ U^{\leq t}\ f_2\right]$ $\mathcal{O}(\|S\|)$	CTL AF $\mathcal{O}(\|S\|)$

Table 1: Combinations of p and t parameter values in formulas.

Dijkstra's shortest path algorithm [Gib85] and observing that we only need to consider paths that are shorter than $t+1$ we define the algorithm $LABEL\_EU$ as follows:

LABEL_EU:

unseen := $S_i \cup S_s$;
fringe := S_s;
mr := min($|S_i|, t$);
for i:=0 to mr do
- $\forall s \in$ fringe do addlabel(s,f);
- unseen := unseen - fringe;
- fringe := $\{s : (s \in \text{unseen} \wedge \exists s' \in \text{fringe} : (\mathcal{T}(s,s') > 0))\}$;

Intuitively, *unseen* is the set of states in S_i and S_s that have not yet been considered for labeling, *fringe* are the states that are being labeled, and *addlabel(s,f)* labels state s with the formula f, i.e., $label(s) := label(s) \cup \{f\}$. After passing the for loop with index i, the algorithm will have labeled all states that satisfy $f_1\ U^{\leq i}_{>0}\ f_2$. A proof of correctness for the algorithm is given in Appendix A.

Emerson et.al. [EMSS89] presents a similar algorithm for model checking in RTCTL. The main difference compared with our algorithm is that they do not partition the state set and that they label states with intermediate formulas, i.e., if $E\left[f_1\ U^{\leq k}\ f_2\right]$ holds in state s and the investigated formula is $E\left[f_1\ U^{\leq t}\ f_2\right]$ their algorithm will label s with the $t-k$ formulas $E\left[f_1\ U^{\leq i}\ f_2\right]$ $(k \leq i \leq t)$, whereas our algorithm only labels s with $f_1\ U^{\leq t}_{>0}\ f_2$.

4.2.2 The case $f_1\ U^{\leq\infty}_{>0}\ f_2$

This case can be reduced to the case $f_1\ U^{\leq t}_{>0}\ f_2$ by the following proposition, i.e., the algorithm $LABEL\_EU$ can be used.

Proposition 1 *The formula $f_1\ U^{\leq\infty}_{>0}\ f_2$ holds in a state iff $f_1\ U^{\leq|S_i|}_{>0}\ f_2$ holds in that state.*

Proof:
($\Leftarrow$) We first observe that if a path σ satisfies $f_1\ U^{\leq|S_i|}\ f_2$, then it will also satisfy $f_1\ U^{\leq\infty}\ f_2$. It follows that the measure of the set of paths satisfying $f_1\ U^{\leq\infty}\ f_2$ is at least as large as the measure of the set of paths satisfying $f_1\ U^{\leq|S_i|}\ f_2$.
($\Rightarrow$) If a state s satisfies $f_1\ U^{\leq\infty}_{>0}\ f_2$ then there exists a finite sequence of states starting in s whose last state satisfies f_2, whose remaining states satisfy f_1, and where all transitions have non-zero probability. We can furthermore choose this sequence so that no state is visited twice. The longest such sequence has length $|S_i|$, since it can at most visit all states in S_i followed by a state in S_s. It follows that s must also satisfy $f_1\ U^{\leq|S_i|}_{>0}\ f_2$

4.2.3 The case $f_1\ U^{\leq\infty}_{\geq p}\ f_2$

In this case, the algorithms in Section 4.1 can not be used, since they would require infinite calculations. Instead we define $\mathcal{P}(\infty, s)$ to be the μ_m-measure for the set of paths σ from s for which $\sigma \models_K f_1\ U^{\leq\infty}\ f_2$. In this algorithm we extend the *failure states* to also include states in S_i from which no success state is reachable via transitions with non-zero probability. We define Q to be the new set of *failure states*. The first step of the algorithm is to identify the states in Q. This can be done with an algorithm similar to $LABEL\_EU$. The difference is that we here are interested in the states in S_i that are not labeled by the algorithm, i.e., states from which no state in S_s is reachable. Q contains the union of these states and S_f. Similarly, we can extend the success states to also include states in S_i for which the μ_m-measure for eventually reaching a success state ($s:\ \ s \in S_s$) is 1. We define R to be the new set of success states. The states in R can be identified in a way similar to the identification of states in Q. Algorithms for identifying the states in Q and R are given in Appendix B. The next step is to solve the set of linear equations defined by:

$$\mathcal{P}(\infty, s) = \quad \text{if } s \in R \text{ then } 1 \quad \text{else} \quad \text{if } s \in Q \text{ then } 0 \quad \text{else} \sum_{s' \in S} \mathcal{T}(s, s') \times \mathcal{P}(\infty, s') \tag{4}$$

This can be done with Gaussian elimination, with a complexity of $\mathcal{O}\left[(|S| - |Q| - |R|)^{2.81}\right]$ [AHU74]. In Appendix A we verify that the solution of the system of equations (4) is unique, and that the solution $\mathcal{P}(\infty, s)$ gives the μ_m-measure of the set of paths σ from s for which $\sigma \models_K f_1\ U^{\leq \infty}\ f_2$.

4.2.4 The case $f_1\ U^{\leq t}_{\geq 1}\ f_2$

In this case we must ensure that the gives the μ_m-measure of the set of paths σ from s for which $\sigma \models_K f_1\ U^{\leq t}\ f_2$ is 1. The algorithm $LABEL\_AU$ is defined as follows:

LABEL_AU:

unseen := S_i;
fringe := S_s;
seen := $\emptyset$;
mr := $\min(|S_i|, t)$;
for i:=0 to mr do {
  $\forall s \in$ fringe do addlabel(s,f);
  unseen := unseen - fringe;
  seen := seen $\cup$ fringe;
  fringe := $\{s : [s \in \text{ unseen } \wedge\ \forall s' : (\mathcal{T}(s, s') > 0 : (s' \in \text{seen}))]\}$;

Intuitively, *seen* are the states that have already been labeled. The other variables have analogous intuitive meanings as in algorithm $LABEL\_EU$. After passing the for loop with index i, the algorithm will have labeled all states that satisfy $f_1\ U^{\leq i}_{\geq i}\ f_2$.

4.2.5 The case $f_1\ U^{\leq \infty}_{\geq 1}\ f_2$

Similarly to $f_1\ U^{\leq \infty}_{> 0}\ f_2$, this case can be reduced to $f_1\ U^{\leq t}_{\geq 1}\ f_2$ by the following proposition, i.e., the Algorithm $LABEL\_AU$ can be used.

Proposition 2 *The formula $f_1\ U^{\leq \infty}_{\geq 1}\ f_2$ holds in a state iff $f_1\ U^{\leq |S_i|}_{\geq 1}\ f_2$ hols in that state.*

Proof: The proof follows the same lines as the proof of Proposition 1. □

5 Example

In this section we provide a simple example to illustrate the proposed method. We will verify that a soft deadline is met by a communication protocol. The protocol, Parrow's Protocol (PP) [Par85], is a simplified version of the well known Alternating Bit Protocol [BSW69]. PP provides an error free communication over a medium that might lose messages. For simplicity it is assumed that acknowledgements (ack) are never lost. PP consists of three entities: a sender, a medium, and a receiver. The components and their interactions are described in Figure 2. The structure in Figure 3 presents the behavior of PP. It is assumed that 10% of the messages are lost.

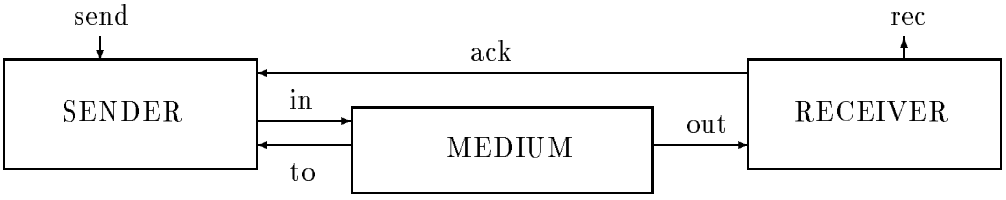

Figure 2: The components of Parrow's Protocol

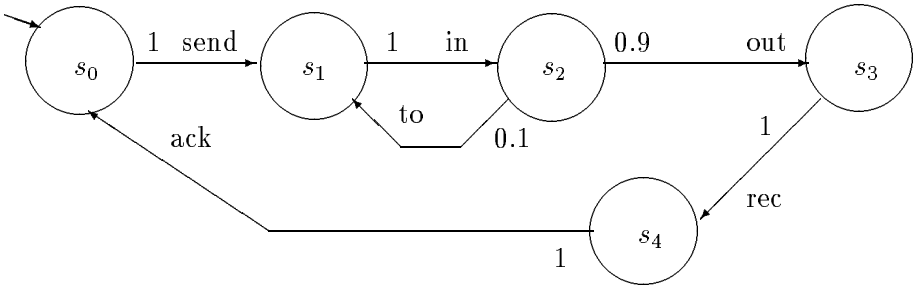

Figure 3: The behavior of PP. The labels on arcs are only added for clarity

PP will be used to illustrate the verification of a *soft deadline*, namely the property that a *rec* (receive) will appear in at least 99% of the cases within 5 time units from the submission of a *send*. In PCTL, this

property can be expressed as:

$$f \;=\; s_0 \underset{\geq 0.99}{\overset{\leq 6}{\rightsquigarrow}} s_4$$

5.1 Verification of PP

We will use the model checking algorithms from Section 4 to verify that PP is a model of f. First, f is formulated in terms of the basic PCTL operators:

$$f \;=\; \underbrace{\underbrace{\left[\underbrace{s_0}_{f_1} \;\rightarrow\; \underbrace{\left(\underbrace{true}_{f_2}\; U^{\leq 6}_{\geq 0.99}\; \underbrace{s_4}_{f_3}\right)}_{f_5}\right]}_{f_6}\; \mathcal{U}^{\leq \infty}_{\geq 1}\; \underbrace{false}_{f_4}}_{f}$$

The labeling of states starts with the smallest subformulas, i.e. f_1, f_2, f_3 and f_4, which is trivial (s_0 will be labeled with f_1, all states will be labeled with f_2, state s_4 will be labeled with f_3, and no state will be labeled with f_4). For labeling of states with f_5 we will use both algorithms for calculation of $\mathcal{P}(t,s)$ presented in Section 4.

Algorithm 1: The labeling of states with f_5 using Algorithm 1 in Section 4.1 is illustrated in Table 2. The table shows the result of the successive calculations, performed from left (time=0) to right (time=6). We can conclude that all states should be labeled with f_5, since after 6 time units $p \geq 0.99$ for all states.

Algorithm 2: When labeling states with f_5 using algorithm 2 we start by deriving the matrix M and the column vector $\overline{\mathcal{P}}(0)$ from the structure.

$$M \;=\; \begin{array}{c} \\ s_0 \\ s_1 \\ s_2 \\ s_3 \\ s_4 \end{array} \begin{array}{c} \begin{array}{ccccc} s_0 & s_1 & s_2 & s_3 & s_4 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0.1 & 0 & 0.9 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \end{array} \qquad \overline{\mathcal{P}}(0) \;=\; \begin{array}{c} \\ s_0 \\ s_1 \\ s_2 \\ s_3 \\ s_4 \end{array} \begin{array}{c} \begin{array}{c} s_0 \end{array} \\ \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \end{array}$$

time: / **state**	0	1	2	3	4	5	6
s_0	0	0	0	0	0.9	0.9	0.99
s_1	0	0	0	0.9	0.9	0.99	0.99
s_2	0	0	0.9	0.9	0.99	0.99	0.999
s_3	0	1	1	1	1	1	1
s_4	1	1	1	1	1	1	1

Table 2: Successive calculations using algorithm 1

The next step is to calculate $\overline{\mathcal{P}}(6)$:

$$\overline{\mathcal{P}}(6) \;=\; M^6 * \overline{\mathcal{P}}(0) \;=\; \begin{pmatrix} 0.99 \\ 0.99 \\ 0.999 \\ 1 \\ 1 \end{pmatrix}$$

We conclude that all states should be labeled with f_5, since $\overline{\mathcal{P}}(6)[s_i] \geq 0.99$ for all states. Not surprisingly, the probabilities in the vector $\overline{\mathcal{P}}(6)$ are exactly the same as the probabilities after 6 time units obtained with Algorithm 1.

Next, we will label all states with f_6, since all states are labeled with f_5. The labeling of states with f can be done via the dual formula:

$$f \;=\; \underbrace{\neg\Big[\underbrace{\underbrace{\neg false}_{f_7}\; U_{>0}^{\leq\infty}\; \underbrace{\Big(\underbrace{\neg f_6}_{f_8} \wedge \underbrace{\neg false}_{f_9}\Big)}_{f_{10}}}_{f_{11}}\Big]}_{f}$$

The labeling of states with f_7, f_8, f_9, and f_{10} is trivial (all states will be labeled with f_7 and f_9, and no state will be labeled with f_8 or f_{10}). For labeling states with f_{11} we use the *label_EU* algorithm. No states will be labeled with f_{11}, Since $S_i = S_s = \emptyset$ and $S_f = S$. Finally, we can label all states with f, since no state is labeled with f_{11}. Note that the labeling

procedure in this last step is very naive. It can be drastically simplified by using a special algorithm for labeling states with formluas of the form AGf'. In this case, such an algorithm is straightforward, since all states should be labeled with AGf' if all states are labeled with f'.

The labeled structure is shown in Figure 4. We can conclude that f holds for the structure, since the initial state (s_0) is labeled with f.

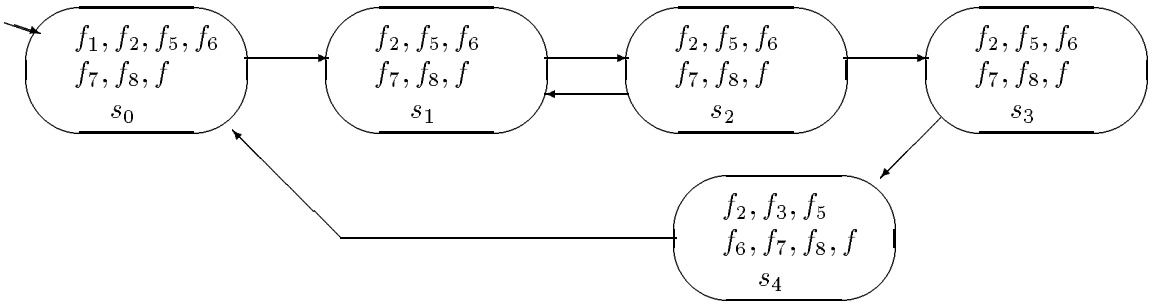

Figure 4: The resulting labeled structure

6 Related Work

6.1 Performance Analysis

One of the most used tools for performance analysis is *Time Petri Nets* (TPNs). There are many variants of the TPN model. TPN are mainly used to calculate exact performance measures of computer system designs. That is, the system is assumed given together wih performance of its parts. The aim is to get a performance measure of the system which is as accurate as possible. Much of the work is therefore to make the model as true as possible to actual systems while retaining the possibility of analysis.

Time Petri nets were introduced by Zuberek [Zub85] and extended by Razouk and Phelps [Raz84, RP84]. The TPN model is based on Petri Nets and associates firing frequencies and deterministic firing times with each transition in the net. The key steps in TPN analysis are:

1. Model the system as a TPN

2. Generate a finite-state Markov chain from the TPN. The generation can take much computational effort, and the state space of the Markov chain can become large. Consequently, much effort is spent finding methods to reduce the computational complexity of this part.

3. Analyze the Markov chain by standard methods to find the long run fraction of time spent in each state. From this information one can make conclusions about utilization of resources such as memory, buses, etc. Waiting times can be analyzed by looking at the fraction of time spent in waiting states.

It follows that the utility of TPN analysis is much in the analysis of designed systems for tuning the behavior of its components. One can make experiments with various values of system parameters to determine optimal configurations. Holliday and Vernon have carried out such analyses for a number of different systems, such as multiprocessor memories [HV87a], cache protocols [VH86]. There are several software packages available that help in the analysis for these models e.g. [HV86, Chi85].

TPN's have a relation to our approach, because our structures are similar to Markov Chains. Much effort in the TPN research goes into generating Markov Chains from TPN's, and that work could probably very well be integrated into our framework. The main difference between the TPN approach and ours is the class of properties that are analyzed for Markov Chains. In TPN tradition, one analyzes mean utilization, and mean waiting times. From these data one can also obtain mean throughput. One does not analyze actual response times in the way we have done. In our approach, we have focussed attention on soft deadlines, as described in the introduction. This kind of analysis can be seen as a complement to the mean-time analysis for TPN's.

6.2 Logics for Real Time

Many of the logics employed to state properties of concurrent programs are various forms of modal logics [Pnu82, Abr80], the most common ones being forms of temporal logic. Many of these are suitable to reasoning about how events or predicates may be ordered in time, without bothering about time quantities. The logic we use is inspired by a simple such logic, CTL [CES83, EC82]. CTL is simple and has a polynomial time model-checking algorithm [CES83] and an exponential time satisfiability algorithm [EC82]. In [ESS89] is described a logic with a polynomial time satisfiability algorithm.

Emerson, Mok, Sistla, and Srinivasan [EMSS89] have extended CTL to deal with quantitative time. As in our logic, they associate one time unit to each transition. A different time model is reported by Alur and Henzinger [AH89]. In their logic, time between successive states is only required not to decrease: it may remain the same, or increase by an arbitrary amount. An early reference to work which has tried to extend modal logics with quantitative time is [BH81], in which traditional linear time temporal logic is extended to cope with quantitative time. Bernstein and Harter present inference rules in the spirit of the proof lattices of Owicki and Lamport [OL82]. No attempt to look at completeness is made. A related logic is presented in [KVdR83], which is richer and includes past-time operators, but there no inference system is presented.

A different approach is the Real-Time Logic (RTL) of Jahanian and Mok [JM86]. RTL is not a modal logic, but a first-order logic. In RTL, one can reason about occurrences of events and the elapsed times between them. The logic is decidable without uninterpreted function symbols, as a special case of Presburger arithmetic. Of course such a decision procedure is highly inefficient. Jahanian and Mok have therefore developed algorithms for checking that a class of finite-state processes satisfy an RTL formula (modelchecking).

6.3 Probabilistic Logics and Logics with Probabilistic models

The above mentioned logics for real-time are not suitable for expressing or reasoning about soft deadlines, since probabilities are not included. On the other hand, there are several examples in the litterature of modal logics that are extended with probabilities (but not time), e.g., PTL by Hart and Sharir [HS84], and TC by Lehman and Shelah [LS82]. However, these works only deal with properties that either hold with probability one or with a non-zero probability.

Probabilistic modal logics have been used in the verification of probabilistic algorithms. Mostly, the objective has been to verify that such algorithms satisfy certain properties with probability 1. The proof methods for these properties resemble the classical proof methods for proving liveness properties under fairness assumptions. There are both non-finite state versions [PZ86], and finite-state modelchecking versions [Var85, Fel83, HSP83, HS84, VW86].

Courcoubetis and Yannakakis [CY88, CY89] have investigated the complexity of modelchecking for linear time propositional temporal logic of

sequential and concurrent probabilistic programs. In the sequential case, the models are (just as our models) Markov chains. They give a modelchecking algorithm that runs in time linear in the program and exponential in the specification, and show that the problem is in PSPACE. Also, they give an algorithm for computing the exact probability that a program satisfies its specification.

7 Conclusions and directions for further work

We have defined a logic, PCTL, that enables us to formulate soft deadline properties, i.e., properties of the form: "after a request for service there is at least a 98% probability that the service will be carried out within 2 seconds".

We interpret formulas in our logic over models that are discrete time Markov chains. Several model checking algorithms, with different suitability for different classes of formulas, have been presented.

The use of Markovian models relates our work to the work on Timed Petri Nets. TPNs could be used as the basis for defining a specification language with our structures as underlying semantic model. Thus, it might be possible to integrate our logic and model checking algorithms into the TPN framework.

The main difference between the TPN approach and ours is the class of properties that are analyzed for Markov chains. In TPN tradition, a steady state solution of the Markov chain is calculated. From this solution mean utilization, mean waiting times, and mean throughput can be obtained. One does not analyze the transient behavior in the way we have done. Our analysis can thus be seen as a complement to the mean-time analysis for TPNs. Also, our logic makes it more convenient to formulate the properties of interest.

So far, we have only considered very simple examples. It would be interesting to examine how well more realistic examples can be handled, both in terms of specification and verification efforts.

Acknowledgements
We are grateful to Ivan Christoff, Linda Christoff, Fredrik Orava, and Parosh for reading and discussing drafts of this manuscript. This work was partially supported by the Swedish Board for Technical Development (ESPRIT/BRA project 3096, SPEC) and the Swedish Telecommunication Administration (project: PROCOM).

References

[ABC86] M. Ajmone Marsan, G. Balbo, and G. Conte. *Performance Models of Multiprocessor Systems*. MIT Press, 1986.

[Abr80] K. Abrahamson. *Decidability and Expressiveness of Logics of Processes*. PhD thesis, Univ. of Washington, 1980.

[AH89] R. Alur and T. Henzinger. A really temporal logic. In *Proc. 30^{th} Annual Symp. Foundations of Computer Science*, 1989.

[AHU74] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley Publishing Company, 1974.

[BH81] A. Bernstein and P.K. Harter. Proving real-time properties of programs with temporal logic. In *Proc. 8th Symp. on Operating System Principles*, pages 1–11, Pacific Grove, California, 1981.

[BSW69] K. Bartlett, R. Scantlebury, and P. Wilkinson. A note on reliable full-duplex transmissions over half duplex lines. *Communications of the ACM*, 2(5):260–261, 1969.

[CES83] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logics specification: A practical approach. In *Proc. 10^{th} ACM Symp. on Principles of Programming Languages*, pages 117–126, 1983.

[Chi85] G. Chiola. A software package for the analysis of generalized stochastic Petri net models. In *Proc. Int. Workshop on Time Petri Nets*, pages 136–143, July 1985.

[Coh80] D. E. Cohn. *Measure Theory*. Birkhauser, 1980.

[CVW86] C. Courcoubetis, M. Vardi, and P. Wolper. Reasoning about fair concurrent programs. In *Proc. 18^{th} ACM Symp. on Theory of Computing*, pages 283–294, 1986.

[CY88] Courcoubetis and Yannakakis. The complexity of probabilistic verification. In *Proc. 29^{th} Annual Symp. Foundations of Computer Science*, pages 338–345, 1988.

[CY89] Courcoubetis and Yannakakis. The complexity of probabilistic verification. Bell labs Murry Hill, 1989.

[EC82] E. Emerson and E. Clarke. Using branching time temporal logic to synthesize synchronization skeletons. *Science of Computer Programming*, 2(3):241–266, 1982.

[EMSS89] A. Emerson, A. Mok, A. Sistla, and J. Srinivasan. Quantitative temporal reasoning. In *Proc. Workshop on Automatic Verification Methods for Finite State Systems*, Grenoble, 1989.

[ESS89] E. Emerson, T. Sadler, and J. Srinivasan. Efficient temporal reasoning. In *Proc. 16^{th} ACM Symp. on Principles of Programming Languages*, pages 166–178, Austin, Texas, 1989.

[Fel83] Y.A. Feldman. A decidable propositional probabilistic dynamic logic. In *Proc. 15^{th} ACM Symp. on Theory of Computing*, pages 298–309, Boston, 1983.

[Gib85] A. Gibbons. *Algorithmic Graph Theory.* Cambridge University Press, 1985.

[Gri81] D. Gries. *The Science of Programming.* Springer Verlag, 1981.

[HS84] S. Hart and M. Sharir. Probabilistic temporal logics for finite and bounded models. In *Proc. 16^{th} ACM Symp. on Theory of Computing*, pages 1–13, 1984.

[HSP83] S. Hart, M. Sharir, and A. Pnueli. Termination of probabilistic concurrent programs. *ACM Trans. on Programming Languages and Systems*, 5:356–380, 1983.

[HV86] M.A. Holliday and M.K. Vernon. The gtpn analyzer: numerical methods and user interface. Technical Report 639, Sept. CS, Univ. Wisconsin – Madison, Apr. 1986.

[HV87a] M.A. Holliday and M.K. Vernon. Exact performance estimates for multiprocessor memory and bus interface. *IEEE Trans. on Computers*, C-36:76–85, Jan. 1987.

[HV87b] M.A. Holliday and M.K. Vernon. A generalized timed Petri net model for performance analysis. *IEEE Trans. Software Eng..*, SE-13(12), 1987.

[JM86] F. Jahanian and A. Mok. Safety analysis of timing properties in real-time systems. *IEEE Trans. on Software Engineering*, SE-12(9):890–904, Sept. 1986.

[Jos88] M. Joseph, editor. *Formal Techniques in Real-Time and Fault-Tolerant Systems*. LNCS 331, Springer-Verlag, 1988.

[KSK76] J. Kemeny, L. Snell, and A. Knapp. *Denumerable Markov Chains*. Springer Verlag, 1976.

[KVdR83] R. Koymans, J. Vytopil, and W.P. de Roever. Real-time programming and asynchronous message passing. In *Proc. 2nd ACM Symp. on Principles of Distributed Computing, Minaki, Canada*, pages 187–197, Montreal, Canada, 1983.

[LS82] D. Lehmann and S. Shelah. Reasoning with time and chance. *Information and Control*, 53:165–198, 1982.

[Mol82] M.K. Molloy. Performane analysis using stochastic petri nets. *IEEE Trans. on Computers*, C-31(9):913–917, Sept. 1982.

[OL82] S. Owicki and L. Lamport. Proving liveness properteis of concurrent programs. *ACM Trans. on Programming Languages and Systems*, 4(3):455–495, 1982.

[OW87] J. Ostroff and W. Wonham. Modelling, specifying and verifying real-time embedded computer systems. In *Proc. IEEE Real-time Systems Symp.*, pages 124–132, Dec. 1987.

[Par85] Joachim Parrow. *Fairness Properties in Process Algebra*. PhD thesis, Uppsala University, Uppsala, Sweden, 1985. Available as report DoCS 85/03, Department of Computer Systems, Uppsala University, Sweden.

[PH88] A. Pnueli and E. Harel. Applications of temporal logic to the specification of real-time systems. In M. Joseph, editor, *Proc. Symp. on Formal Techniques in Real-Time and Fault-Tolerant Systems*, pages 84–98. Springer Verlag, 1988. LNCS 331.

[Pnu82] A. Pnueli. The temporal semantics of concurrent programs. *Theoretical Computer Science*, 13:45–60, 1982.

[PZ86] A. Pnueli and L. Zuck. Verification of multiprocess probabilistic protocols. *Distributed Computing*, 1(1):53–72, 1986.

[Raz84] R.R. Razouk. The derivation of performance expressions for communication protocols from timed Petri net models. In *Proc. ACM SIGCOMM '84*, pages 210–217, Montréal, Québec, 1984.

[RP84] R.R. Razouk and C.V. Phelps. Performance analysis of timed Petri net models. In *Proc. Ifip WG 6.2 Symp. on Protocol Specification, Testing, and Verification IV*, pages 126–129. North-Holland, June 1984.

[SL87] A.U. Shankar and S.S. Lam. Time dependent distributed systems: Proving safety, liveness and real-time properties. *Distributed Computing*, 2, 1987.

[Var85] M. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *Proc. 26^{th} Annual Symp. Foundations of Computer Science*, pages 327–337, 1985.

[VH86] M.K. Vernon and M.A. Holliday. Performance analysis of multiprocessor cache concistency protocols using generalized timed Petri nets. In *Proc. of Performance 86 and ACM SIGMETRICS 1986 Joint conf. on Computer Performance Modelling, Measurement, and Evaluation*, pages 9–17. ACM press, May 1986.

[VW86] M.Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. IEEE Symp. on Logic in Computer Science*, pages 332–344, June 1986.

[Zub85] Zuberek. Performance evaluation using extended timed Petri nets. In *Proc. International Workshop on Timed Petri Nets*, Torino Italy, 1985. IEEE Computer Society 674.

A Proofs of Some Claims

A.1 Proof of the Recurrence Equation in Section 4.1

Proposition 3 *Assume that in a structure, states that satisfy f_1 (f_2) have been labeled by f_1 (f_2). Define $\mathcal{P}(t,s)$ to be 0 if $t < 0$, and otherwise through the recurrence equation*

$$\begin{aligned}\mathcal{P}(t,s) = \quad & \textit{if } f_2 \in \textit{label}(s) \textit{ then } 1 \\ & \textit{else} \quad \textit{if } f_1 \notin \textit{label}(s) \textit{ then } 0 \\ & \qquad \textit{else } \sum_{s' \in S} \mathcal{T}(s,s') \times \mathcal{P}(t-1,s').\end{aligned}$$

Then $\mathcal{P}(t,s)$ is the μ_m-measure for the set of paths σ from s for which $\sigma \models\!\!\!\equiv_K f_1\ U^{\leq t}\ f_2$.

Proof: For states s and integers t, let $\chi(t,s)$ be the set of finite sequences $s \rightarrow s_1 \rightarrow \ \cdots \ \rightarrow s_j$ of states from s such that $j \leq t$, $s_j \models_K f_2$, and for all i with $0 \leq i < j$ we have $s_i \models_K f_1$ and $s_i \not\models_K f_2$. Let $\mu_m^t(s)$ denote the μ_m-measure of the set of paths σ from s for which $\sigma \models\!\!\!\equiv_K f_1\ U^{\leq t}\ f_2$. By definition, $\mu_m^t(s)$ satisfies

$$\mu_m^t(s) \ = \sum_{s \rightarrow s_1 \rightarrow \ \cdots \ \rightarrow s_j \in \chi(t,s)} \mathcal{T}(s,s_1) \times \cdots \times \mathcal{T}(s_{j-1},s_j)$$

We consider three cases.

Case $s \models_K f_2$: By definition, any path σ from s satisfies $\sigma \models\!\!\!\equiv_K f_1\ U^{\leq t}\ f_2$ when $t \geq 0$, hence $\mu_m^t(s) = 1$.

Case $s \not\models_K f_2$ and $s \not\models_K f_1$: By definition, for any path σ from s we have $\sigma \not\models\!\!\!\equiv_K f_1\ U^{\leq t}\ f_2$, hence $\mu_m^t(s) = 0$.

Case $s \not\models_K f_2$ and $s \models_K f_1$: Here we consider two cases.

- **Case $t = 0$:** By definition, for a path σ from s we have $\sigma \models\!\!\!\equiv_K f_1\ U^{\leq 0}\ f_2$ iff $s \models_K f_2$, thus $\mu_m^0(s) = 0$.
- **Case $t > 0$:** Since $s \not\models_K f_2$, any finite sequence in $\chi(t,s)$ will have at least two states. Hence we can denote each such sequence σ as $s \ \rightarrow \ \sigma'$, where σ' is the sequence σ minus its first state. For such a sequence we have $\sigma \in \chi(t,s)$ iff $\sigma' \in \chi(t-1,\sigma'[1])$.

Hence we have

$$\begin{aligned}
\mu_m^t(s) &= \sum_{s \to s_1 \to \cdots \to s_j \in \chi(t,s)} \mathcal{T}(s,s_1) \times \cdots \times \mathcal{T}(s_{j-1},s_j) \\
&= \sum_{s_1} \mathcal{T}(s,s_1) \times \sum_{s_1 \to \cdots \to s_j \in \chi(t-1,s_1)} \mathcal{T}(s_1,s_2) \times \cdots \times \mathcal{T}(s_{j-1},s_j) \\
&= \sum_{s_1} \mathcal{T}(s,s_1) \times \mu_m^{t-1}(s_1)
\end{aligned}$$

We see that $\mu_m^t(s)$ satisfies exactly the same recurrence equation as $\mathcal{P}(t,s)$. Since the equation has a unique solution, we conclude that $\mu_m^t(s) = \mathcal{P}(t,s)$.

□

A.2 The definitions of $\mathcal{P}(t,s)$ in Section 4.1 are equivalent

Proposition 4 *Assume that $\mathcal{P}(t,s)$ is defined by recurrence equation (1), and that $\overline{\mathcal{P}}(t)$ is defined by (3) in Section 4.1. Then $\mathcal{P}(t,s) = \overline{\mathcal{P}}(t)[s]$ for all states s and integers t.*

Proof: Assume a structure $K = \langle S, s^i, \mathcal{T}, L \rangle$. We consider different cases.

Case: $t = 0$
By Definition (1), we have $\mathcal{P}(0,s) = 1$, if $s \models_K f_2$, otherwise $\mathcal{P}(0,s) = 0$. By Definition (3), we have $\overline{\mathcal{P}}(0)[s] = 1$, if $s \in S_s$ (i.e., $s \models_K f_2$), otherwise $\overline{\mathcal{P}}(0)[s] = 0$. Thus the two definitions are equivalent.

Case: $t > 0$
Definition (3) gives $\overline{\mathcal{P}}(t) = M \times \overline{\mathcal{P}}(t-1)$. We consider three cases:

1. if $s \in S_s$ then (since $M[s,s'] = 1$ if $s = s'$ otherwise 0) we have $\overline{\mathcal{P}}(t)[s] = \overline{\mathcal{P}}(t-1)[s]$. Since $\overline{\mathcal{P}}(0)[s] = 1$ we conclude that $\overline{\mathcal{P}}(t)[s] = 1$.

2. if $s \in S_f$ then (since $M[s,s'] = 1$ if $s = s'$ otherwise 0) we have $\overline{\mathcal{P}}(t)[s] = \overline{\mathcal{P}}(t-1)[s]$. Since $\overline{\mathcal{P}}(0)[s] = 0$ we conclude that $\overline{\mathcal{P}}(t)[s] = 0$.

3. if $s \in S_i$ then (since $M[s,s'] = \mathcal{T}(s,s')$) we have
$$\overline{\mathcal{P}}(t)[s] = \sum_{s' \in S} \mathcal{T}(s,s') \times \overline{\mathcal{P}}(t-1)[s'].$$

In all three cases we see that $\overline{\mathcal{P}}(t)[s]$ has exactly the same definition as $\mathcal{P}(t,s)$. □

A.3 Proof of Correctness of algorithm *LABEL_EU*

We will use standard program verification methods (see e.g. Gries [Gri81]) to prove the correctness of the algorithm LABEL_EU.

As a first step we add assertions to the program. The result is shown in Figure 5. It is assumed that $s \models_K f_1 \Leftrightarrow labeled(s, f_1)$ and that $s \models_K f_2 \Leftrightarrow labeled(s, f_2)$. P0 is the initial assumption, Li denotes the i:th statement of the program, P, P', and Pi denote assertions, and Q denotes the property to be verified. Termination follows trivially, since the only loop is a "for"-loop.

LABEL_EU:

P0 $= \{\forall s \in S : \neg labeled(s, f)\}$

L1: $\langle unseen := (S_i \cup S_s);\ fringe := S_s;\ mr := min(|S_i|, t)\rangle$

L2: for $i := 0$ to mr do

P = $\{\forall s \in S :$

$labeled(s, f) \Leftrightarrow \left(s \models_K f_1\ U_{>0}^{\leq i-1}\ f_2\right)$

$\neg labeled(s, f) \Leftrightarrow s \in (unseen \cup S_f) \Leftrightarrow s \not\models_K f_1\ U_{>0}^{\leq i-1}\ f_2$

$s \in fringe \Leftrightarrow \left(s \models_K f_1\ U_{>0}^{\leq i}\ f_2\ \wedge\ s \not\models_K f_1\ U_{>0}^{\leq i-1}\ f_2\right)$

$(unseen \cup fringe) \subseteq (S_s \cup S_i)$

L3: $\langle \forall s \in fringe\ do\ addlabel(s, f)\rangle$

L4: $\langle unseen :=\ unseen - fringe\rangle$

L5: $\langle fringe :=\ \{s : (s \in unseen\ \wedge\ \exists s' \in fringe : (\mathcal{T}(s, s') > 0))\}\rangle$

P' = $\forall s \in S :$

$labeled(s, f) \Leftrightarrow \left(s \models_K f_1\ U_{>0}^{\leq i}\ f_2\right)$

$\neg labeled(s, f) \Leftrightarrow s \in (unseen \cup S_f) \Leftrightarrow s \not\models_K f_1\ U_{>0}^{\leq i}\ f_2$

$s \in fringe \Leftrightarrow \left(s \models_K f_1\ U_{>0}^{\leq i+1}\ f_2\ \wedge\ s \not\models_K f_1\ U_{>0}^{\leq i}\ f_2\right)$

$(unseen \cup fringe) \subseteq (S_s \cup S_i)$

Q $= \left\{\forall s \in S : s \models_K f_1\ U_{>0}^{\leq t}\ f_2 \Leftrightarrow labeled(s, f)\right\}$

Figure 5: The algorithm Label_EU with added assertions

In the verification we will verify the Hoare-triple {P0} Label_EU {Q}.

This is done by verifying the following three Hoare-triples ($X[z/y]$ denotes X with all occurrences of y replaced by z):

1. {P0} L1 {P[0/i]}
2. {P} L2 L3 L4 {P'}
3. P'[mr/i] $\Rightarrow$ Q

1. Here, we must verify the three conjuncts of P after the assignment L1, i.e., we must verify that P0 implies

$$(\forall s \in S) \left[\begin{array}{l} labeled(s,f) \Leftrightarrow \left(s \models_K f_1\ U_{>0}^{\leq -1}\ f_2\right) \\ \neg labeled(s,f) \Leftrightarrow s \in (S_i \cup S_s \cup S_f) \Leftrightarrow s \not\models_K f_1\ U_{>0}^{\leq -1}\ f_2 \\ s \in S_s \Leftrightarrow \left(s \models_K f_1\ U_{>0}^{\leq 0}\ f_2\ \wedge\ s \not\models_K f_1\ U_{>0}^{\leq -1}\ f_2\right) \\ (S_s \cup S_i) \subseteq (S_s \cup S_i) \end{array} \right]$$

   The first conjunct is trivially true since no states are labeled. The second conjunct follows, since no states satisfies $f_1\ U_{>0}^{\leq -1}\ f_2$ and $S = S_i \cup S_s \cup S_f$. The third conjunct follows since for $s \in S_s$ we have $s \models_K f_1\ U_{>0}^{\leq 0}\ f_2$. The fourth conjunct is trivial.

2. Here we must verify that P implies the following assertion:

$$(\forall s \in S) \left[\begin{array}{l} labeled(s,f)\ \vee\ s \in fringe \Leftrightarrow \left(s \models_K f_1\ U_{>0}^{\leq i}\ f_2\right) \\ \neg(labeled(s,f)\ \vee\ s \in fringe) \Leftrightarrow s \in ((unseen - fringe) \cup S_f) \Leftrightarrow s \not\models_K f_1\ U_{>0}^{\leq i}\ f_2 \\ (s \in unseen\ \wedge\ \exists s' \in fringe : (\mathcal{T}(s,s') > 0)) \Leftrightarrow \\ \qquad\qquad \Leftrightarrow \left(s \models_K f_1\ U_{>0}^{\leq i+1}\ f_2\ \wedge\ s \not\models_K f_1\ U_{>0}^{\leq i}\ f_2\right) \\ unseen \subseteq (S_s \cup S_f) \end{array} \right]$$

   The first conjunct follows from the first and third conjuncts in P. The second conjunct follows from the second, third, and fourth conjuncts in P. The third conjunct follows from the second and third conjunct in P. The fourth follows trivially from the fourth conjunct in P.

3. Follows directly from the first part of P' and Proposition 1.

A.4 Correctness of Equation (4)

$\mathcal{P}(\infty, s)$ is the μ_m measure for the set of paths σ for which $\sigma \models_K f_1\ U^{\leq \infty}\ f_2$.

Proposition 5 *Assume that in a structure, states that satisfy f_1 (f_2) have been labeled by f_1 (f_2). Let Q and R be as defined in Section 4.2. Then the solution of the system of linear equations*

$$\begin{aligned}\mathcal{P}(\infty, s) = \quad & \textit{if } s \in R \textit{ then } 1 \\ & \textit{else} \quad \textit{if } s \in Q \textit{ then } 0 \\ & \qquad \textit{else} \sum_{s' \in S} \mathcal{T}(s, s') \times \mathcal{P}(\infty, s')\end{aligned}$$

satisfy that for each s, $\mathcal{P}(\infty, s)$ is the μ_m measure for the set of paths σ from s for which $\sigma \models_K f_1\ U^{\leq \infty}\ f_2$.

Proof: If $s \in R$ or $s \in Q$, then the definitions of R and Q imply the proposition. In other cases, we can analogously to the proof of Proposition 3 show that the μ_m measure for the set of paths σ from s for which $\sigma \models_K f_1\ U^{\leq \infty}\ f_2$ satisfies the same equations as $\mathcal{P}(\infty, s)$. This shows the existence of a solution of the equations which is the desired one. The uniqueness of the solution can be seen as follows. Assume that there are two solutions. Then the difference between them, denoted $\Delta(s)$ for $s \in S$, satisfies the equations

$$\Delta(s) = \sum_{s' \in S \setminus (R \cup Q)} \mathcal{T}(s, s') \times \Delta(s')$$

for $s \in S \setminus (R \cup Q)$. We know $\sum_{s' \in S} \mathcal{T}(s, s') = 1$ for all $s \in S$. If we consider the set Max of states s in $S \setminus (R \cup Q)$ for which $\Delta(s)$ has the highest absolute value, this implies that there are no transitions from a state in Max to a state outside Max with non-zero probability. This would imply that $Max \subseteq Q$ which is a contradiction. □

B Additional Algorithms

B.1 Algorithm 1 in Section 4.1 modified for the Unless case

Let us introduce the function $\mathcal{R}(t, s)$ for $s \in S$, t an integer. We define $\mathcal{R}(t, s)$ to be the μ_m-measure for the set of paths σ from s for which $\sigma \models_K f_1\ \mathcal{U}^{\leq t}\ f_2$. If $t < 0$, then we use the convention that $\mathcal{R}(t, s) = 1$. Analogously to $\mathcal{P}(t, s)$, we can define $\mathcal{R}(t, s)$ for $t \geq 0$ as follows:

$$\mathcal{R}(t,s) = \begin{array}{l} \text{if } f_2 \in label(s) \text{ then } 1 \\ \quad \text{else} \quad \text{if } f_1 \notin label(s) \text{ then } 0 \\ \qquad\quad \text{else } \sum_{s' \in S} \mathcal{T}(s,s') \times \mathcal{R}(t-1,s') \end{array} \tag{5}$$

Note that the difference in the definition of $\mathcal{P}(t,s)$ and $\mathcal{R}(t,s)$ is derived only from the values for $t < 0$. The following algorithm calculates $\mathcal{R}(t,s)$:

Algorithm 1':

```
for i := 0 to t do
  for all s ∈ S do
    if f2 ∈ label(s) then R(i,s) := 1
    else begin
      R(i,s) := 0;
      if f1 ∈ label(s) then for all s' ∈ S do
          R(i,s) := R(i,s) + T(s,s') * R(i-1,s')
    end
```

The state s will be labeled with $f_1 \; U^{\leq t}_{\geq p} \; f_2$ if $\mathcal{R}(t,s) \geq p$.

B.2 Algorithm 2 in Section 4.1 modified for the Unless case

Analogusly to Algorithm 1' above we can define an algorithm for the Unless case that corresponds to Algorithm 2.

Algorithm 2':

```
for all s ∈ S do
  if f2 ∈ label(s) or f1 ∈ label(s) then R̄(0)[s] := 1
  else R̄(0)[s] := 0;
R̄(t) = M^t * R̄(0)
```

B.3 Algorithm LABEL_EUnless

The algorithm LABEL_EUnless labels states s for which $s \models_K f_1 \; \mathcal{U}^{\leq t}_{>0} \; f_2$ with $f_1 \; \mathcal{U}^{\leq t}_{>0} \; f_2$. Intuitively, the algorithm will not label states in $(S_i \cup S_f)$ from which all sequences of states of length $\leq t$ pass through S_f.

LABEL_EUnless:

unseen := S_i;
fringe := S_f;
bad := $\emptyset$;
mr := $\min(|S_i|, t)$;
for i:=0 to mr do { bad:= bad $\cup$ fringe;
unseen := unseen - fringe;
fringe := $\{s : [s \in \text{ unseen } \wedge \forall s' : (\mathcal{T}(s,s') > 0 : (s' \in \text{bad}))]\}$
$\forall s \notin$ bad do addlabel(s,f);

Intuitively, the variable bad will after passing through the for-loop with index i contain all states in $S_f \cup S_i$ from which all sequences of states of length $\leq i$ pass through S_f.

B.4 Algorithm LABEL_AUnless

The algorithm LABEL_AUnless labels states, s, for which $s \models_K f_1 \, \mathcal{U}_{\geq 1}^{\leq t} \, f_2$ with $f_1 \, \mathcal{U}_{\geq 1}^{\leq t} \, f_2$. Intuitively, the algorithm will not label states in $S_i \cup S_f$ from which there is a sequence of states in $S_i \cup S_f$ of length at most t which ends in S_f.

LABEL_AUnless:

unseen := S_i;
fringe := S_f;
bad := $\emptyset$;
mr := $\min(|S_i|, t)$;
for i:=0 to mr do { bad := bad $\cup$ fringe;
unseen := unseen - fringe;
fringe := $\{s : [s \in \text{ unseen } \wedge \exists s' : (\mathcal{T}(s,s') > 0 : (s' \in \text{fringe})$
$\forall s \notin$ bad do addlabel(s,f);

Intuitively, the variable bad will after passing through the for-loop with index i contain all states in $S_f \cup S_i$ from which there is a sequence of states of length $\leq i$ which passes through S_f without going to S_s.

B.5 Algorithm Identify_Q

All states in S_f, and all states from which it is not possible to reach a state in S_s should be included in Q. The algorithm is the same as the algorithm LABEL_EU for $t = |S_i|$.

Identify_Q:

unseen := $S_i \cup S_s$;
fringe := S_s;
mark := $\emptyset$;
for i:=0 to $|S_i|$ do { mark := mark $\cup$ fringe;
unseen := unseen - fringe;
fringe := $\{s : (s \in$ unseen $\wedge\ \exists s' \in$ fringe : $(\mathcal{T}(s, s') > 0))\}$;
Q := S - mark;

B.6 Algorithm Identify_R

All states for which the μ_m measure is 1 for eventually reaching a success state should be included in R. These are exactly the states in S_s, and the states in S_i from which there is no sequence of transitions outside S_s with non-zero probability, leading to Q or S_f.

Identify_R:

Identify_Q;
unseen := S_i;
fringe := $S_f \cup Q$;
mark := $\emptyset$;
for i:=0 to $|S_i|$ do { mark := mark $\cup$ fringe;
unseen := unseen - fringe;
fringe := $\{s : (s \in$ unseen $\wedge\ \exists s' \in$ fringe : $(\mathcal{T}(s, s') > 0))\}$;
R := S - mark;

In the algorithm, first Q becomes the set of states from which no success states are reachable. Then mark becomes the states from which a state in S_f or Q is reachable. Thus, R should be the complement of the set mark.