Formal Aspects of Computing (1994) 6: 512–535
© 1994 BCS

A Logic for Reasoning about Time and Reliability[1]

Hans Hansson and Bengt Jonsson

Department of Computer Systems, Uppsala University, Uppsala, Sweden

Keywords: Markov chains; Modal logic; CTL; Real time; Probability; Soft deadlines; Automatic verification; Model checking

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

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 behaviour is an important aspect of the correctness of the system,

[1] This paper 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 10th IEEE Real-time Systems Symposium, Santa Monica CA, December 1989. The work presented here was performed while the authors were employed by the Swedish Institute of Computer Science (SICS), and partially supported by the Swedish Board for Technical Development (ESPRIT/BRA project 3096, SPEC) and the Swedish Telecommunication Administration (project: PROCOM).
Correspondence and offprint requests to: Hans Hansson, Department of Computer Systems, Uppsala University, Box 325, S-751 05, Uppsala, Sweden. Email: hansh@docs.uu.se

and the interest for research on these aspects of formal methods seems to be increasing (see e.g. [Jos88, Vyt91, dBH92]).

For some systems, it is very important that certain time bounds on their behaviour 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 (e.g. [PnH88, ShL87, OsW87]), or develop logics that deal explicitly with time quantities (e.g. [BeH81, JaM86, KVR83, EMS92]).

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, RaP84, HoV87b].

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.

In this paper, we present a logic called PCTL for stating soft deadlines. The logic is based on Emerson, Clarke, and Sistla's Computation Tree Logic (CTL) [CES86]. CTL is a modal (temporal) logic for reasoning about qualitative programme 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). Independently of the work presented here, Emerson, Mok, Sistla, and Srinivasan [EMS92] 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 PCTL, 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. To enable reasoning about soft deadlines we have replaced path quantifiers by probabilities. Thus, instead of saying that some property holds for all paths or for some paths, we can express that a property holds for a certain fraction of the paths. 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, as defined e.g., by Hart and Sharir [HaS84] and others [LeS82, CVW86]. However, these works only deal with properties that either hold with probability one or with a non-zero probability.

The paper is organised as follows. 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 summarise the results and indicate directions for further work.

2. Probabilistic Real Time Computation Tree Logic

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*, denoted by a, a_1, etc. Formulas in PCTL are built from atomic propositions, propositional logic connectives and operators for expressing time and probabilities.

Definition 1. (PCTL Syntax) The syntax of PCTL formulas is defined inductively as follows:

- Each atomic proposition a is a PCTL formula,
- If f_1 and f_2 are PCTL formulas, then so are $\neg f_1$ and $(f_1 \wedge f_2)$,
- If f_1 and f_2 are PCTL formulas, t is a nonnegative integer or ∞, and p is a real number with $0 \leq p \leq 1$, then $f_1 \; U^{\leq t}_{\geq p} \; f_2$ and $f_1 \; U^{\leq t}_{>p} \; f_2$ are PCTL formulas. □

We shall use f, f_1, etc. to range over PCTL formulas. Intuitively, the PCTL formulas represent properties of states. The propositional connectives $\neg$ and $\wedge$ have their usual meanings. The operator U is the (strong) until operator. The formula $f_1 \; U^{\leq t}_{\geq p} \; f_2$ expresses that with at least probability p 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. The formula $f_1 \; U^{\leq t}_{>p} \; f_2$ has analogous meaning.

PCTL formulas are interpreted over structures that are discrete time Markov chains. A specified initial state is associated with the structure, and for each state there is an assignment of truth values to atomic propositions appearing in a given formula.

Definition 2. (Structure) A *structure* is a quadruple $\langle S, s^i, \mathscr{T}, L\rangle$, where

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

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

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

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

$L : S \rightarrow 2^A$ is a *labelling function* assigning atomic propositions to states. □

Intuitively a structure represents a system, which at any instant is in one of its states. At each time unit, the system changes state according to a probability distribution given by the transition probability function. Thus, each transition can be considered to require one time unit. We will display structures as transition diagrams, where states (circles) are labelled with atomic propositions and transitions with non-zero probability are represented as arrows labelled with their probabilities (e.g., the arrow from state s_k to state s_l is labelled with $\mathscr{T}(s_k, s_l)$). The initial state (s^i) is indicated with an extra arrow. For example, Figure 1 shows a structure (K) with 4 states and 5 transitions with non-zero probability. The state A, labelled with a_1, a_2, is the initial state.

A *path* σ from a state s_0 in a structure is an infinite sequence $s_0 \; s_1 \; s_2 \; \ldots$ of states with s_0 as the first state. The state s_n in σ is denoted $\sigma[n]$, and the prefix $s_0 \; \ldots \; s_n$ of σ is denoted $\sigma\uparrow n$.

For each structure K and state s we will define a probability measure μ^K_s on the set of paths from s.

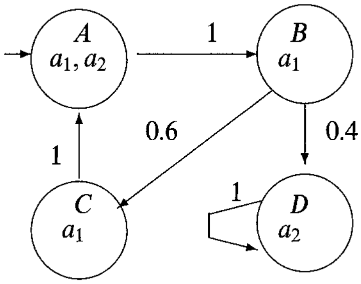

Fig. 1. The sample structure K.

Definition 3. (Probability measure) Let $paths^K_{s_0}$ denote the set of paths of K starting in s_0. In accordance with measure theory, we define:

- For any sequence $s_0 \ldots s_n$, starting in s_0

  $$\mu^K_{s_0}(\{\sigma \in paths^K_{s_0} \mid \sigma\uparrow n = s_0 \ldots s_n\}) = \mathscr{T}(s_0, s_1) \times \cdots \times \mathscr{T}(s_{n-1}, s_n)$$

  i.e., the measure of the set of paths σ for which $\sigma\uparrow n = s_0 \ldots s_n$ is equal to the product $\mathscr{T}(s_0, s_1) \times \cdots \times \mathscr{T}(s_{n-1}, s_n)$.
- For $n = 0$

  $$\mu^K_{s_0}(\{\sigma \in paths^K_{s_0} \mid \sigma\uparrow 0 = s_0\}) = 1.$$

- For any countable set $\{X_i\}_{i\in I}$ of disjoint subsets of $paths^K_{s_0}$

  $$\mu^K_{s_0}\left(\bigcup_{i\in I} X_i\right) = \sum_{i\in I} \mu^K_{s_0}(X_i)$$

  Note that the sum is well-defined, since it is bounded by 1 and each summand is non-negative.
- If X is a subset of $paths^K_{s_0}$, then the measure of the complement set $paths^K_{s_0} \setminus X$ is defined as:

  $$\mu^K_{s_0}(paths^K_{s_0} \setminus X) = 1 - \mu^K_{s_0}(X)$$ □

As an illustration, consider the labelled Markov chain K in Fig. 1. The set of paths $paths^K_A$ of K is infinite. The set of paths with non-zero probability can be characterised by the ω-regular expression $AB(CAB)^*D^\omega$. The probability measure for the set of paths in K for which C is visited exactly n times is given by

$$\mu^K_A(\{AB(CAB)^n D^\omega\}) = 0.6^n \times 0.4$$

For $n \neq m$ the set of paths $\{AB(CAB)^n D^\omega\}$ is disjoint from $\{AB(CAB)^m D^\omega\}$. Hence,

$$\mu^K_A(\{AB(CAB)^* D^\omega\}) = \sum_{i\in\mathscr{N}} \mu^K_A(\{AB(CAB)^i D^\omega\}) = 1$$

Also, it follows that the measure of the set of paths which never reaches D is 0.

In the definition of PCTL we will use the auxiliary *path formula* $f_1\ U^{\leq t}\ f_2$ to represent properties of *paths*, i.e., sequences of states. Intuitively, $f_1\ U^{\leq t}\ f_2$ holds for a particular path if f_2 becomes *true* within t time units and f_1 holds until then.

We will define the truth of PCTL-formulas for a state s in a structure K by a satisfaction relation

$$s \models_K f$$

which intuitively means that the PCTL-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

$$\sigma \models_K f_1 \; U^{\leq t} \; f_2$$

which intuitively means that the path σ in K satisfies the path formula $f_1 \; U^{\leq t} \; f_2$.

Definition 4. (PCTL Semantics) The relations $\models_K$ and $\models_K$ are inductively defined as follows:

$$\begin{array}{ll}
s \models_K a & \text{iff } a \in L(s) \\
s \models_K \neg f & \text{iff not } s \models_K f \\
s \models_K f_1 \wedge f_2 & \text{iff } s \models_K f_1 \text{ and } s \models_K f_2 \\
\sigma \models_K f_1 \; U^{\leq t} \; f_2 & \text{iff there exists an } i \leq t \text{ such that} \\
 & \sigma[i] \models_K f_2 \text{ and} \\
 & \sigma[j] \models_K f_1, \text{ for all } j : \; 0 \leq j < i. \\
s \models_K f_1 \; U^{\leq t}_{\geq p} \; f_2 & \text{iff } \mu^K_s(\{\sigma \mid \sigma[0] = s \wedge \sigma \models_K f_1 \; U^{\leq t} \; f_2\}) \geq p. \\
s \models_K f_1 \; U^{\leq t}_{> p} \; f_2 & \text{iff } \mu^K_s(\{\sigma \mid \sigma[0] = s \wedge \sigma \models_K f_1 \; U^{\leq t} \; f_2\}) > p.
\end{array}$$

□

We will use $\models_K f$ to denote $s^i \models_K f$, where s^i is the initial state of K.

In the following we will use the abbreviations (derived operators):

$$\begin{array}{rcl}
f_1 \vee f_2 & \equiv & \neg(\neg f_1 \wedge \neg f_2) \\
f_1 \rightarrow f_2 & \equiv & \neg f_1 \vee f_2 \\
f_1 \; \mathcal{U}^{\leq t}_{\geq p} \; f_2 & \equiv & \neg\left(\neg f_2 \; U^{\leq t}_{>1-p} \; \neg(f_1 \vee f_2)\right) \\
f_1 \; \mathcal{U}^{\leq t}_{> p} \; f_2 & \equiv & \neg\left(\neg f_2 \; U^{\leq t}_{\geq 1-p} \; \neg(f_1 \vee f_2)\right)
\end{array}$$

Intuitively, the propositional connectives $\vee$ and $\rightarrow$ have their usual meanings. The operator $\mathcal{U}$ is the *unless* (or weak until) operator. The formula $f_1 \; \mathcal{U}^{\leq t}_{\geq p} \; f_2$ means that with at least probability p either f_1 will remain true for at least t time units, or f_2 will become true within t time units and f_1 will be true from now on until f_2 becomes true. The formula $f_1 \; \mathcal{U}^{\leq t}_{> p} \; f_2$ has analogous meaning.

3. Properties Expressible in PCTL

In this section we present examples of properties that can be expressed in PCTL. First, we discuss why PCTL is 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. Consider for instance the PCTL formula $f_1\ U^{\leq\infty}_{\geq 0.5}\ f_2$ which does not have a CTL analogue. Analogues to universal and existential quantifications in CTL can however be expressed in PCTL, e.g.

$$
\begin{aligned}
A[f_1\ U\ f_2] &\equiv f_1\ U^{\leq\infty}_{\geq 1}\ f_2\\
E[f_1\ U\ f_2] &\equiv f_1\ U^{\leq\infty}_{>0}\ f_2\\
AGf &\equiv f\ \mathcal{U}^{\leq\infty}_{\geq 1}\ \mathit{false}\\
AFf &\equiv \mathit{true}\ U^{\leq\infty}_{\geq 1}\ f\\
EGf &\equiv f\ \mathcal{U}^{\leq\infty}_{>0}\ \mathit{false}\\
EFf &\equiv \mathit{true}\ U^{\leq\infty}_{>0}\ f
\end{aligned}
$$

Intuitively, $A[f_1\ U\ f_2]$ means that $f_1\ U\ f_2$ holds for all paths, or more precisely, for a set of paths with probability measure 1. The formula $E[f_1\ U\ f_2]$ means that there exists a path for which $f_1\ U\ f_2$ holds, or more precisely, there exists a set of paths with non-zero probability measure for which $f_1\ U\ f_2$ holds. The formula 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 with probability 1 eventually be reached, 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.

Quantitative time allows us to specify time-critical properties that relate the occurrence of events in a system in real-time. This is very important for programmes 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, e.g. 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 respond to stimuli with an appropriate action, e.g., every time a controller receives an alarm signal from a sensor the controller should take an appropriate action. Owicki and Lamport [OwL82] have defined a *leads-to* operator ($a \leadsto b$), with the intuitive meaning that whenever a becomes true, b will eventually hold. We can in PCTL define a quantified leads-to operator as follows:

$$f_1 \underset{\geq p}{\overset{\leq t}{\leadsto}} f_2 \equiv AG(f_1 \rightarrow F^{\leq t}_{\geq p} f_2)$$

Intuitively, $f_1 \underset{\geq p}{\overset{\leq t}{\leadsto}} f_2$ means that whenever f_1 holds there is a probability of at least p that f_2 will hold within t time units.

As an example consider an industrial controller which monitors the level of fluid in a container. An alarm is raised when the fluid reaches a critical level. The controller should respond to an alarm by opening a valve within 4 time units. This can in PCTL be expressed as:

$$alarm \underset{\geq 1}{\overset{\leq 4}{\leadsto}} open\ valve$$

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

$$alarm \underset{\geq 0.99}{\overset{\leq 4}{\leadsto}} open\ valve$$

Relaxing the timing requirement might enable a less costly implementation that still shows acceptable behaviour. 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 the controller to always respond within 10 time units, and almost always within 4 time units we get the property:

$$(alarm \underset{\geq 1}{\overset{\leq 10}{\leadsto}} open\ valve) \wedge (alarm \underset{\geq 0.99}{\overset{\leq 4}{\leadsto}} open\ valve)$$

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 [CES86]. It is designed so that when it terminates each state will be labelled with the set of subformulas of f that are true in that state. One can then conclude that $\models_K f$ iff the initial state (s^i) is labelled with f.

For each state s of the structure, the algorithm uses a variable *label*(s) to indicate the subformulas that have been found to be true in s. Initially, each state s is labelled with the atomic propositions that are *true* in s, i.e., *label*(s) := $L(s)$, $\forall s \in S$. The labelling is then performed starting with the smallest subformulas of f that have not yet been labelled, and ending with labelling states with f itself. Composite formulas are labelled based on the labelling of their parts. Assuming that we have performed the labelling of f_1 and f_2, the labelling 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.,

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

Probability	Time		
	0	finite	∞
>0	$s \models_K f_2$	$E[f_1\ U^{\leq t}\ f_2]$ $\mathcal{O}(\lvert S\rvert)$	CTL $E[f_1\ U\ f_2]$ $\mathcal{O}(\lvert S\rvert)$
$0<p<1$	$s \models_K f_2$	The general case $\mathcal{O}(t\times(\lvert S\rvert + \lvert E\rvert))$ or $\mathcal{O}(\log t \times \lvert S\rvert^3)$	"probabilistic CTL" $\mathcal{O}(\lvert S\rvert^{2.81})$
≥ 1	$s \models_K f_2$	$A[f_1\ U^{\leq t}\ f_2]$ $\mathcal{O}(\lvert S\rvert)$	CTL $A[f_1\ U\ f_2]$ $\mathcal{O}(\lvert S\rvert)$

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

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 algorithms for labelling states with the modal subformulas of PCTL. In section 4.2 we present more efficient algorithms for labelling in cases with extreme parameter values (e.g. $p = 1$ and $p = 0$).

4.1. Labelling States with the Modal Subformulas of PCTL

In this section we give algorithms for labelling states with formulas of type $f_1\ U^{\leq t}_{\geq p}\ f_2$ and $f_1\ U^{\leq t}_{>p}\ f_2$. We will present several algorithms with different suitability for different values of the t and p parameters. Table 1 gives a classification of possible combinations of p and t parameter values as well as complexities of performing the labelling using the algorithms we will propose.

For the three entries in the left column, corresponding to $t = 0$, the labelling problem collapses to the problem of labelling states with f_2. The cases in the middle row of Table 1 will be considered in this section. Note that, we will give two algorithms for the case with finite t value and $0 < p < 1$. The first is suitable for small t parameter values, while the second performs better for large t values. In section 4.2 we will present alternative algorithms for the remaining cases of the table: $f_1\ U^{\leq t}_{>0}\ f_2$, $f_1\ U^{\leq \infty}_{>0}\ f_2$, $f_1\ U^{\leq t}_{\geq 1}\ f_2$, and $f_1\ U^{\leq \infty}_{\geq 1}\ f_2$, where $t < \infty$. As indicated in the table, model checking for $f_1\ U^{\leq \infty}_{>0}\ f_2$ is analogous to CTL model checking for the formula $E[f_1\ U\ f_2]$. Likewise, $f_1\ U^{\leq \infty}_{\geq 1}\ f_2$ corresponds to CTL model checking of $A[f_1\ U\ f_2]$. Model checking for $f_1\ U^{\leq t}_{>0}\ f_2$ amounts to check if there exists a sequence of states satisfying $f_1\ U^{\leq t}\ f_2$. This is analogous to RTCTL model checking [EMS92] for the formula $E[f_1\ U^{\leq t}\ f_2]$. Likewise, $f_1\ U^{\leq t}_{\geq 1}\ f_2$ corresponds to RTCTL model checking of $A[f_1\ U^{\leq t}\ f_2]$.

Modal Subformulas with Finite t Parameter Value

We shall give an algorithm for labelling states with the formula $f_1 \; U^{\leq t}_{\geq p} \; f_2$, assuming that we have done the labelling for formulas f_1 and f_2, and that $t \neq \infty$.

Definition 5. (Measure for Paths Satisfying Until-Formulas) For a state $s \in S$ in a structure K and an integer $t \geq 0$, we define the function $\mathscr{P}(t,s)$ to be the measure for the set of paths σ in $paths^K_s$ for which $\sigma \models_K f_1 \; U^{\leq t} \; f_2$. If $t < 0$, we define $\mathscr{P}(t,s) = 0$. □

Proposition 1. If $t \geq 0$ then $\mathscr{P}(t,s)$ satisfies the following recurrence equation:

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

Proof. For states s and integers t, let $\chi(t,s)$ be the set of finite sequences $s\, s_1 \ldots s_j$ 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 $\mathscr{P}(t,s)$ denote the measure of the set of paths σ in $paths^K_s$ for which $\sigma \models_K f_1 \; U^{\leq t} \; f_2$. By definition, $\mathscr{P}(t,s)$ satisfies

$$\mathscr{P}(t,s) \;=\; \sum_{s\, s_1 \ldots s_j \in \chi(t,s)} \mathscr{T}(s,s_1) \times \cdots \times \mathscr{T}(s_{j-1},s_j)$$

We consider three cases.

Case $s \models_K f_2$**:** By definition, any path σ in $paths^K_s$ satisfies $\sigma \models_K f_1 \; U^{\leq t} \; f_2$ when $t \geq 0$, hence $\mathscr{P}(t,s) = 1$.

Case $s \not\models_K f_2$ **and** $s \not\models_K f_1$**:** By definition, for any path σ in $paths^K_s$ we have $\sigma \not\models_K f_1 \; U^{\leq t} \; f_2$, hence $\mathscr{P}(t,s) = 0$.

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 in $paths^K_s$ we have $\sigma \models_K f_1 \; U^{\leq 0} \; f_2$ iff $s \models_K f_2$, thus $\mathscr{P}(t,s) = 0$.

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 $s\,\sigma'$, 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

$$\begin{aligned} \mathscr{P}(t,s) &= \sum_{s\, s_1 \ldots s_j \in \chi(t,s)} \mathscr{T}(s,s_1) \times \cdots \times \mathscr{T}(s_{j-1},s_j) \\ &= \sum_{s_1} \mathscr{T}(s,s_1) \times \sum_{s_1 \ldots s_j \in \chi(t-1,s_1)} \mathscr{T}(s_1,s_2) \times \cdots \times \mathscr{T}(s_{j-1},s_j) \\ &= \sum_{s_1} \mathscr{T}(s,s_1) \times \mathscr{P}(t-1,s_1) \end{aligned}$$

□

Recurrence equation 1 gives an algorithm that labels the state s with $f_1 \; U^{\leq t}_{\geq p} \; f_2$ if $\mathscr{P}(t,s) \geq p$.

Recurrence equation 1 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 labelled with f_2 (i.e., states for which $f_2 \in$ *label*(s)).

S_f - the *failure* states, are states which are not labelled with f_1 nor f_2 (i.e., states for which $f_1, f_2 \notin$ *label*(s)).

S_i - the *inconclusive* states, are states labelled with f_1 but not with f_2 (i.e., states for which $f_1 \in$ *label*(s) and $f_2 \notin$ *label*(s)).

Define the $|S| \times |S|$-matrix M by

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

For $t \geq 0$, let $\overline{\mathscr{P}}(t)$ be a column vector of size $|S|$ whose ith element $\overline{\mathscr{P}}(t)_i$ is $\mathscr{P}(t, s_i)$. Thus $\overline{\mathscr{P}}(0)_i$ is 1 if $s_i \in S_s$ and otherwise 0.

Proposition 2. It follows that

$$\overline{\mathscr{P}}(t) = \overbrace{M \times M \times \cdots \times M}^{t} \times \overline{\mathscr{P}}(0) = M^t \times \overline{\mathscr{P}}(0) \tag{3}$$

for $t > 0$.

Proof. We will use equation 1. When $t = 0$ the proposition follows by definition. For $t > 0$ equation 3 gives $\overline{\mathscr{P}}(t) = M \times \overline{\mathscr{P}}(t-1)$. We consider three cases:

1. if $s_n \in S_s$ then (since $M[s_n, s'] = 1$ if $s_n = s'$ otherwise 0) it follows that $\overline{\mathscr{P}}(t)_n$ is equal to $\overline{\mathscr{P}}(t-1)_n$. Since $\overline{\mathscr{P}}(0)_n = 1$ we conclude that $\overline{\mathscr{P}}(t)_n = 1$.
2. if $s_n \in S_f$ then analogously to Case 1. it follows that $\overline{\mathscr{P}}(t)_n = 0$.
3. if $s_n \in S_i$ then (since $M[s_n, s'] = \mathscr{T}(s, s')$) it follows that

$$\overline{\mathscr{P}}(t)_n = \sum_{s_m \in S} \mathscr{T}(s_n, s_m) \times \overline{\mathscr{P}}(t-1)_m$$

We see that $\overline{\mathscr{P}}(t)_i$ satisfies exactly the same recurrence equation as $\mathscr{P}(t, s_i)$. □

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

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 s for which $\mathscr{P}(t, s) > p$. Model checking for unless-formulas ($f_1 \ \mathscr{U}^{\leq t}_{\geq p} \ f_2$ and $f_1 \ \mathscr{U}^{\leq t}_{>p} \ f_2$) can be done via the dual formulas:

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

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

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

Calculating $\mathscr{P}(t, s)$.

We propose two algorithms for calculating $\mathscr{P}(t, s)$. 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
               $\mathcal{P}(i,s) := \mathcal{P}(i,s) + \mathcal{T}(s,s') \times \mathcal{P}(i-1,s')$
      end
```

Algorithm 2.

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

Algorithm 1 requires $\mathcal{O}(t \times |S|^2)^2$ arithmetical operations. Ignoring the zero-probability transitions in $\mathcal{T}$ we can reduce the number of arithmetical operations required to $\mathcal{O}(t \times (|S| + |E|))$, where $|E|$ is the number of transitions in $\mathcal{T}$ with non-zero probability. For a fully connected structure these expressions coincide, since in that case $|E| = |S|^2$. The matrix multiplication in algorithm 2 can be performed with $\mathcal{O}(log(t) \times |S|^3)$ arithmetical operations, since M^t can be calculated with $\mathcal{O}(log\ t)$ matrix multiplications, each requiring $|S|^3$ (or less) 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 $\mathcal{O}(t_{max} \times (|S| + |E|) \times |f|)$ or $\mathcal{O}(|S|^3 \times |f|)$ 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 above in the verification of a simple communication protocol.

Modal subformulas with $t = \infty$

In this section we consider labelling states with the formula $f_1\ U^{\leq \infty}_{\geq p}\ f_2$. The algorithms above cannot be used in this case, since they would require infinite calculations. Instead we define $\mathcal{P}(\infty, s)$ to be the measure for the set of paths σ in $paths^K_s$ 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. Inspired by Dijkstra's shortest path algorithm [Gib85] and observing that we only need to consider paths that are shorter than the number of inconclusive states $|S_i|$ we define the algorithm *Identify_Q* as follows:

[2] The actual worst case complexity is $((t+1) \times (|S| + 1) \times 2 \times |S|)$, 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.

Algorithm 3. *(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 := $\{s \mid s \in$ unseen $\wedge\ \exists s' \in$ fringe : $(\mathcal{T}(s,s') > 0)\}$;
Q := S $\setminus$ mark;
```

Intuitively, the algorithm marks all states from which there is a positive probability of reaching a state in S_s. Q is the complement of these states. In the algorithm, *unseen* denotes the set of states in S_i and S_s that have not yet been considered, *fringe* are the states that are being marked. After passing the for loop with index i, the algorithm will have marked all states that satisfy $f_1\ U^{\leq i}_{\geq 0}\ f_2$.

Similar to the definition of Q, we can extend the success states to also include states s in S_i for which the μ^K_s-measure for eventually reaching a success state ($s' \in S_s$) without passing S_f 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. An algorithm for identifying the states in R is given in the Appendix. The next step, when Q and R have been identified, is to solve the set of linear equations defined by:

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

These equations can be solved with Gaussian elimination, with a complexity of $\mathcal{O}\left[(|S| - |Q| - |R|)^{2.81}\right]$ [AHU74].

Proposition 3. $\mathcal{P}(\infty, s)$ is the μ^K_s measure for the set of paths σ in $paths^K_s$ 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 1 show that the measure for the set of paths σ in $paths^K_s$ 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

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

for $s \in S \setminus (R \cup Q)$. By definition, $\Delta(s) = 0$ if $s \in R \cup Q$ and $\sum_{s' \in S} \mathcal{T}(s,s') = 1$ for all $s \in S$. If $\Delta \neq 0$ for a non-empty subset in $S \setminus (R \cup Q)$, then 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. □

4.2. Special Algorithms when $p = 0$ or $p = 1$

In this section we will discuss alternative algorithms for cases when the modal operator has extreme probability (1 or 0) parameter value. 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 the Appendix.

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 labelled if there exists a path which is shorter than $t + 1$ from the state to a state in S_s. This can be done with an algorithm similar to Algorithm *Identify_Q*. We define the algorithm *LABEL_EU* as follows:

Algorithm 4. (*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 $\setminus$ fringe;
        fringe := $\{s \mid s \in \text{unseen} \wedge \exists s' \in \text{fringe} : (\mathscr{T}(s, s') > 0)\}$;
```

Intuitively, *unseen* is the set of states in S_i and S_s that have not yet been considered for labelling, *fringe* are the states that are being labelled, 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 labelled all states that satisfy $f_1 \ U^{\leq i}_{>0} \ f_2$.

Emerson et al. [EMS92] present a similar algorithm for model checking in RTCTL. A small difference compared with our algorithm is that they do not partition the state set.

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 4. 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 $f_1 \ U^{\leq |S_i|}_{>0} \ f_2$. $\square$

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

In this case we must ensure that the μ^K_s-measure of the set of paths σ in $paths^K_s$ for which $\sigma \models_K f_1\ U^{\leq t}\ f_2$ is 1. The algorithm *LABEL_AU* is defined as follows:

Algorithm 5. (*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 := $\{s \mid s \in \text{unseen} \ \wedge\ \forall s' : (\mathcal{T}(s,s') > 0 \rightarrow s' \in \text{seen})\}$;
```

Intuitively, *seen* are the states that have already been labelled. The other variables have analogous intuitive meanings as in algorithm *LABEL_EU*. After passing the for loop with index i, the algorithm will have labelled all states that satisfy $f_1\ U^{\leq i}_{\geq 1}\ f_2$.

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

Intuitively, the states for which $f_1\ U^{\leq \infty}_{\geq 1}\ f_2$ holds are the states from which there is a 0 probability of eventually reaching a state in Q (as defined in section 4.1). These states are labelled by the following algorithm:

Algorithm 6. (*LABEL_AU$^\infty$*)
```
    unseen := $S_i$;
    fringe := $Q$;
    failure := $\emptyset$;
    repeat
        unseen := unseen \ fringe;
        failure:= failure $\cup$ fringe;
        fringe := $\{s \mid s \in \text{unseen} \ \wedge\ \exists s' \in \text{failure} : \mathcal{T}(s,s') > 0\}$
    until fringe = $\emptyset$;
    $\forall s \in (S \setminus$ failure) do addlabel(s,f);
```

Intuitively, *failure* are the states for which it has been established that $f_1\ U^{\leq \infty}_{\geq 1}\ f_2$ does not hold. The other variables have analogous intuitive meanings as in algorithm *LABEL_EU*. The algorithm terminates when no more failure-states can be identified, i.e., after at most $|S_i|$ iterations. The states which are not identified as failure states are then labelled with $f_1\ U^{\leq \infty}_{\geq 1}\ f_2$.

5. Example

In this section we present a simple example to illustrate the proposed method. We will verify that a soft deadline is met by a communication protocol. The protocol, called Parrow's Protocol (PP) [Par85], is a simplified version of the well known Alternating Bit Protocol [BSW69]. By retransmitting lost messages, 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

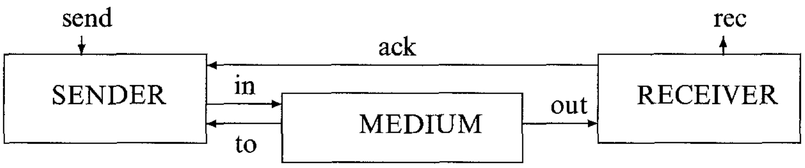

Fig. 2. The components of Parrow's Protocol.

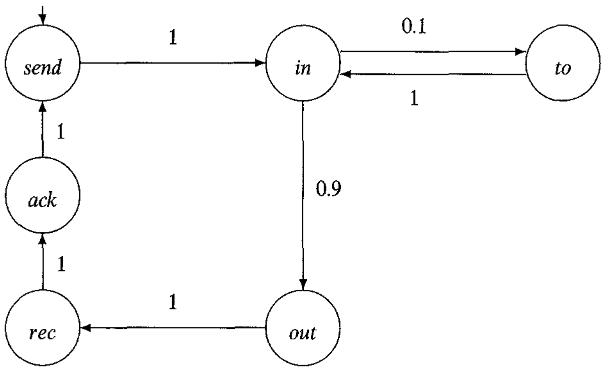

Fig. 3. The behaviour of PP.

and their interactions are described in Fig. 2. The structure in Fig. 3 presents the behaviour of PP. It is assumed that 10% of the messages are lost.

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:

$$f \;=\; \mathit{send} \underset{\geq 0.99}{\overset{\leq 5}{\leadsto}} \mathit{rec}$$

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 expanded to allow model checking:

$$f \;=\; \underbrace{\underbrace{\left[\underbrace{\mathit{send}}_{f_1} \to \underbrace{\left(\underbrace{\mathit{true}}_{f_2}\; U^{\leq 5}_{\geq 0.99}\; \underbrace{\mathit{rec}}_{f_3}\right)}_{f_5}\right]}_{f_6} \;\mathcal{U}^{\leq\infty}_{\geq 1}\; \underbrace{\mathit{false}}_{f_4}}_{f}$$

Table 2. Successive calculations using algorithm 1

time:	0	1	2	3	4	5
state						
ack	0	0	0	0	0.9	0.9
send	0	0	0	0.9	0.9	0.99
to	0	0	0.9	0.9	0.99	0.99
in	0	0.9	0.9	0.99	0.99	0.999
out	0	1	1	1	1	1
rec	1	1	1	1	1	1

The labelling of states starts with the smallest subformulas, i.e. f_1, f_2, f_3 and f_4. The state *send* will be labelled with f_1, all states will be labelled with f_2, state *rec* will be labelled with f_3, and no state will be labelled with f_4. We will illustrate labelling of states with f_5 using both algorithms for calculation of $\mathscr{P}(t,s)$ presented in section 4.

Algorithm 1: We illustrate the labelling of states with f_5 using algorithm 1 in section 4.1. The algorithm assumes that states have been labelled with f_2 and f_3. Table 2 shows the result of the successive calculations, performed from left (time=0) to right (time=5). We can conclude that all states except *ack* should be labelled with f_5, since after 5 time units $p \geq 0.99$ for all other states.

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

$$M = \begin{array}{c} \\ ack \\ send \\ to \\ in \\ out \\ rec \end{array} \begin{array}{c} \begin{array}{cccccc} ack & send & to & in & out & rec \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0.1 & 0 & 0.9 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \end{array} \qquad \overline{\mathscr{P}}(0) = \begin{array}{c} ack \\ send \\ to \\ in \\ out \\ rec \end{array} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

The next step is to calculate $\overline{\mathscr{P}}(5)$:

$$\overline{\mathscr{P}}(5) = M^5 \times \overline{\mathscr{P}}(0) = \begin{array}{c} ack \\ send \\ to \\ in \\ out \\ rec \end{array} \begin{pmatrix} 0.9 \\ 0.99 \\ 0.99 \\ 0.999 \\ 1 \\ 1 \end{pmatrix}$$

We conclude that all states except *ack* should be labelled with f_5, since $\overline{\mathscr{P}}(5)$ gives a probability greater than 0.99 for all these states. Not surprisingly, the probabilities in the vector $\overline{\mathscr{P}}(5)$ are exactly the same as the probabilities after 5 time units obtained with algorithm 1.

Next, we will label all states with f_6, since the *send* state is labelled with f_5. The labelling of states with f can be done via the dual formula (as defined in section 2)

$$\neg\left(\neg \mathit{false}\ U^{\leq\infty}_{>0}\ (\neg f_6\ \wedge\ \neg \mathit{false})\right).$$

Note that the labelling procedure in this last step is very naive. It can be drastically

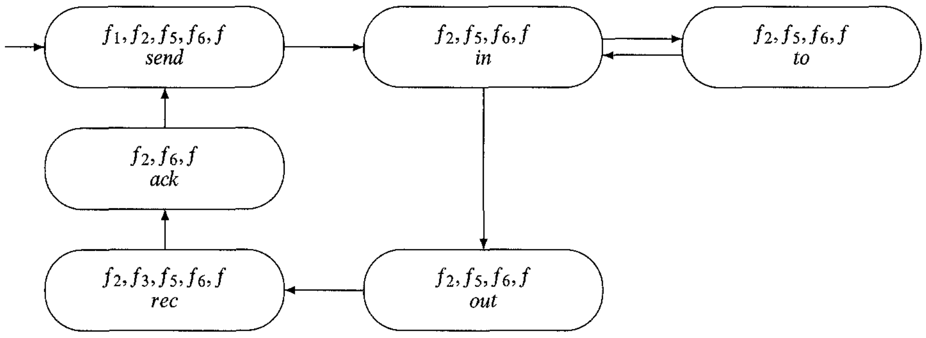

Fig. 4. The resulting labelled structure.

simplified by using a special algorithm for labelling states with formulas of the form AGf'. Such an algorithm is straightforward to construct, since all states should be labelled with AGf' if all states are labelled with f'. Hence, in our case all states should be labelled with f, since all states are labelled with f_6.

The labelled structure is shown in Fig. 4. We can conclude that f holds for the structure, since the initial state (*send*) is labelled with f.

6. Related Work

6.1. Performance Analysis

One of the most used tools for performance analysis is *Petri Nets* extended with time. There are different categories of these nets, e.g, *Timed Petri Nets* (TPNs) and *Stochastic Petri Nets* (SPNs). TPNs were introduced by Zuberek [Zub85] and extended by Razouk and Phelps [Raz84, RaP84]. The TPN model is based on Petri nets and associates firing frequencies and deterministic firing times with each transition in the net. SPN were introduced by Molloy [Mol82] and extended by Marsan, Balbo, and Conte [ABC86]. In the SPN model a stochastic firing time is associated to transitions. TPNs and SPNs are mainly used to calculate performance measures of computer system designs. That is, the system is assumed given together with the 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 faithful as possible to actual systems while retaining the possibility of analysis.

The key steps in analysising TPNs and SPNs are:

1. Model the system as a Petri-net (TPN or SPN).
2. Generate a finite-state Markov chain or Markov process from the net.
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 utilisation of resources such as memory, buses, etc. Waiting times can be analyzed by looking at the fraction of time spent in waiting states.

TPN and SPN analysis is mainly used for tuning the behaviour of system 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 [HoV87a] and cache protocols [VeH86]. There are several software packages available that help in the analysis for these models, e.g. [HoV86, Chi85, CMT89, SaM86].

TPNs and SPNs are related to our approach in that our models are essentially Markov Chains. The use of deterministic time and probabilities makes our model more closely related to the TPN than the SPN models. The main difference between the TPN approach and ours is the class of properties that are analyzed for Markov Chains. We have focussed our attention on soft deadlines, while TPN analysis mainly deals with steady state analysis.

6.2. Logics for Real Time

Many of the logics employed to state properties of concurrent programmes are various forms of modal logics [Pnu82, Abr80], the most common ones being forms of temporal logic. Many of these are suitable for reasoning about how events or predicates may be ordered in time, without bothering about time quantities. The logic we use is inspired by a one such logic, CTL [CES86]. CTL has a polynomial time model-checking algorithm and an exponential time satisfiability algorithm [EmC82].

Emerson, Mok, Sistla, and Srinivasan [EMS92] extend CTL to deal with quantitative (discrete) time. Examples of properties expressible in their 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. As in PCTL, one time unit is associated to each transition. In [Eme92], Emerson generalizes the results in [EMS92] by defining a quantitative version of the μ-calculus. Alur, Courcoubetis, and Dill [ACD90] extend CTL in a way similar to RTCTL, but in their logic (TCTL) formulas are interpreted over models with continuous time. Alur, Courcoubetis, and Dill have also developed automatic techniques for verification of probabilistic real-time processes. In [ACD91] they present an algorithm for checking whether a semi-Markov process satisfy a formula in TCTL, and [ACD92] presents a corresponding algorithm for properties given as deterministic timed automata [AlD90].

An early reference to work on extending modal logics with quantitative time is by Bernstein and Harter [BeH81]. They extend traditional linear time temporal logic with quantitative time. A related logic is presented in [KVR83]. Timed extensions of linear time temporal logic have also been defined by Ostroff [Ost89], Pnueli and Harel [PnH88], and Alur and Henzinger [AlH89]. Alur and Henzinger have also written a nice survey of logics for real-time [AlH92].

The Real-Time Logic (RTL) of Jahanian and Mok [JaM86] 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. Such a decision procedure is however highly inefficient. Jahanian and Mok therefore develop algorithms for checking if restricted sub-classes of finite-state processes satisfy safety specifications [JaM87]. Hooman [Hoo91] uses another first order logic to formulate real-time assertions (pre and post conditions) of real-time programmes, and gives a compositional proof system for verification of Occam-like programmes.

6.3. Probabilistic Logics

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 literature of modal logics that are extended with probabilities (but not time), e.g., PTL by Hart and Sharir [HaS84], and TC by Lehman and Shelah [LeS82]. 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 [PnZ86], and finite-state model-checking versions [Var85, Fel83, HSP83, HaS84, VaW86].

Courcoubetis and Yannakakis [CoY88, CoY89] have investigated the complexity of model-checking for linear time propositional temporal logic of sequential and concurrent probabilistic programmes. In the sequential case, the models are (just as our models) Markov chains. They give a model-checking algorithm that runs in time linear in the programme 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 programme satisfies its specification.

Larsen and Skou [LeS89] define a probabilistic version of Hennessy-Milner Logic, interpreted over probabilistic labelled transition systems. Christoff and Christoff [ChC92b] define three recursive logics and corresponding model checking algorithms with which properties of probabilistic labelled transition systems can be specified and verified.

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 Markov chains 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, one does not usually analyze the transient behaviour as we do. Our analysis can thus be seen as a complement to the mean-time analysis for TPNs.

To facilitate modeling of concurrent systems we have developed a structured specification language (a process algebra) [HaJ90]. The language, TPCCS, is an extension of Milner's CCS [Mil89] with quantitative time and probabilities. The TPCCS-models are "concurrent Markov chains" [Var85] in which some transitions are non-deterministic and others probabilistic. Based on PCTL, we develop a new modal logic (TPCTL) in [Han91], and define a model checking algorithm for deciding if a TPCCS-process satisfies a given TPCTL-formula.

Acknowledgements

We are grateful to Ivan Christoff, Linda Christoff, Jozef Hooman, Fredrik Orava, and Parosh for reading and discussing drafts of this manuscript.

References

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

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

[ACD90] Alur, R., Courcoubetis, C. and Dill, D.: Model-checking for real-time systems. In *Proc. 5th IEEE Int. Symp. on Logic in Computer Science*, pages 414–425, 1990.

[ACD91] Alur, R., Courcoubetis, C. and Dill, D.: Model-checking for probabilistic real-time systems. In *Proc. 18th Int. Coll. on Automata Languages and Programming (ICALP)*, volume 510 of *Lecture Notes in Computer Science*, pages 115–126. Springer Verlag, 1991.

[ACD92] Alur, R., Courcoubetis, C. and Dill, D.: Verifying Automata Specifications of Probabilistic Real-Time Systems. In J. de Bakker, C. Huizing, W.-P. de Roever, and G. Rozenberg, editors, *Real-Time: Theory in Practice*, volume 600 of *Lecture Notes in Computer Science*, pages 28–44. Springer Verlag, 1992.

[AlD90] Alur, R. and Dill, D.: Automata for modeling real-time systems. In *Proc. 17th Int. Coll. on Automata Languages and Programming (ICALP)*, volume 443 of *Lecture Notes in Computer Science*, Springer Verlag, 1990.

[AlH89] Alur, R. and Henzinger, T.: A really temporal logic. In *Proc. 30th IEEE Annual Symp. Foundations of Computer Science*, pages 164–169, 1989.

[AlH92] Alur, R. and Henzinger, T.: Logics and Models of Real Time: A Survey. In J. de Bakker, C. Huizing, W.-P. de Roever, and G. Rozenberg, editors, *Real-Time: Theory in Practice*, volume 600 of *Lecture Notes in Computer Science*, pages 28–44. Springer Verlag, 1992.

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

[BeH81] Bernstein, A. and Harter, P.K.: Proving real-time properties of programs with temporal logic. In *Proc. 8th ACM Symp. on Operating System Principles*, pages 1–11, Pacific Grove, California, 1981.

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

[ChC92b] Christoff, L. and Christoff, I.: Reasoning about safety and liveness properties for probabilistic processes. In R. Shyamasundar, editor, *Proc. 12th Conf. on Foundations of Software Technology and Theoretical Computer Science*, volume 652 of *Lecture Notes in Computer Science*, pages 342–355. Springer-Verlag, 1992.

[CES86] Clarke, E.M., Emerson, E.A. and Sistla, A.P.: Automatic verification of finite-state concurrent systems using temporal logic specification. *ACM Trans. on Programming Languages and Systems*, 8(2):244–263, April 1986.

[Chi85] Chiola, G.: 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.

[CMT89] Ciardo, G., Muppala, J. and Trivedi, K.S.: Spnp: Stochastic petri net package. In *Proc. of the third International Workshop on Petri Nets and Performance Models.* IEEE Computer Society Press, Kyoto, Japan, December 1989.

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

[CoY88] Courcoubetis, C. and Yannakakis, C.: The complexity of probabilistic verification. In *Proc. 29th IEEE Annual Symp. Foundations of Computer Science*, pages 338–345, 1988.

[CoY89] Courcoubetis, C. and Yannakakis, C.: The complexity of probabilistic verification. Bell labs Murry Hill, 1989.

[dBH92] de Bakker, J., Huizing, C., de Roever, W.-P. and Rozenberg, G.: editors. *Real-Time: Theory in Practice*, volume 600 of *Lecture Notes in Computer Science.* Springer Verlag, 1992.

[EmC82] Emerson, E.A. and Clarke, E.M.: Using branching time Temporal Logic to synthesize synchronization skeletons. *Science of Computer Programming*, 2(3):241–266, 1982.

[Eme92] Emerson, A.: Real-Time and the Mu-Calculus. In J. de Bakker, C. Huizing, W.-P. de Roever, and G. Rozenberg, editors, *Real-Time: Theory in Practice*, volume 600 of *Lecture Notes in Computer Science*, pages 176–194. Springer Verlag, 1992.

[EMS92] Emerson, A., Mok, A., Sistla, A. and Srinivasan, J.: Quantitative temporal reasoning. *Real-Time Systems - The International Journal of Time-Critical Computing Systems*, 4:331–352, 1992.

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

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

[Han91] Hansson, H.: *Time and Probabilities in Formal Design of Distributed Systems.* PhD thesis, Department of Computer Systems, Uppsala University, 1991. Available as report DoCS 91/27, Department of Computer Systems, Uppsala University, Sweden, and as report 05 in SICS dissertation series, SICS, Kista, Sweden. A revised version of the thesis will appear in the Elsevier book series Real-Time Safety Critical Systems.

[HaJ90] Hansson, H. and Jonsson, B.: A calculus for communicating systems with time and probabilities. In *Proc. 11th IEEE Real -Time Systems Symp.*, pages 278–287, Orlando, Fl., December 1990. IEEE Computer Society Press.

[Hoo91] Hooman, J.: *Specification and Compositional Verification of Real-Time Systems*, volume 558 of *Lecture Notes in Computer Science.* North-Holland, 1991.

[HaS84] Hart, S. and Sharir, M.: Probabilistic temporal logics for finite and bounded models. In *Proc. 16th ACM Symp. on Theory of Computing*, pages 1–13, 1984.

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

[HoV86] Holliday, M.A. and Vernon, M.K.: The GTPN Analyzer: numerical methods and user interface. Technical Report 639, Dept. of Computer Science, Univ. of Wisconsin – Madison, Apr. 1986.

[HoV87a] Holliday, M.A. and Vernon, M.K.: Exact performance estimates for multiprocessor memory and bus interface. *IEEE Trans. on Computers*, C-36:76–85, Jan. 1987.

[HoV87b] Holliday, M.A. and Vernon, M.K.: A generalized timed Petri net model for performance analysis. *IEEE Trans. on Software Engineering*, SE-13(12), 1987.

[JaM86] Jahanian, F. and Mok, K.-L.: Safety analysis of timing properties in real-time systems. *IEEE Trans. on Software Engineering*, SE-12(9):890–904, Sept. 1986.

[JaM87] Jahanian, F. and Mok, A.K.: A graph-theoretic approach for timing analysis and its implementation. *IEEE Trans. on Computers*, 36(8):961–975, August 1987.

[Jos88] Joseph, M.: editor. *Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume 331 of *Lecture Notes in Computer Science.* Springer Verlag, 1988.

[KVR83] Koymans, R., Vytopil, J. and de Roever, W.P.: Real-time programming and asynchronous message passing. In *Proc. 2nd ACM Symp. on Principles of Distributed Computing*, pages 187–197, Montréal, Canada, 1983.

[LeS82] Lehmann, D. and Shelah, S.: Reasoning with time and chance. *Information and Control*, 53:165–198, 1982.

[LeS89] Larsen, K.G. and Skou, A.: Bisimulation through probabilistic testing. In *Proc. 16th ACM Symp. on Principles of Programming Languages*, pages 344–352, 1989.

[Mil89] Milner, R.: *Communication and Concurrency.* Prentice-Hall, 1989.

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

[OwL82] Owicki, S. and Lamport, L.: Proving liveness properties of concurrent programs. *ACM Trans. on Programming Languages and Systems*, 4(3):455–495, 1982.

[Ost89] Ostroff, J.: Automatic verification of timed transition models. In Sifakis, editor, *Workshop on automatic verification methods for finite state systems*, volume 407 of *Lecture Notes in Computer Science*, pages 247–256. Springer Verlag, 1989.

[OsW87] Ostroff, J. and Wonham, W.: Modelling, specifying and verifying real-time embedded computer systems. In *Proc. IEEE Real-time Systems Symp.*, pages 124–132, Dec. 1987.

[Par85] Parrow, J.: *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.

[PnH88] Pnueli, A. and Harel, E.: 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*, volume 331 of *Lecture Notes in Computer Science*, pages 84–98. Springer Verlag, 1988.

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

[PnZ86] Pnueli, A. and Zuck, L.: Verification of multiprocess probabilistic protocols. *Distributed Computing*, 1(1):53–72, 1986.

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

[RaP84] Razouk, R.R. and Phelps, C.V.: 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.

[ShL87] Shankar, A.U. and Lam, S.S.: Time dependent distributed systems: Proving safety, liveness and real-time properties. *Distributed Computing*, 2:61–79, 1987.

[SaM86] Sanders, W.H. and Meyer, J.F.: Metasan: a performability evaluation tool based on stochastic activity networks. In *Proc of the ACM-IEEE Comp. Soc. Fall Joint Conf.* IEEE Computer Society Press, November 1986.

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

[VeH86] Vernon, M.K. and Holliday, M.A.: Performance analysis of multiprocessor cache consistency 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.

[VaW86] Vardi, M.Y. and Wolper, P.: An automata-theoretic approach to automatic program verification. In *Proc. IEEE Symp. on Logic in Computer Science*, pages 332–344, June 1986.

[Vyt91] Vytopil, P.: editor. *Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume 571 of *Lecture Notes in Computer Science.* Springer Verlag, 1991.

[Zub85] Zuberek, W.: Performance evaluation using extended timed Petri nets. In *Proc. International Workshop on Timed Petri Nets*, pages 272–278, Torino Italy, 1985. IEEE Computer Society Press.

APPENDIX

Algorithms for Labelling States with $f_1 \ \mathcal{U}^{\leq t}_{\geq p} \ f_2$.

In this section we present modifications of algorithms 1 and 2 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^K_s-measure for the set of paths σ in $paths^K_s$ 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:

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

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

Algorithm 7.
```
for $i := 0$ to $t$ do
  for all $s \in S$ do
    if $f_2 \in label(s)$
    then $\mathcal{R}(i,s) := 1$
    else
      $\mathcal{R}(i,s) := 0$;
      if $f_1 \in label(s)$ then for all $s' \in S$ do
        $\mathcal{R}(i,s) := \mathcal{R}(i,s) + \mathcal{T}(s,s') \times \mathcal{R}(i-1,s')$
```

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

Analogously to algorithm 7 above we can define an algorithm for the Unless case that corresponds to algorithm 2.

Algorithm 8.
```
for all $s \in S$ do
    if $f_2 \in label(s)$ or $f_1 \in label(s)$
    then $\overline{\mathcal{R}}(0)[s] := 1$
    else $\overline{\mathcal{R}}(0)[s] := 0$;
$\overline{\mathcal{R}}(t) = M^t \times \overline{\mathcal{R}}(0)$
```

Algorithm for Labelling States with $f_1 \mathcal{U}^{\leq t}_{\geq 0} f_2$.

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.

Algorithm 9. (*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 $\setminus$ fringe;
    fringe :=
    $\{s \mid s \in$ unseen $\wedge \forall s' : (\mathcal{T}(s,s') > 0 \rightarrow s' \in$ bad$)\}$;
$\forall s \in S \setminus$ 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.

Algorithm for Labelling States with $f_1 \mathcal{U}^{\leq t}_{\geq 1} f_2$.

The algorithm *LABEL_AUnless* labels states, s, for which $s \models_K f_1 \mathcal{U}^{\leq t}_{\geq 1} f_2$ with $f_1 \mathcal{U}^{\leq t}_{\geq 1} f_2$. 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.

Algorithm 10. (*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 $\setminus$ fringe;
    fringe :=
    $\{s \mid s \in$ unseen $\wedge \exists s' : (\mathcal{T}(s,s') > 0 \rightarrow s' \in$ fringe$)\}$;
$\forall s \in S \setminus$ 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.

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.

Algorithm 11. (*Identify_R*)

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

Received November 1990
Accepted in revised form October 1993 by J.V. Tucker