"Almost always" and "definitely sometime" are not enough: Probabilistic quantifiers and Probabilistic model-checking[1]

Purush Iyer and Murali Narasimha
Dept of Computer Science
North Carolina State University
Raleigh, NC 27695-8206
e-mail: {purush, mnarasi}@eos.ncsu.edu
Phone: +1 (919)-515-7291
Fax: +1 (919)-515-7896 NC State University

Tech Report
TR-96-16

Abstract

Specifications for probabilistic programs often use the notion of *almost always* and *definitely sometime* to capture the probabilistic information. But there are a number of instances (eg. network protocols) where probabilistic information needs to be explicitly specified. In this paper we present PCTL*, a probabilistic version of the branching time logic CTL*, where the quantifiers for universality (almost always) and existence (definitely sometime) are replaced by a single probabilistic quantifier. We argue that this logic is more general than previous logics. We show how to model-check probabilistic programs (given as markov chains) against specifications in PCTL*. We also prove our algorithm correct and provide an analysis of its computational complexity.

[1]Supported in part by grant under NSF CCR-9404619

1 Introduction

Over the past decade and a half model-checking has been used successfully as a verification technique for hardware designs [6], network protocol designs [11] and, more recently, software designs (i.e., requirement specifications) [4, 13]. Model-checking can be characterized as the use of semantic techniques to answer the question "Does a (finite-state) program P satisfy a (temporal) logic specification S?"

The context of our current work is model-checking of finite-state probabilistic programs against specifications in a branching time logic which combines probabilistic and temporal information. The motivation for our work comes from analysis of network protocols where failure is modeled. For instance, assume we have a protocol whose functionality can be stated as follows: every message sent by the sender is eventually received by the receiver. But under assumptions of communication failure, the specifications should probably read as follows:

> if the failure rate of transmission is less than 5% then the probability that every message sent by the sender will reach the receiver in less than 15 steps (i.e., clock ticks) is greater than 95%.

Such a specification allows one to quantify the rate of failure and the rate of success, in addition to specifying the required sequence of events of a system. Clearly, such specifications are needed to validate communication protocols and real-time programs, today. It is conceivable that, in the future, such service specifications would be required of net-based programs.

The main thesis of this paper is that *almost always* and *definitely sometime* are **not** general enough notions for probabilistic reasoning, and that a more general notion of *probabilistic quantifier* is appropriate. We enrich CTL* [8] – a branching time logic – with a (family of) probabilistic quantifier(s) ($Æ_p\psi$ denoting that ψ holds with probability at least p) to obtain a new logic PCTL*. We provide a model-checking algorithm to check whether a finite state markov chain satisfies a specification in this logic, show that it is correct and provide the computational complexity of our algorithm. Our contributions are, therefore, a branching time logic of chance and an associated algorithm for model-checking markov chains (i.e., probabilistic programs).

1.1 Related work

Most of the earlier efforts in probabilistic model-checking can be divided into two classes: those which dealt with linear time propositional temporal logic [10, 16, 7] and those which dealt with branching time temporal logics [3, 9]. In both of these cases the traditional (i.e., non-probabilistic) logic was used. In the case of linear logic, model-checking boil downs to the following problem:

> Given a markov chain $\mathcal{M}$ and a LTL formula ψ do the set of sequences from $\mathcal{M}$ which satisfy ϕ have the measure one.

Consequently, the traditional model-checking problem is replaced by one where the question is whether a property holds almost always. But it turns out that the algorithm for probabilistic model-checking problem can be reduced to the algorithm for traditional model-checking problem,

using zero-one laws. In effect, the notion of "almost always" is impervious to the probabilities adorning the edges of a markov chain, as long as they are non-zero.

In the work of [3] the branching time logic CTL* is considered for specification. The important difference between LTL and CTL* is the presence of path quantifiers $\forall$ and $\exists$. In this context the universal quantifier is interpreted as almost always and the existential quantifier as *at least somewhere*. Our work differs from [3] in that we use a single probabilistic quantifier. This not only allows us to talk about specific probabilities but is also general enough to capture the semantics of the two quantifiers in Alur *et al*'s work.

The work that is most closely related to ours is presented in [9, 5, 12]. In [9] Hansson considers a logic TPCTL which combines both explicit probabilities and real time. Given his program model, Hansson finds use for both the universal and existential path quantifiers. His program model, however, differs from ours in that it differentiates between non-determinism and randomness; where as we believe that markov chains are a more natural model for specifying reliability information.

In [5] Aziz *et al* consider a logic very similar to ours – a branching time logic based on CTL, but where the two quantifiers are supplanted by a single probabilistic quantifier. Their work differs from ours in that they consider continuous time markov chains as models. This makes for an interesting decidability result based on properties of exponential distributions. Their work differs from this paper in that we consider discrete markov chains, and hence the algorithms and the proof techniques are completely different.

Finally, in [12] we consider the problem of model-checking a class of infinite state markov chains, which arise from finite state machines communicating over lossy FIFO buffers, against a linear time logic. The main emphasis in that paper was on reducing model-checking infinite state systems to model-checking (a sequence of) finite state systems. While the current paper grew out of our efforts to extend that work to branching-time logics, we believe, it is of independent importance.

1.2 Outline

In Section 2 we provide the necessary definitions for markov chains and the logic. We also provide some examples to illustrate the use of the logic. In Section 3 we develop the model-checking algorithm, show its correctness and develop its computation complexity. Finally, our conclusion is in Section 4.

2 Definitions

We take markov chains as our notion of programs. Formally:

Definition 2.1 [Markov chain] A Markov chain $\mathcal{M}$ is a tuple (Q, δ, Σ, I) where

- $(q, q_1, q_2 \in) Q$ is a countable set of states,
- $\delta \subseteq Q \times Q \rightarrow [0, 1]$ is the transition relation,
- $(\sigma \in) \Sigma$ is a set of atomic propositions,
- $I : Q \rightarrow 2^{\Sigma}$ is the labeling function.

Given a markov chain $\mathcal{M}$ and a designated *start state* q_s of $\mathcal{M}$ we can define a sequence space [14] $\wp(\mathcal{M}, q_s) = (\Omega^{q_s}, \mathcal{F}^{q_s}, \mu^{q_s})$ where

- $\Omega^{q_s} = Q^\omega$ is the set of all infinite sequences of states of $\mathcal{M}$ starting at q_s,
- $\mathcal{F}^{q_s} : Q^* \rightarrow 2^{Q^\omega}$ is a Borel field generated by the *basic cylindric sets*

$$\mathcal{F}^{q_s}(q_0 q_1 \ldots q_n) = \{\pi \in \Omega^{q_s} | \pi = q_0 q_1 \ldots q_n \ldots\},$$

- μ^{q_s} is a probability measure function defined by

$$\mu^{q_s}(\mathcal{F}(q_0 q_1 \ldots q_n)) = p_1 \times p_2 \times \ldots \times p_n$$

  where $p_i = \delta(q_{i-1}, q_i)$.

When the markov chain $\mathcal{M}$ is understood, we will use μ^q to refer to the measure function of the sequence space $\wp(\mathcal{M}, q)$ without actually mentioning the sequence space.

For specifications we will use PCTL*, a branching time logic, which allows us to express both temporal and probabilistic information. To that end, we first define the syntax and semantics of LTL, a linear time logic, and a sub-logic of PCTL*.

2.1 LTL

The syntax of LTL is inductively defined as follows:

Definition 2.2 [LTL syntax] Given a set Σ of atomic propositions an LTL formula is built from elements of Σ using the boolean operators $\wedge$ and $\neg$, the unary temporal connective X (*next*) and the binary temporal connective $\mathcal{U}$ (*until*).

We define the semantics of LTL (and PCTL*) formulae with respect to a markov chain $\mathcal{M} = (Q, \delta, \Sigma, I)$. Define a *computation* of $\mathcal{M}$ to be an infinite sequence of states $q_0 q_1, \ldots$ such that $\forall i : \delta(q_i, q_{i+1}) > 0$. We will use π, π_1 etc to denote computations. We will just use a sequence of states enclosed in parentheses to refer to a valid computation as in $(q_0 q_1 \ldots)$. Given a computation π let π^i denote the i-th suffix of a computation i.e. if $\pi = (q_0 q_1 \ldots)$ then $\pi^i = (q_i q_{i+1} \ldots)$. Also, let $\pi(i)$ denote the i-th state of π.

The satisfaction of an LTL formula with respect to a markov chain $\mathcal{M} = (Q, \delta, \Sigma, I)$ is given by a relation $\models_{\mathcal{M}}$, defined inductively as follows:.

- $\pi \models_{\mathcal{M}} \sigma$ iff $\sigma \in I(\pi(0))$.
- $\pi \models_{\mathcal{M}} \psi_1 \wedge \psi_2$ iff $\pi \models_{\mathcal{M}} \psi_1$ and $\pi \models_{\mathcal{M}} \psi_2$.
- $\pi \models_{\mathcal{M}} \neg\psi$ iff $\neg(\pi \models_{\mathcal{M}} \psi)$.
- $\pi \models_{\mathcal{M}} X\psi$ iff $\pi^1 \models_{\mathcal{M}} \psi$.
- $\pi \models_{\mathcal{M}} \psi_1 \mathcal{U} \psi_2$ iff there exists $n \geq 0$ such that $\pi^n \models_{\mathcal{M}} \psi_2$ and for each $i < n$, $\pi^i \models_{\mathcal{M}} \psi_1$.

For the model-checking algorithm we shall make use of an automata theoretic characterization of LTL formulae. Given an LTL formula ψ one can associate Büchi automata A_ψ on infinite words which captures exactly those infinite sequences over Σ which satisfy the formula ψ. Formally, we have:

Definition 2.3 A *Büchi automaton* is a tuple $A = (\tau, s_0, F)$ where:

- $\tau = (\Sigma, S, \Delta)$ is a table where $(s \in)S$ is a set of states, Σ is the alphabet and $\Delta : S \times \Sigma \rightarrow 2^S$ is the transition function,
- $F \subseteq S$ is a set of accepting states.

Given an infinite word $\eta = \sigma_0\sigma_1, \ldots \in \Sigma^\omega$ a run of τ on η is an infinite sequence of states $r = s_0 s_1 \ldots$ such that $s_{i+1} \in \Delta(s_i, \sigma_i)$ for $i \geq 0$. Let $inf(r)$ be the set of states that appear infinitely often in r. A accepts an infinite word η iff there exists a run r of τ on η, such that $inf(r) \cap F \neq \emptyset$. Given a *linear temporal logic* formula ψ, it is possible to construct a Büchi automaton A_ψ such that A_ψ accepts a word $I(\pi)$ exactly when a computation π of some Markov chain (with labeling function I) satisfies ϕ[2, 17].

2.2 PCTL*

We now present the syntax and semantics of our probabilistic branching temporal logic PCTL*.

Definition 2.4 [PCTL*] A PCTL* formula can be either a state formula or a path formula. Given a set of atomic propositions Σ, the syntax of state and path formulae is defined as follows:

- Every atomic proposition $\sigma \in \Sigma$ is a state formula.
- If ϕ_1 and ϕ_2 are state formulae then $\phi_1 \wedge \phi_2$ and $\neg\phi_1$ are state formulae.
- If ϕ is a state formula then it is also a path formula.
- If ψ is a path formula and $p \in (0, 1]$ then $Æ_p\psi$ is a state formula.
- If ψ_1 and ψ_2 are path formulae then $\psi_1 \wedge \psi_2$ and $\neg\psi_1$ are path formulae.
- If ψ_1 and ψ_2 are path formulae then $X\psi_1$ and $\psi_1 \mathcal{U} \psi_2$ are path formulae.

In the following we will use ϕ and ψ to denote a typical state and path formula, respectively.

Note that LTL formulae are properly included in the set of path formulae. The inclusion is proper as a path formula can have nested PCTL* formulae in it, while LTL formulae can not. We can also, similarly, define PCTL as a sublogic of PCTL*, much in the tradition of CTL and CTL* [8]. PCTL is obtained by imposing the restriction on PCTL* formulae that every *path modality* (X and $\mathcal{U}$) should be immediately enclosed in a *path quantifier* ($Æ_p$). For example, given some $p > 0$, $Æ_p[\sigma_1 \mathcal{U} [\sigma_2 \mathcal{U} \sigma_3]]$ and $Æ_p[X\, X\sigma]$ are not valid PCTL formulae but are valid PCTL* formulae. Formally, PCTL formulae are those that can be derived from the first two rules for PCTL* and the following rule:

If ϕ_1 and ϕ_2 are PCTL formulae then $Æ_p X\phi_1$ and $Æ_p \phi_1 \mathcal{U} \phi_2$ are also PCTL formulae.

Given a Markov chain $\mathcal{M}$ we define a relation $\models_{\mathcal{M}}$, where $q \models_{\mathcal{M}} \phi$ means that the state formula ϕ holds at state q, and $\pi \models_{\mathcal{M}} \psi$ indicates that the path formula ψ holds of computation π. The relation $\models_{\mathcal{M}}$ is defined inductively as follows:

- $q \models_{\mathcal{M}} \sigma$ iff $\sigma \in I(q)$.
- $q \models_{\mathcal{M}} \neg\phi$ iff $\neg(q \models_{\mathcal{M}} \phi)$.
- $q \models_{\mathcal{M}} \phi_1 \wedge \phi_2$ iff $q \models_{\mathcal{M}} \phi_1$ and $q \models_{\mathcal{M}} \phi_2$.
- $q \models_{\mathcal{M}} Æ_p\psi$ iff $\mu^q(\{\pi | \pi = (qq_1 \ldots) \wedge \pi \models_{\mathcal{M}} \psi\}) \geq p$.
- $\pi \models_{\mathcal{M}} \phi$ where ϕ is a state formula, iff $\pi = (q_0 q_1 \ldots)$ and $q_0 \models_{\mathcal{M}} \phi$.
- $\pi \models_{\mathcal{M}} \psi_1 \wedge \psi_2$ iff $\pi \models_{\mathcal{M}} \psi_1$ and $\pi \models_{\mathcal{M}} \psi_2$.
- $\pi \models_{\mathcal{M}} \neg\psi_1$ iff $\neg(\pi \models_{\mathcal{M}} \psi_1)$.
- $\pi \models_{\mathcal{M}} X\psi$ iff $\pi = (q_0 q_1 \ldots)$ and $\pi^1 \models_{\mathcal{M}} \psi$.
- $\pi \models_{\mathcal{M}} \psi_1 \mathcal{U} \psi_2$ iff there exists $n \geq 0$ such that $\pi^n \models_{\mathcal{M}} \psi_2$ and for each $i < n$, $\pi^i \models_{\mathcal{M}} \psi_1$.

As is customary, we will use abbreviations for disjunction $\phi_1 \vee \phi_2 = \neg(\neg\phi_1 \wedge \neg\phi_2)$, implication $\phi_1 \Rightarrow \phi_2 = \neg\phi_1 \vee \phi_2$, *global* modality $G\psi = \neg(true \mathcal{U} \neg\psi)$ and future $F\psi = (true \mathcal{U} \psi)$.

Note that the notion of "almost always" and "definitely sometime" can also be expressed as abbreviations in our logic:

$$\forall\psi = Æ_1\psi$$
$$\exists\psi = \neg Æ_1 \neg\psi$$

Finally, we will also the abbreviation $Æ_{<p}\psi = \neg Æ_{1-p}\neg\psi$ to denote that the measure of the set of path which satisfy ψ is less than p.

Example: Use ℓ to denote loss of a message, S to denote that a message was sent and R to denote that the message was received, the property "if the probability of loss is less than 5% then with 95% probability a message sent would be received within fifteen steps" can be written as:

$$(\forall G(Æ_{<0.05} X\ell))$$
$$\Rightarrow$$
$$\forall G(\mathrm{S} \Rightarrow Æ_{0.95}(\bigvee_{1 \leq i \leq 15} X^i \mathrm{R}))$$

3 Model-checking against PCTL* formulae

We have defined satisfaction of a PCTL* formula in terms of satisfaction of sub-formulae. In this section we use automata based techniques for model-checking LTL formulae as a step in building an algorithm that checks whether a given finite state Markov chain satisfies a PCTL* formula. The outline of the algorithm is as follows: First we identify the most deeply nested state formula that

Algorithm PCTL*-sat
Input: A finite Markov chain $\mathcal{M} = (Q, \delta, \Sigma, I)$, a start state $q_s (\in Q)$ of $\mathcal{M}$ and a PCTL* formula ϕ.
Output: "true" if $\mathcal{M} \models \phi$ and "false" otherwise.
var ψ-sat_p: set;

Step 1: Identify a most deeply nested (non-propositional) state formula of ϕ. Let it be θ.

case θ is of the form $\neg\sigma$:
foreach $q \in Q$ such that $\sigma \notin I(q)$ **do**
$I(q) := I(q) \cup \{\sigma_\theta\}$; (* σ_θ is a new atomic proposition *)
Replace all occurrences of θ in ϕ by σ_θ.
end; (* foreach *)
Goto Step 4.
case θ is of the form $\sigma_1 \wedge \sigma_2$:
foreach $q \in Q$ such that $\sigma_1 \in I(q) \wedge \sigma_2 \in I(q)$ **do**
$I(q) := I(q) \cup \{\sigma_\theta\}$; (* σ_θ is a new atomic proposition *)
Replace all occurrences of θ in ϕ by σ_θ.
end; (* foreach *)
Goto Step 4.
case θ is of the form $Æ_p\psi$:
Goto Step 2.

Step 2: For the LTL formula ψ construct the Büchi automaton $A_\psi = (\tau, s_0, F)$ where $\tau = (\Sigma, S, \Delta)$ is a table and F is the set of accepting states.

Step 3:

ψ-sat_p :=**sat-states**$(\mathcal{M}, A_\psi, p)$;
$\Sigma := \Sigma \cup \{\sigma_{Æ_p\psi}\}$; (* $\sigma_{Æ_p\psi}$ is a new atomic proposition *)
foreach $q \in \psi$-sat_p **do**
$I(q) := I(q) \cup \{\sigma_{Æ_p\psi}\}$;
end; (* foreach *)
In PCTL* formula ϕ replace all occurrences of $Æ_p\psi$ with atomic proposition $\sigma_{Æ_p\psi}$;

Step 4: If ϕ has not been replaced by an atomic proposition Goto Step 1.

Step 5: **if** $\sigma_\phi \in I(q_s)$ **then exit**(true) **else exit**(false);

Figure 1:

is not an atomic proposition. This could be of one of the types $\neg\sigma_1$, $\sigma_1 \wedge \sigma_2$ or $Æ_p\psi$ (where ψ is an LTL formula). Our central idea is to, recursively, find all states of the markov chain that satisfy the most deeply nested formula and replace that formula by an atomic proposition, and to extend the labeling function over the new atomic proposition. If a formula is true in the start state of the given markov chain then it will be reduced to an atomic proposition which is true of the initial state, by our algorithm.

An innermost state formula could either be a boolean combination of atomic propositions or is a formula with the probabilistic quantifier $Æ_p$. States that satisfy a boolean combination can be easily identified by checking the labeling function. Finding states that satisfy a formula with a quantifier is non-trivial and shall be discussed below. The algorithm is shown in Figure 1. Note that in Step 3 the algorithm uses **sat-states** to compute the set of states $q \in Q$ at which the innermost state formula $Æ_p\psi$ is true. In order to make the proof easy we now show that **PCTL*-sat** is correct assuming **sat-states** is correct.

Theorem 3.1

- *Under the assumption that* **sat-states** *terminates the algorithm* **PCTL*-sat** *terminates.*
- *Under the assumption that*

$$\textbf{sat-states}(\mathcal{M}, A_\psi, p) = \{q \in Q | q \models_{\mathcal{M}} Æ_p\psi\}$$

algorithm **PCTL*-sat** *is correct, i.e.,*

PCTL*-sat$(\mathcal{M}, q_s, \phi)$ *returns* **true** *iff* $q_s \models_{\mathcal{M}} \phi$

Proof: The termination follows from the fact that the size of the formula reduces in every iteration. The soundness of the algorithm can be established by an induction on the number of iterations, and its completeness by an induction on the size of the formula. ■

3.1 Satisfaction probability of an LTL formula

Given a state formula of the form $\phi = Æ_p\psi$ the algorithm **sat-states** should identify states $q \in Q$ such that ϕ is true in q. This is equivalent to checking the probability with which the LTL formula ψ holds[2] of a state q. By definition, the probability with which an LTL formula holds in a state q is the measure of all computations starting at q that satisfy the LTL formula. Since sets of computations with positive measure are composed of basic cylindric sets, it follows that we need to consider only those computations that define basic cylindric sets, and which satisfy ψ. To that end we define:

Definition 3.2 A strongly connected component (SCC) of a Markov chain is closed provided every state that is reachable from a state of the SCC is in the SCC.

The reason for considering closed SCCs is that they are the main contributors to a non-zero probability. Formally, we have the following:

[2]Since we will always be dealing the innermost state formula we are assured that ψ is an LTL formula.

Subroutine sat-states
Input: A finite Markov chain $\mathcal{M} = (Q, \delta, \Sigma, I)$, a Büchi automaton $A_\psi = (\tau, s_0, F)$ (where $\tau = (\Sigma, S, \Delta)$) and a probability $p \in (0, 1]$.
Output: The set of all states $q \in Q$ such that $\mu^q(\{\pi | \pi = q, q_1, q_2 \ldots$ is a computation $\wedge$ *inf*$(I(\pi))$ visits F infinitely often$\}) \geq p$.
var *sat-states* : set;
var *prob* : real;

Step 1: Construct new automaton
$A_{\mathcal{M},\psi} = (\tau_{\mathcal{M}}, G)$ is a table in which the start state is not specified,
$\tau_{\mathcal{M},\psi} = (\{a\}, Q \times S, \Delta_{\mathcal{M},\psi})$,
$\Delta_{\mathcal{M},\psi}((q, s), a) = \{(q', s') | \delta(q, q') > 0 \wedge s' \in \Delta(s, I(q))\}$, and
$G = Q \times F$.

Step 2: Compute the strongly connected components of the graph $A_{\mathcal{M},\psi}$.

Step 3: Collect all the closed strongly connected components $C_1, C_2, \ldots, C_n$ of $A_{\mathcal{M},\psi}$ such that

1. each C_i contains at least one final state (from G) and
2. the projection of each C_i onto $\mathcal{M}$ is a closed SCC of $\mathcal{M}$.

Step 4: Replace each closed connected component C_i from Step 3 by a new node m_i to get a new graph $A'_{\mathcal{M},\psi}$. Mark each m_i as "closed".

Step 5: For each node (q, s_0) in the new graph calculate the probability of reaching the closed nodes m_i from (q, s_0) taking care that (a) no two paths from q to m_i have the same projection in $\mathcal{M}$ and (b) none of the paths repeat any node. Let the probability be p_q for the node (q, s_0). Return $\{q \in Q | p_q \geq p\} \cup \{q \in Q | \exists i : 1 \leq i \leq n : (q, s) \in C_i\}$.

Figure 2:

Lemma 3.3

1. *The measure of the set of computations which visit infinitely often a strict subset of a closed SCC is 0.*

2. *The measure of the set of computations which visit infinitely often a non-closed SCC is 0.*

Proof: Both facts follow from the notion of *transient* state in markov chains, and the fact that the set of computations that visit a transient state has a measure 0. ■

While the previous lemma discusses what sets have a zero measure the following allows us to concentrate on the sets we are interested in, those with non-zero measure. In particular, it follows from the previous lemma that we need to concentrate on only those computations which visit all of the states of a closed SCC infinitely often. Formally,

Lemma 3.4 *Let C be a closed SCC and let α be a finite path, not in C, from the start state q_s to a state q in the SCC C. We then have the following:*

$$\mu^{q_s}(\mathcal{F}(\alpha)) = \mu^{q_s}(\{\pi | \pi = \alpha\pi',\ inf(\pi') = C \wedge \pi' \in C^{\infty}\}$$

Since we are interested in computations that satisfy a given LTL formula ψ, we define a product automaton whose computations mimic runs of the automata on the computations of $\mathcal{M}$.

Definition 3.5 [Product of Markov chain and Büchi automaton] Let $\mathcal{M} = (Q, \delta, \Sigma, I)$ be a Markov chain and $A_\psi = (\tau, s_0, F)$ (where $\tau = (\Sigma, S, \Delta)$) be a Büchi automaton for an LTL formula ψ. We define a product automaton, parameterized by a start state $q_0 \in Q$, $[\mathcal{M} \times A_\psi](q_0)$ as $(\tau_{\mathcal{M},\psi}, (q_0, s_0), G)$ as follows:

- $\tau_{\mathcal{M},\psi} = (\{a\}, Q \times S, \Delta_{\mathcal{M},\psi})$,
- $\Delta_{\mathcal{M},\psi}((q,s),a) = \{(q',s') | \delta(q,q') > 0 \wedge s' \in \Delta(s, I(q))\}$,
- $G = Q \times F$.
- The state (q_0, s_0) is the start state.

We observe the following fact, which says that computations of $\mathcal{M}$ (starting at q_0) satisfying ψ are captured by the product automaton.

Lemma 3.6 *A computation $\pi = ((q_0,s_0)(q_1,s_1),\ldots)$ of $[\mathcal{M} \times A_\psi](q_0)$ visits some state of G infinitely often iff the computation $(q_0 q_1 \ldots)$ satisfies the formula ψ.*

Define the projection of a state (q,s) onto $\mathcal{M}$ as q. Define, similarly, projection of sequences of states and SCCs. Given this definition, we have the following obvious facts about SCCs (proven easily by appealing to the definition of SCC):

Lemma 3.7

1. *Every SCC C' of $\mathcal{M} \times A_\psi$ projects to an SCC C of $\mathcal{M}$.*

2. *If a closed SCC C of $\mathcal{M}$ is the projection of an SCC C' of $\mathcal{M} \times A_\psi$ then C' is closed.*

We finally arrive at the lemma which characterizes the closed SCCs of the product automaton which contribute to the measure of computations that satisfy ψ.

Lemma 3.8 *The measure of the set computations, of $\mathcal{M}$, starting at q_0, which satisfy ψ is equal to the measure of the projections onto $\mathcal{M}$ of computations from $[\mathcal{M} \times A_\psi](q_0)$ which loop in a closed SCC containing a final state.*

Proof: For a computation to satisfy ψ, it must have a run and the run must be accepting. If this computation loops in an SCC that is not closed, it belongs to a set of computations that have 0 measure. Assume it loops in a closed SCC C. If C is not the projection of an SCC of $[\mathcal{M} \times A_\psi](q_0)$ then clearly none of the computations of that loop in C can satisfy ψ. Let C be the projection of an SCC C' of $[\mathcal{M} \times A_\psi](q_0)$. By Lemma 3.7, C' is closed. If C' has no accepting state then, by Lemma 3.6, no computation that loops in C can satisfy ψ. If C' has an accepting state then every computation which visits all states of C infinitely often has a corresponding computation in $[\mathcal{M} \times A_\psi](q_0)$ that visits the final state infinitely often; hence all computations that visit all states of C, infinitely often, satisfy ψ. Consequently, by Lemma 3.4, the measure of all computations that loop in C and satisfy ψ is equal to the measure of all computations that are projections of computations of $\mathcal{M} \times A_\psi$ that loop in C'. ■

The algorithm for **sat-states** presented in Figure 2 implements the strategy outlined by the preceding lemma. We can, consequently, infer the correctness of **sat-states**.

Lemma 3.9

1. *Algorithm* **sat-states** *terminates on all inputs.*

2. **sat-states**$(\mathcal{M}, A_\psi, p)$ *computes the set of states which satisfy $Æ_p\psi$.*

This leads us (along with Theorem 3.1) to the final correctness theorem:

Theorem 3.10 *Algorithm* PCTL*-sat *returns* **true** *on the markov chain $\mathcal{M}$, a start state q_s and PCTL* formula ϕ iff ϕ holds in the state q_s of $\mathcal{M}$.*

3.2 Complexity

In this subsection we will provide a calculation for the computational complexity of our algorithm. To that end, assume $\mid \mathcal{M} \mid$ to be the number of nodes of $\mathcal{M}$, and $\mid \phi \mid$ to be the size of the formula ϕ. We start the calculations with the following known result:

Theorem 3.11 ([15, 16]) *Given an LTL formula ψ a Büchi automaton, corresponding to ψ, containing $O(2^{|\psi|})$ states can be constructed.*

We will first analyze the complexity of the algorithm **sat-states**. Since the running time of that algorithm is dominated by Step 5, where a depth-first search needs to be carried out for each node of the automaton $A_{\mathcal{M},\psi}$, we have the following:

Lemma 3.12 *The complexity of each call of* **sat-states** *is* $O(\mid \mathcal{M} \mid^3 \times 2^{3|\phi|})$.

Now we move to Algorithm **PCTL*-sat**. Clearly, the number of iterations of this algorithm is bounded by $O(\mid \phi \mid)$ and the dominant step is Step 3, where **sat-states** is called. This leads us to:

Theorem 3.13 *The complexity of* **PCTL*-sat** *is* $O(\mid \phi \mid \times \mid \mathcal{M} \mid^3 \times 2^{3|\phi|})$.

Call the model-checking algorithm **PCTL-sat** when the formulae are restricted to be PCTL formulae. Since the size of the path formulae we need to consider for **sat-states** is constant, we have the following:

Corollary 3.14 *The complexity of* **PCTL-sat** *is* $O(\mid \phi \mid \times \mid \mathcal{M} \mid^3)$.

4 Conclusion

We have defined a new logic of chance which can express (explicitly) the measure with which we expect a property to hold of a system. We have also presented a model-checking algorithm which deals with the specific probabilities, but yet makes use of algorithms from non-probabilistic model-checking. We believe that this aspect of our algorithm is important as, often times, it is claimed that the probabilistic model-checking with traditional quantifiers is reducible to non-probabilistic model-checking. Finally, we have also shown how the new quantifier, we have introduced, is more general than the notions of *almost always* and *definitely sometimes*.

Acknowledgment: We would like thank V. Natarajan for his help on Markov chains.

References

[1] *Fifth Annual Symposium on Logic in Computer Science '90*, Philadelphia, June 1990. Computer Science Press.

[2] A. Sistla, M. Vardi, and P. Wolper. The complementation problem for Büchi automata with applications to temporal logic. *Theoretical Computer Science*, 49:217–237, 1987.

[3] R. Alur, C. Courcoubetis, and D.L. Dill. Model-checking for probabilistic real-time systems. In *Automata, Languages and Programming: Proceedings of the 18th ICALP*, volume 510 of *LNCS*, pages 115–136, 1991.

[4] J. Atlee, M. Chechik, and J. Gannon. *Advances in Computers*, chapter Using Model Checking to Analyze Requirements and Designs. 1995.

[5] A. Aziz, K. Sanwal, V. Singhal, and R.K. Brayton. Verifying continuous-time markov chains. In *Proc of Computer Aided Verification '96*. Springer-Verlag, Jul 1996.

[6] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic model checking: 10^{20} states and beyond. In *Fifth Annual Symposium on Logic in Computer Science (LICS '90)* [1], pages 428–439.

[7] C. Courcoubetis and M. Yannakakis. Verifying temporal properties of finite-state probabilistic programs. In *Proc. 1988 IEEE Symp. on the Foundations of Comp. Sci.*, 1988.

[8] E. A. Emerson and J. Y. Halpern. "sometimes" and "not never" revisited: On branching time versus linear time temporal logic. *JACM*, 33(1):151–178, 1986.

[9] H. Hansson. *Time and Probability in Formal Design of Distributed Systems*. Elsevier, 1994.

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

[11] G. Holzmann. *Design and Validation of Computer Protocols*. Prentice Hall, Englewood Cliffs, New Jersey, 1991.

[12] P. Iyer and M. Narasimha. Probabilistic lossy channel systems. Technical Report TR-96-14, NC State University, May 1996. Submitted for publication.

[13] D. Jackson, S. Jha, and Craig Damon. Faster checking of software specifications by eliminating isomorphs. In *Proc of Principles of Programming Languages '96*, pages 79–90, 1996.

[14] J. G. Kemeny, J. L. Snell, and A. W. Knapp. *Denumerable Markov Chains*. Van Nostrand, New Jersey, 1966.

[15] M. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In Dexter Kozen, editor, *First Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 322–331, 1986.

[16] M. Y. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *IEEE Symposium on Foundations of Computer Science*, pages 327–338, 1985.

[17] P. Wolper, M. Vardi, and A. Sistla. Reasoning about infinite computation paths. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, 1983.