Indian Institute of Technology, Guwahati

MA 402 Seminar Report

# “Probabilistic Model Checking”

Under the guidance of

**Dr. N Selvaraju**

Department of Mathematics

Indian Institute of Technology Guwahati

Submitted By,

**Varun Aggarwala (04010150)**

B. Tech, 7th Semester

Department of Computer Science and Engineering

Indian Institute of Technology Guwahati

## Abstract

Probabilistic modeling finds its uses in uses in *design* and *analysis* of computer systems and has seen unforeseen growth in the last decade. Its application varies from diverse areas such as randomized distributed algorithms, communication protocols, biological process modeling, power management and lots of other fields are cropping up. Probabilistic model checking focuses on the probability of a given temporal logic property being satisfied. Probabilistic Model Checking is an *automated technique* for checking if a specified property holds in a probabilistic model such as “after a request of service there is at least 90% probability that the service will be carried out within 1 second”. Queuing theory on the other hand helps us to understand and evaluate the properties of a markovian system.

The survey presents an overview of model checking of both discrete and continuous time Markov chains (DTMCs and CTMCs). We also discuss the syntax and semantic of PCTL and CSL logic. Model checking algorithms for verifying DTMCs against specification in the logic discussed in the survey, including quantitative properties with rewards will also be considered.

## Introduction:

Probability is an important aspect which is used in the design and analysis of computer systems and has seen an enormous upwards swing in the last decade. Traditionally probability has been used to analyze system performance, where by queuing theory was used to obtain various estimates of mean waiting time, throughput etc. Probability has also been used in various communication protocols and computer networks where various properties like packet loss and the channel failure can be analyzed probabilistically.
Probabilistic Model Checking has been defined as an automatic procedure for checking if a desired property (specified in temporal logic) holds in a probabilistic model. Conventional model checking focuses on the *absolute correctness* of a system which is difficult to realize in an real world scenario because system are subjected to various phenomenon of probabilistic type such as message loss, unpredictable environments, faults and delays. So the concept of absolute correctness is out of the window and here enters the realm of probabilistic model checking which can check properties like “the process terminates with probability more than 50%”.
In case of Conventional model checking the model description is just the state transition system. In case of Probabilistic model checking the model are probabilistic (variants of Markov chains), so as to encode the probability of making a transition instead between states instead of simply the possibility of such a transition. As a result of which we can make quantitative statements about the system, in addition the qualitative statements made by conventional model checking. Some of the real world scenarios captured by probabilistic model checkers are.*

- For a randomized leader election algorithm “leader election is resolved with probability 1”.
- For a communication medium “the packet will eventually be delivered within 10 ms with probability more than 0.5”.
- For a system which can suffer failures “the chance of shutdown is at maximum 0.1%”.
- For a sub clusters with N workstations each “in the long run, Premium QOS will be delivered with probability at least 0.7”.
- For FGF (Fibroblast Growth Factor) Signaling “the long run probability that Grb2 is bound to FRS2 is greater than 0.4”.

* They have actually been modeled and verified using Probabilistic Model Checker PRISM.

## Preliminaries

**Definition 1***:* *Let $\Omega$ be an arbitrary non-empty set and $F$ is a family of subsets of $\Omega$. We say that $F$ is a field on $\Omega$ if:*

- *The empty set $\varnothing$ is in $F$ ;*
- *Whenever $A$ is an element of $F$, then the complement $\Omega \setminus A$ is in $F$ ;*
- *Whenever $A$ and $B$ are elements of $F$, then $A \bigcup B$ is in $F$.*

A *field of subsets $F$* is *an $\sigma$ algebra* if it is field which is *closed under countable union i.e.* whenever $A_i \in F$ *for* $i \in N$ *, then* $\bigcup_{i \in N} A_i$ *is also in* $F$ .
The *elements of $\sigma$ algebra* are called *measurable sets*, and *$(\Omega , F)$* is a *measurable space.*

**Proposition 1**: For any *non empty set* $\Omega$ and $A$ *is a family of subsets of* $\Omega$, there is a *unique smallest $\sigma$ algebra* containing $A$ .

**Definition 2:** *Let $(\Omega , F)$ is a measurable space. A function $u : F \rightarrow [0,1]$ is a probability measure on $(\Omega , F)$ and $(\Omega , F , u)$ a probability space, if $u$ satisfies the following properties:*

- $u(\Omega) = 1$*;*
- $u(\bigcup_{i=1}^{k} A_i) = \sum_{i=1}^{k} u(A_i)$ *for any countable disjoint sequence* $A_{1,} A_2 .....$ *of* $F$ .

The *measure $u$* is known as *probability distribution.* $\Omega$ is known as *sample space* and the *elements of $F$* are called *events*.

**Definition 3:** *A family of subsets of $\Omega$ is called a semiring if*

- *The empty set $\varnothing$ is in $F$ ;*
- *Whenever $A$ and $B$ are elements of $F$, then $A \bigcap B$ is in $F$ ;*
- *If $A \subseteq B$ is in $F$, then there are many finitely many pair wise disjoint subsets $C_1, ....., C_k \in F$ such that $B \setminus A = \bigcup_{i=1}^{k} C_i$ .*

**Theorem 1:** *If $F$ is a semiring on $X$ and $u : F \rightarrow [0, \infty]$ satisfies*

- $u(\varnothing) = 0$ *;*
- $u(\bigcup_{i=1}^{k} A_i) = \sum_{i=1}^{k} u(A_i)$ *for any finite disjoint sequence* $A_{1,} A_2 ....., A_k \in F$ *;*
- $u(\bigcup_{i=1}^{k} A_i) \leq \sum_{i=1}^{k} u(A_i)$ *for any countable sequence* $A_{1,} A_2 ..... \in F$ ,

*Then $u$ extends to a unique measure on the $\sigma$ algebra generated by $F$ .*

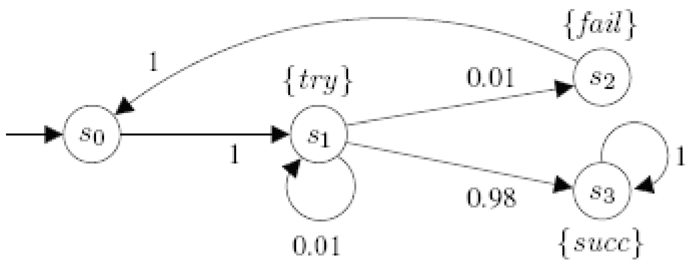
**Definition 4:** *Let* $(\Omega, F, u)$ *be a probability space. A function* $X : \Omega \to R_{\geq 0}$ *is called a random variable.*

## Model Checking Discrete Time Markov Chains

Let *AP* be a *fixed* and *finite* set of *atomic propositions*, which are used to label states of a Markov Chain, and express properties of a state. We assume that we can easily determine the validity of an atomic proposition in a state.

**Definition 5:** *A labeled DTMC D is a tuple* $(S, s^i\ P, L)$ *where*

- *S is a finite set of states;*
- $s^i \in S$ *is the initial state;*
- *P:* $S \times S \to [0,1]$ *is the transition probability matrix where* $\sum_{s' \in S} P(s, s') = 1$ *for all* $s \in S$*;*
- *L:* $S \to 2^{AP}$ *is a labeling function which assigns to each state* $s \in S$ *the set L(s) of atomic propositions that are valid in the state.*



A labeled DTMC $D_1$

The above diagram shows a labeled DTMC $(S, s^i\ P, L)$. The initial state is show here with an incoming arrow. The states S of the DTMC are $\{s_0, s_1, s_2, s_3\}$ and the initial state $s^i = s_0$. The transition probability matrix is

$$P = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0.01 & 0.01 & 0.98 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

The atomic propositions AP = {try, fail, succ}. The Labeling function assigns to each of the state L $(s_0)$ = $\emptyset$ , L $(s_1)$ = {try}, L $(s_2)$ = {fail}, L $(s_3)$ = {succ}.

## Paths and Probability Measures

A *path* is defined to be as an execution of a DTMC $D = (S, s^i$ P, L). A path $\omega$ is *a non empty sequence* of states $s_0 s_1 s_2 ......$ where $s_i \in S$ and P ($s_i, s_{i+1}$)>0 for all i$\geq 0$. A path can both be finite or infinite. The $i^{th}$ state of a path $\omega$ is denoted by $\omega$ (i). The *length of path* $\omega$ is denoted by $|\omega|$. The finite path is denoted by $\omega_{fin}$ and its *last state* by last ($\omega_{fin}$).

The *set of infinite and finite paths* of $D$ starting in state s are denoted by $Path^D$ (s) and $Path^D_{fin}$ (s) respectively. A finite path $\omega_{fin}$ of length n is a *prefix* of an infinite path $\omega$ if $\omega_{fin}(i) = \omega(i)$ for all $0 \leq$ i $\leq 1$.

For any finite path $\omega_{fin} \in Path^D_{fin}$ (s) we can define probability $P_s(\omega_{fin})$ to be

$$P_s(\omega_{fin}) = \begin{cases} 1 & if\ n = 0 \\ P(\omega(0), \omega(1)) ...... P(\omega(n-1), \omega(n)) & otherwise \end{cases}$$ where n=$|\omega_{fin}|$.

A *cylinder set* $C(\omega_{fin}) \subseteq Path^D(s)$ is defined as
$C(\omega_{fin}) = \{ \omega \in Path^D$ (s) | $\omega_{fin}$ is a prefix of $\omega$ }

This is the same as set of all infinite paths with prefix $\omega_{fin}$. Let $\sum Path^D(s)$ is the *smallest sigma algebra* on $Path^D$ (s) which contains all the sets $C(\omega_{fin})$ where $\omega_{fin}$ ranges over all the finite paths $Path^D_{fin}$ (s). The set of cylinders form a semiring over ($Path^D$ (s), $\sum Path^D(s)$) we can apply Theorem 1 and can define a *probability measure* $\Pr_s$ on ($Path^D$ (s), $\sum Path^D(s)$) as $\Pr_s(C(\omega_{fin})) = P_s(\omega_{fin})$ for all $\omega_{fin} \in$ $Path^D_{fin}$ (s).

$\Pr_s$ will be used to quantify that, starting from a state s∈S the DTMC $D$ behaves in the property specified by indentifying the set of paths which satisfy the property. We assume that the set of paths is measurable so we can use $\Pr_s$ in this regard.

## Probabilistic Computation Tree Logic (PCTL)

We specify the properties for the DTMC model using PCTL (Probabilistic Computation Tree Logic), which is almost a probabilistic extension of CTL (Computational Tree Logic).

**Definition 6:** *The syntax of PCTL logic is as follows:*

$$\varphi = true \mid a \mid \neg\varphi \mid \varphi \wedge \varphi \mid P_{\square}[\phi]$$

$$\phi = X\varphi \mid \varphi U^{\leq k} \varphi$$

*Where a is an atomic proposition,* $\sqcup \in \{<, \leq, >, \geq\}$, $p \in [0,1]$ *and* $k \in N \cup \{\infty\}$.

We define $\varphi$ to be a *state formula* evaluated over the states of DTMC while $\phi$ which is a *path formula* is evaluated over the paths. For specifying a property in DTMC we always use the state formula. Only when specifying something like this $P_{\Box}[\phi]$ do we need to use the path formula. The above formula only specifies that the formula is only satisfied if the probability of taking a path from s satisfying $\phi$ is in interval specified by $\Box\ p$. There are path formula which are made from *X ("next")* and $U^{\leq}$ *("bounded until")* operator.
$X\varphi$ is satisfied when $\varphi$ is satisfied in the next state. $\varphi U^{\leq k}\psi$ is satisfied if $\psi$ is satisfied inside k time steps and $\varphi$ is true till that point.

Writing $s \mapsto \varphi$ indicates that s satisfies $\varphi$ where s is a state and $\varphi$ is a PCTL formula. Writing $\omega \mapsto \phi$ indicates that path $\omega$ satisfies the PCTL path formula $\phi$.

**Definition 7:** *D=* $(S, s^i, P, L)$ *be a labeled DTMC. We define the relation* $\mapsto$, *for any state* $s \in S$ *as*

- $s \mapsto true$ *for all* $s \in S$
- $s \mapsto a \quad \Leftrightarrow \quad a \in L(s)$
- $s \mapsto \neg\varphi \quad \Leftrightarrow \quad s \not\mapsto \varphi$
- $s \mapsto \varphi \wedge \psi \quad \Leftrightarrow \quad s \mapsto \varphi \wedge s \mapsto \psi$
- $s \mapsto P_{\Box}[\phi] \quad \Leftrightarrow \quad \mathrm{Prob}^D(s,\phi)\ \Box\ p$

*Where* $\mathrm{Prob}^D(s,\phi) = \Pr_s\{\omega \in Path^D(s) \mid \omega \mapsto \phi\}$
*Also for any path* $\omega \in Path^D(s)$ *we have*
$\omega \mapsto X\varphi \quad \Leftrightarrow \quad \omega(1) \mapsto \varphi$
$\omega \mapsto \varphi U^{\leq k}\psi \quad \Leftrightarrow \quad \exists i \in N\ (i \leq k \wedge \omega(i) \mapsto \psi\ \wedge \forall j < i.(\omega(j) \mapsto \phi))$

Some of the useful operators that can be specified using PCTL logic are

- *Eventually Operator* (diamond operator) $\Diamond$

  $\Diamond\varphi$ means that $\varphi$ is eventually satisfied.
  $\Diamond^{\leq k}\varphi$ means that $\varphi$ is satisfied within k time units
  $P_{\Box}[\Diamond^{\leq k}\varphi] = P_{\Box}[\mathrm{true}U^{\leq k}\varphi]$
  This operator is also known as the *F operator* in some texts

- *Always Operator* (box operator) $\Box$

  $\Box\varphi$ means that $\varphi$ is true in every state of the path
  $\Box^{\leq k}$ means that $\varphi$ is true in first k states of the path

$\Box$ can be expressed in terms of $\Diamond$ as:
$\Box\,\varphi = \neg\Diamond\neg\varphi$

We are not equipped with the negation operator in the PCTL syntax so we solve for the box operator using the fact that $\text{Prob}^D(s,\neg\phi) = 1 - \text{Prob}^D(s,\phi)$

$$
\begin{aligned}
P_{\geq p}[\Box\,\varphi] &\Leftrightarrow \text{Prob}^D(s,\Box\,\varphi) \geq p \\
&\Leftrightarrow 1 - \text{Prob}^D(s,\Diamond\neg\varphi) \geq p \\
&\Leftrightarrow \text{Prob}^D(s,\Diamond\neg\varphi) \leq 1 - \text{p} \\
&\Leftrightarrow P_{\leq 1-p}[\Diamond\neg\varphi]
\end{aligned}
$$

This operator is also known as the *G operator* in some texts.

- *A operator* which means that for *all the paths* the PCTL state formula is satisfied
  $\text{A}\,[\varphi] \quad \Leftrightarrow \quad P_{\geq 1}[\varphi]$

- *E operator* which means that *at least for one path* the PCTL formula is satisfied
  $\text{F}\,[\varphi] \quad \Leftrightarrow \quad P_{\geq 0}[\varphi]$

## Model Checking PCTL

We present algorithms for model checking PCTL over the DTMC structure. The input to the algorithm is a DTMC $D = (S, s^i, P, L)$ and a PCTL formula $\varphi$. The output is the *set of states* containing all the states of the model which satisfy $\varphi$. This set is denotes by *Sat* $(\varphi) = \{s \in S \mid s \mapsto \varphi\}$. We will get the result by checking if only the initial state $s^i$ satisfies $\varphi$ however the algorithm computes it for all the states.

The algorithm can be summarized as follows

$$
\begin{aligned}
&Sat\ (true) && = S \\
&Sat\ (a) && = \{s \mid a \in L(s)\} \\
&Sat\ (\neg\varphi) && = S \setminus Sat\ (\varphi) \\
&Sat\ (\varphi \wedge \psi) && = Sat\ (\varphi) \bigcap Sat\ (\psi) \\
&Sat\ (P_{\Box}[\phi]) && = \{s \in S \mid \text{Prob}^D(s,\phi) \Box\ p\}
\end{aligned}
$$

Model Checking for the first four cases is trivial but for the $P_{\Box}[\phi]$ case we have to calculate for all the states s of the DTMC $\text{Prob}^D(s,\phi)$ and then compare it to the bound p specified. A point to note is that the model checking algorithm here is *recursive* in nature and for formulas like $P_{\geq p}[\varphi U^{\leq k}\psi]$ we assume that Sat $(\varphi)$ and Sat $(\psi)$ are already known.

- Model Checking $P_{\Box}\,[X\varphi]$ formula

This asks us for calculating the probabilities of the immediate transitions from state s∈S of the DTMC *D*

$$\text{Prob}^{D}(s, X\varphi) = \sum_{s' \in Sat(\varphi)} P(s, s')$$

The vector $\text{Prob}^{D}(\text{X}\varphi)$ of the probabilities for all the states can be computed as follows

$$\text{Prob}^{D}(X\varphi) = P.\ \underline{\varphi}$$

*Where* $\underline{\varphi}$ *is a state indexed column vector defined as*

$$\underline{\varphi} = \begin{Bmatrix} 1 & if\ s \in Sat(\varphi) \\ 0 & otherwise \end{Bmatrix}$$

- Model Checking $P_{\bowtie}[\varphi U^{\le k}\psi]$ formula

This asks us to find the probability $\text{Prob}^{D}(s,\ \varphi U^{\le k}\psi)$ for all states s where k ∈N.
For this let us define

$S^{Yes}$ = *Sat* ($\psi$)
$S^{>0}$ = *Reach* ($\varphi$, $\psi$) *i.e. states from where one can reach a* $\psi$ *state from a* $\varphi$ *state*
$S^{No}$ = $S \setminus S^{>0}$
$S^{?}$ = $S \setminus S^{No} \cup S^{Yes}$

Therefore $\text{Prob}^{D}(s,\ \varphi U^{\le k}\psi)$ can be defined as

$$\text{Prob}^{D}(s,\ \varphi U^{\le k}\psi) = \begin{Bmatrix} 0 & if\ s \in S^{No} \\ 1 & if\ s \in S^{Yes} \\ \sum_{s' \in S} P(s, s^{i}).\text{Prob}^{D}(s', \varphi U^{\le k-1}\psi) & if\ s \in S^{?} \end{Bmatrix}$$

$$So\ \text{Prob}^{D}(s,\ \varphi U^{\le k}\psi) = \sum_{s' \in S^{?}} P(s, s^{i}).\text{Prob}^{D}(s', \varphi U^{\le k-1}\psi) + \sum_{s' \in S^{yes}} P(s, s^{i})$$

Equivalently in *matrix form* this can be specified as
$X = A.X + P.b$
*Where*
*A is the sub matrix for* $S^{?}$
*b is the column vector for* $S^{Yes}$

This can be solved through *direct method* which can be

- *Gaussian Elimination*
- *LU Decomposition*

This can also be solved using *iteration*

$$\text{Prob}^D(s\ ,\varphi U^{\leq 0}\psi) \quad = 0$$
$$\text{Prob}^D(s\ ,\varphi U^{\leq k+1}\psi) \quad = \sum_{s'\in S^?} P(s,s^i).\text{Prob}^D(s',\varphi U^{\leq k}\psi) \ + \sum_{s'\in S^{yes}} P(s,s^i)$$

## Extending DTMCs and PCTL with Rewards

For a DTMC $D = (S, s^i\ \text{P, L})$ we can define a *reward structure* $(\rho, \iota)$. This structure can help us define two types of rewards

- *State rewards* which are assigned to states by means of a reward function $\rho : \text{S} \rightarrow R_{\geq 0}$. The state reward is acquired in state s per time-step.
- *Transition rewards* which are assigned to transitions by means of a reward function $\iota : \text{S}\ X\ \text{S} \rightarrow R_{\geq 0}$. The transition reward is acquires each time a transition occurs from one state to another.

Reward structure can be used to represent various additional information about the system modeled by the DTMC for example the cost incurred in spending some time in an particular state like in Power consumption the active states requires more power.

The PCTL logic is extended to incorporate the reward structure as follows

$R_{\sqcup r}[C^{\leq K}] \mid R_{\sqcup r}[I^{=K}] \mid R_{\sqcup r}[F\varphi]$

*Where* $\sqcup \in \{<, \leq, >, \geq\}$, *k*∈*N*, *r*∈ $R_{\geq 0}$ *and* $\varphi$ *is a PCTL state formula*

Some example of reward based specification using these formulae are

- $R_{\leq 3}[C^{\leq 100}]$ : The expected power consumption within the first 100 time steps of operation is less than equal to 3.
- $R_{\geq 4}[I^{=10}]$ : The expected number of messages delivered after 10 time steps are at least 4.
- $R_{\geq 5}[F\ succ]$ : The expected number of successful messages is at least 5.

## Model Checking Continuous Time Markov Chains

Discrete time Markov chains are not capable enough of modeling continuous time. Therefore we resort to introducing a new model with *discrete state space* but with *continuous time*.

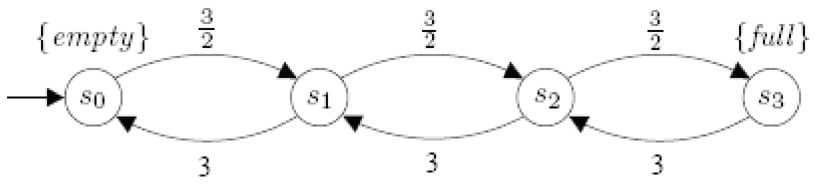
**Definition 8:** *A labeled CTMC C is a tuple C = $(S, s^i, R, L)$ where*

- *S is a finite set of states;*
- $s^i \in S$ *is the initial state;*
- *R:* $S \times S \rightarrow R_{\geq 0}$ *is the transition rate matrix.*
- *L:* $S \rightarrow 2^{AP}$ *is a labeling function which assigns to each state s∈S the set L(s) of atomic propositions that are valid in the state.*

The transition rate matrix R assigns rates to every pair of states in the CTMC model which are used as parameters of the exponential distribution. $R(s, s^{'}) \geq 0$ implies that a transition can occur between states s and $s^{'}$ with probability of being triggered in *t* time units is $1 - e^{-R(s,s^{'}).t}$ .
If in a state s there is more than one state $s^{'}$ with R(s,$s^{'}$) > 0 than there is a *race* condition. The first transition to occur then describes the next state of the CTMC. In the case of race condition the probability of going to any of the next state $s^{'}$ is

$$P(s, s^{'}) = R(s, s^{'}) / \sum_{s''\in S} R(s, s'')$$

However if there are no outgoing transition from a state s, then $P(s, s^{'}) = 1$ *iff* $s = s^{'}$ *and 0 if* $s \neq s^{'}$.



A labeled CTMC *C*

The above diagram is a CTMC $C = (S, s^i$, R, L). The initial state is show here with an incoming arrow. The states S of the CTMC are $\{s_0, s_1, s_2, s_3\}$ and the initial state $s^i = s_0$. The transition rate matrix R is

$$R = \begin{pmatrix} 0 & 1.5 & 0 & 0 \\ 3 & 1.5 & 0 & 0 \\ 0 & 3 & 0 & 1.5 \\ 0 & 0 & 3 & 0 \end{pmatrix}$$

The atomic propositions AP = {empty, full}. The Labeling function assigns to each of the state L ($s_0$) = empty, L ($s_1$) = {$\varnothing$}, L ($s_2$) = {$\varnothing$}, L ($s_3$) = {full}.

## Paths and Probability Measures

A *infinite path* of a CTMC $C$ = ($S, s^i$, R, L) is defined as an non empty sequence $s_0t_0s_1t_1s_2$...... where R($s_i, s_{i+1}$)>0 and $t_i \in R_{>0}$ for all $i \geq 0$. A *finite path* is a sequence $s_0t_0s_1t_1s_{k-1}t_{k-1}s_k$ such that state $s_k$ is *absorbing (* $\sum_{s \in S} R(s, s')$ *= 0).* The value $t_i$ represents the time spent in state i. The $i^{th}$ state of a path $\omega$ is denoted by $\omega$(i). For an infinite path $\omega$ we denote by *time (*$\omega$*, i)* the *time spent* in the state $s_i$. $\omega$@t denotes the *state s occupied at time t.* For a finite path $\omega = s_0t_0s_1t_1s_{k-1}t_{k-1}s_k$, time ($\omega$,i) is defined for all i$\leq$k and time ($\omega$, k) = $\infty$.

$Path^C$ (s) denotes the set of all *infinite and finite paths* of the CTMC $C$ starting in state s. If the states $s_0 \ldots\ s_n \in$S satisfy $R(s, s') > 0$ for all 0$\leq$ i < n and $I_0, ..., I_{n-1}$ are non empty intervals in $R_{\geq 0}$ then the *cylinder set C* ($s_0, I_0, ..., I_{n-1}, s_n$) is defined as the set containing all paths $\omega \in Path^C(s_0)$ such that $\omega$(i) = $s_i$ for all i$\leq$n and time ($\omega$, i) $\in I_i$ for all I < n.

We denote by $\sum Path^C(s)$ the *smallest sigma algebra* on $Path^C(s)$ which contains all cylinder sets $C$ ( $s_0, I_0, ..., I_{n-1}, s_n$ ). The set of cylinders form a *semiring* over ($Path^C(s)$, $\sum Path^C(s)$) therefore by Theorem 1 we can define the *unique probability measure* $\Pr_s$ on the smallest sigma algebra $\sum Path^C(s)$.

We also consider for a CTMC $C$ additional properties like

- Transient behavior which describes the state of the CTMC at a particular time instant. For a CTMC $C$ = ($S, s^i$, R, L), the transient probability $\pi^C_{s,t}(s')$ is defined as the probability of having started in state s and being in state $s'$ at time t.
$$\pi^C_{s,t}(s') = \Pr_s \{ \omega \in Path^C(s) \mid \omega @t = s' \}$$

- Steady State behavior which describes the state of the CTMC in the long run.
$$\pi^C_s(s') = \lim_{t \to \infty} \pi^C_{s,t}(s')$$

## Continuous Stochastic Logic (CSL)

We specify the properties for the CTMC model using CSL (Continuous Stochastic Logic), which is a probabilistic extension of CTL (Computational Tree Logic).

**Definition 9:** *The syntax of the CSL logic is as follows*

$$\varphi = true \,|\, a \,|\, \neg\varphi \,|\, \varphi \wedge \varphi \,|\, P_{\Box}[\phi] \,|\, S_{\Box p}[\varphi]$$
$$\phi = X\varphi \,|\, \varphi U^{I}\varphi$$

*Where a is an atomic proposition,* $\sqcup \in \{<, \leq, >, \geq\}$, *p*∈*[0,1] and I is an interval of* $R_{\geq 0}$

As for PCTL $P_{\Box}[\phi]$ formula specifies that the probability of taking a path from s satisfying *path formula* $\phi$ is in interval specified by $\Box\, p$. Path formula for CSL and PCTL are same except for the fact that the parameter of the until operator is an *interval I of non negative reals*, rather than an integer bound as was in the case of PCTL. The path formula $\varphi U^{I}\psi$ is true when $\psi$ is satisfied at some instant in interval I and $\varphi$ holds at all previous time instants. The *S* operator specifies the steady state behavior of CTMC. Intuitively the state formula $S_{\Box p}[\varphi]$ means that the steady state probability of being in state satisfying $\varphi$ meats the bound $\Box\, p$.

$s \mapsto \varphi$ indicates that the CSL formula $\varphi$ is satisfied in a state s.

$\omega \mapsto \phi$ indicates that path formula $\phi$ is satisfies in a path $\omega$.

**Definition 10:** *C=* $(S, s^{i}, P, L)$ *be a labeled CTMC. We define the relation* $\mapsto$, *for any state s*∈*S as*

- $s \mapsto true$ *for all s*∈*S*
- $s \mapsto a \quad \Leftrightarrow \quad a \in L(s)$
- $s \mapsto \neg\varphi \quad \Leftrightarrow \quad s \not\mapsto \varphi$
- $s \mapsto \varphi \wedge \psi \quad \Leftrightarrow \quad s \mapsto \varphi \wedge s \mapsto \psi$
- $s \mapsto P_{\Box}[\phi] \quad \Leftrightarrow \quad \mathrm{Prob}^{C}(s,\phi) \Box\, p$
- $s \mapsto S_{\Box p}[\varphi] \quad \Leftrightarrow \quad \sum_{s' \mapsto \varphi} \pi_{s}^{C}(s') \Box\, p$

*Where* $\mathrm{Prob}^{C}(s,\phi) = \mathrm{Pr}_{s}\{\omega \in Path^{C}(s) \mid \omega \mapsto \phi\}$

*Also for any path* $\omega \in Path^{C}(s)$ *we have*

$\omega \mapsto X\varphi \quad \Leftrightarrow \quad \omega(1) \mapsto \varphi$

$\omega \mapsto \varphi U^{I}\psi \quad \Leftrightarrow \quad \exists t \in N(\omega @ t \mapsto \psi \wedge \forall x \in [0,t).(\omega @ x \mapsto \varphi)$

The box and the diamond operator for the case of CSL are derived exactly in the same manner as in the PCTL case.

CSL does not explicitly include operators to reason about transient probabilities, but it can be stated by a trick. The probability of satisfying a formula $\varphi$ at time instant t is

$$P_{\Box p}[\Diamond^{[t,t]}\varphi]$$

Some examples of the CSL formulae are

- $P_{>0.9}[\Diamond^{[0,4.5]} served]$ : The probability that a request is served within the first 4.5 seconds is greater than 0.9.

- $down \rightarrow P_{0.75}[\neg fail U^{[1,2]} up]$ : When a shut down occurs ,the probability of system being up within 1 or 2 hours without further failures is greater than 0.75.

## The Probabilistic Model Checker PRISM

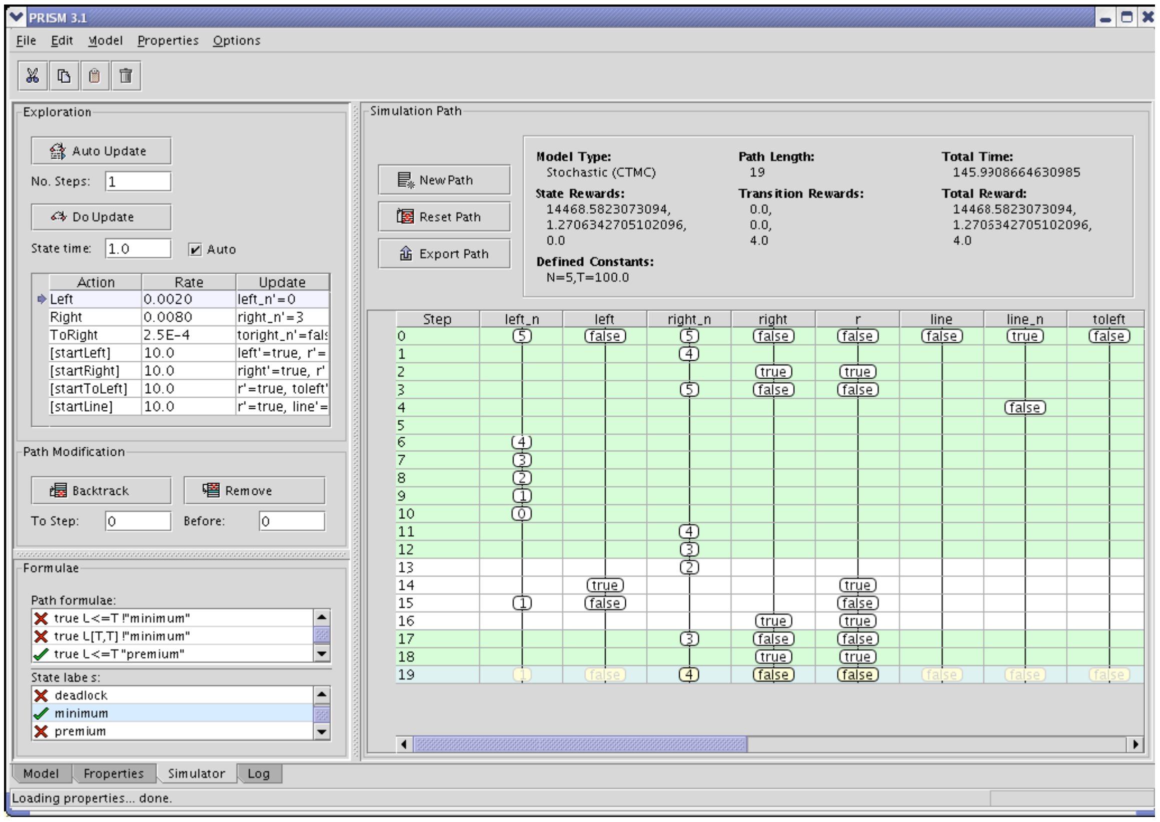
PRISM [7, 8] is a probabilistic model checker tool developed at University of Birmingham. It accepts probabilistic models described in the PRISM *modeling language*, which is a high level state based language. PRISM has direct support for three types of probabilistic models: Discrete Time Markov Chains (DTMCs), Continuous Time Markov Chains (CTMCs) and Markov Decision Processes (MDPs).

**Tool Overview:**

- PRISM works by first parsing the model description and then constructing an internal representation of the probabilistic model, computing the reachable state space of the model and discarding any unreachable states. This represents the feasible configurations that can arise in the modeled system.

- Now, the specification is parsed and model checking algorithms are performed on the model.

- PRISM can report a true/false answer, indicating if a property is satisfied by the model. PRISM can also return *quantitative* results like the probability of a certain event occurring in the model. PRISM also supports the idea of *experiments* by which one can see the outcome of properties of the model as a function of the model.



No. Steps: 1

State time: 1.0 ✓ Auto

| Action | Rate | Update |
|---|---|---|
| Left | 0.0020 | left_n'=0 |
| Right | 0.0080 | right_n'=3 |
| ToRight | 2.5E-4 | toright_n'=fals |
| [startLeft] | 10.0 | left'=true, r'= |
| [startRight] | 10.0 | right'=true, r' |
| [startToLeft] | 10.0 | r'=true, toleft' |
| [startLine] | 10.0 | r'=true, line'= |

To Step: 0 Before: 0

Path formulae:

- ✗ true L<=T !"minimum"
- ✗ true L[T,T] !"minimum"
- ✓ true L<=T "premium"

State labe s:

- ✗ deadlock
- ✓ minimum
- ✗ premium

**Model Type:** Stochastic (CTMC)
**Path Length:** 19
**Total Time:** 145.9308664630985
**State Rewards:** 14468.5823073094, 1.2706342705102096, 0.0
**Transition Rewards:** 0.0, 0.0, 4.0
**Total Reward:** 14468.5823073094, 1.2706342705102096, 4.0
**Defined Constants:** N=5,T=100.0

| Step | left_n | left | right_n | right | r | line | line_n | toleft |
|---|---|---|---|---|---|---|---|---|
| 0 | 5 | false | 5 | false | false | false | true | false |
| 1 | | | 4 | | | | | |
| 2 | | | | true | true | | | |
| 3 | | | 5 | false | false | | | |
| 4 | | | | | | | false | |
| 5 | | | | | | | | |
| 6 | 4 | | | | | | | |
| 7 | 3 | | | | | | | |
| 8 | 2 | | | | | | | |
| 9 | 1 | | | | | | | |
| 10 | 0 | | | | | | | |
| 11 | | | 4 | | | | | |
| 12 | | | 3 | | | | | |
| 13 | | | 2 | | | | | |
| 14 | | true | | | true | | | |
| 15 | 1 | false | | | false | | | |
| 16 | | | | true | true | | | |
| 17 | | | 3 | false | false | | | |
| 18 | | | | true | true | | | |
| 19 | 1 | false | 4 | false | false | false | false | false |

Loading properties... done.

A screenshot of PRISM GUI

PRISM is bundled with a *graphical user interface*, which can illustrate the results of a model checking experiment. It also has a built in *text editor* for the PRISM language. A *command line version* of the tool is also available. PRISM is a *free, open source* application. At present PRISM operates on UNIX, Windows and Macintosh operating system. Prism can be downloaded from the website.*

**Implementation:** PRISM is a *symbolic model checker* which means that its implementation uses data structures based on *binary decision diagrams (BDDs)*. These data structures provide compact representation and efficient manipulation of large, structured model by exploiting their regular structure. The reason for this is that the models are specified in a structure, high level modeling language PRISM. Actually PRISM uses *multi terminal binary decision diagrams* developed to improve the efficiency of probabilistic analysis.
The underlying computation in PRISM is a combination of:

- **Graph algorithms** for reachability analysis, conventional model checking and probabilistic model checking.

- **Numerical Computation** for *quantitative* model checking, e.g. solution of linear equation systems (for CTMCs and DTMCs)

**PRISM modeling language:** The fundamental components of the PRISM language are *modules* and *variables*. Variables are typed (can be integers, real or Boolean) and can be local to a module or global. The model is composed of modules which interact with each other. A module is composed of local variables and the values of these variables determine the state of the module. We define *global state* of the module as a function of *local state* of the modules and the *global variables*. The behavior of the module is specified in terms of set of *commands.*

## Case Study 1: Synchronous Leader election Protocol.

This case study is based on the synchronous leader election protocol of Itai and Rodeh [6]. The protocol is a solution to the problem stated below:

"Given a synchronous ring of ***N*** processors design a protocol such that they will be able to elect a leader (a uniquely designated processor) by sending messages around the ring."

The protocol proceeds in rounds and is parameterized by a constant K. Each round begins by all processors (independently) choosing a random number (uniformly) from {1 ... **K}** as an id. The processors then pass their ids around the ring. If there is a unique id, then the processor with the *maximum unique* **id** is elected the leader, and otherwise the processors begin a new round.
We assume that the ring is *synchronous*: there is a global clock and at every time slot a processor reads the message that was sent at the previous time slot (if it exists), makes at most one state transition and then may send at most one message.

The model is constructed as a *DTMC* and some of the properties verified for the model are

- *With probability 1, eventually a leader is elected.*
  The property, in case of N = 4 is specified in terms of PCTL as

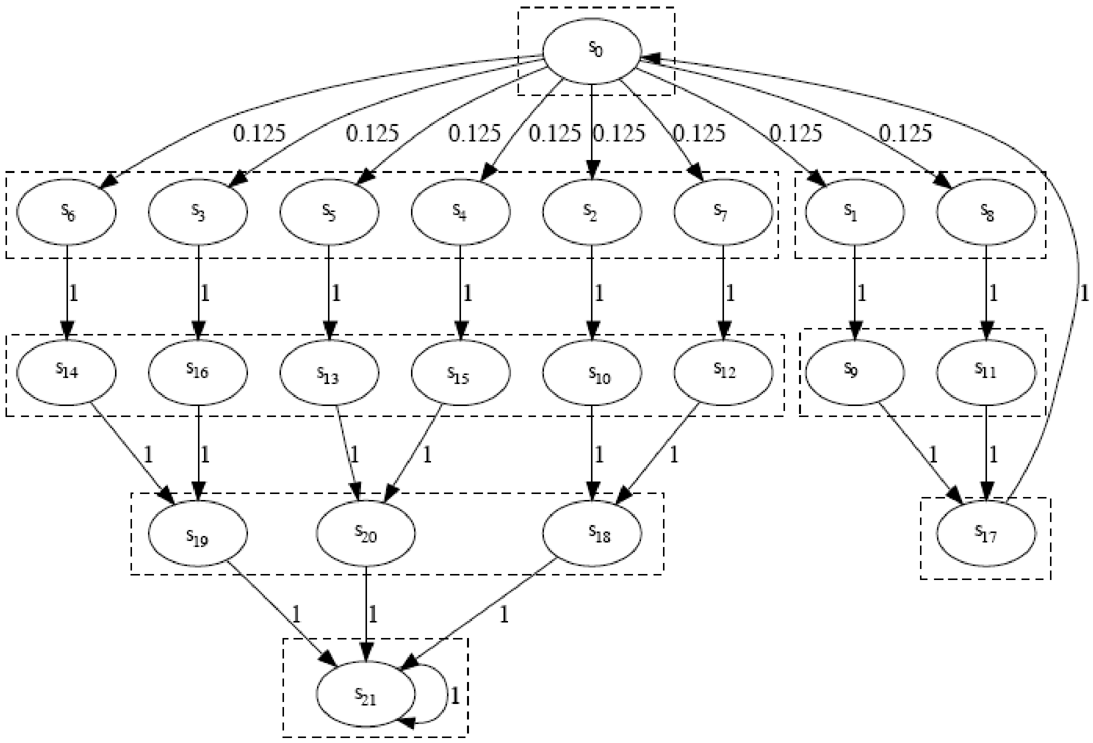
  P>=1 [true U (s1=3 & s2=3 & s3=3 & s4= 4)]

  Conclusion: The property holds in all states.

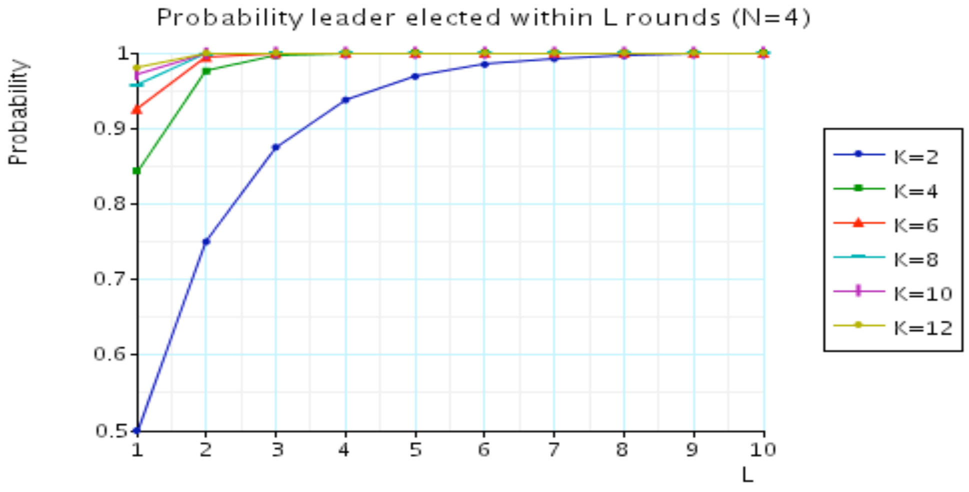
- *The probability of electing a leader in L rounds.*
  The property, in case of N = 4 is specified in terms of PCTL as

  P=? [true U<=L*(N+1) (s1=3 & s2=3 & s3=3 & s4= 4)]

DTMC for N=3 and K=2



$s_0$
0.125
$s_6$ $s_3$ $s_5$ $s_4$ $s_2$ $s_7$ $s_1$ $s_8$
1
$s_{14}$ $s_{16}$ $s_{13}$ $s_{15}$ $s_{10}$ $s_{12}$ $s_9$ $s_{11}$
$s_{19}$ $s_{20}$ $s_{18}$ $s_{17}$
$s_{21}$



The graph shows the expected value of L over the value of N = 4 and various values of K.

## Conclusion

Probabilistic Model checking provides a nice alternative to study the properties of a markov chain. Also many state space reduction techniques can be used to reduce the size of markov chain, which is an added advantage over the queuing theory techniques.

## References

[1] H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6(5):512-535, 1994

[2] J. P. Katoen. Stochastic Model Checking. In Christos G. Cassandras, and John Lygeros, editors, *Stochastic Hybrid Systems: Recent Developments and Research Trends.* Taylor and Francis, 2006.

[3] M. Kwiatkowska. Model Checking for Probability and Time: From Theory to Practice. In *Proc. 18th IEEE Symposium on Logic in Computer Science (LICS'03)*, pages 351-360, IEEE Computer Society Press. Invited paper. June 2003.

[4] M. Kwiatkowska, G. Norman and D. Parker. Stochastic Model Checking. In M. Bernardo and J. Hillston (editors) *Formal Methods for the Design of Computer, Communication and Software Systems: Performance Evaluation*, volume 4486 of Lecture Notes in Computer Science (Tutorial Volume), pages 220-270, Springer. To appear 2007.

[5] J. P. Katoen, M. Khattri and I. Zapreev. *A Markov reward model checker.* In *Quantitative Evaluation of Systems (QEST).* pages 243–244. IEEE CS Press, 2005.

[6] A. Itai and M. Rodeh. Symmetry breaking in distributed networks. *Information and Computation,* 88(1):60-87, 1990.

[7] A. Hinton, M. Kwiatkowska, G. Norman and D. Parker. PRISM: A Tool for Automatic Verification of Probabilistic Systems. In H. Hermanns and J. Palsberg (editors) *Proc. 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'06)*, volume 3920 of LNCS, pages 441-444, Springer. March 2006.

[8] PRISM web site. www.cs.bham.ac.uk/~dxp/prism

[9] J. P. Katoen, T. Kemma, I. Zapreev and D. Jansen. *Bisimulation minimization mostly speeds up probabilistic model checking.*. In *Proceedings of the 13th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS'07).* Springer, 2007.

[10] J. Heath, M. Kwiatkowska, G. Norman, D. Parker and O. Tymchyshyn. Probabilistic model checking of complex biological pathways. In C. Priami (editor) *Proc. Computational Methods in Systems Biology (CMSB'06)*, volume 4210 of Lecture Notes in Bioinformatics, pages 32-47, Springer Verlag. October 2006.

[11] M. Huth and M. Ryan. *Logic in Computer Science*, 2$^{nd}$ edition.