Learning Continuous Time Markov Chains from Sample Executions

Koushik Sen, Mahesh Viswanathan, Gul Agha
Department of Computer Science
University of Illinois at Urbana Champaign
{ksen,vmahesh,agha}@cs.uiuc.edu

Abstract

Continuous-time Markov Chains (CTMCs) are an important class of stochastic models that have been used to model and analyze a variety of practical systems. In this paper we present an algorithm to learn and synthesize a CTMC model from sample executions of a system. Apart from its theoretical interest, we expect our algorithm to be useful in verifying black-box probabilistic systems and in compositionally verifying stochastic components interacting with unknown environments. We have implemented the algorithm and found it to be effective in learning CTMCs underlying practical systems from sample runs.

1. Introduction

Stochastic models such as continuous-time Markov chains (CTMCs) [22] are widely used to model practical software systems and analyze their performance and reliability. Before building a complex software system, CTMCs are generated from higher-level specifications, like queueing networks, stochastic process algebra [14, 11], or stochastic Petri-nets [4]. These models are then used for quantitative evaluation of reliability and performance for example to determine the throughput of production lines, to calculate average failure time of systems, or to find other reliability or performance bottlenecks of the system. Once a model has been validated against performance and reliability requirements, the system is implemented. However, even if a model has been carefully validated, the implementation may not conform to the model. There are two potential sources of error: first, there could be bugs introduced when translating the design into system code, and second the estimated values of various parameters used in constructing the stochastic model may differ considerably from the actual values in the deployed system. To catch such potential post-implementation problems, testing for performance and reliability is performed by running the system a large number of times in the real environment and checking for reliability problems or performance bottlenecks. However, because it is difficult achieve complete *coverage* during testing, despite its evident importance in practice, testing fails to guarantee the full correctness of a deployed system.

An approach that tries to leverage the benefits of formal analysis - usually done in the design phase - to the post-implementation phase, is *learning* the model from sample executions of the system and then formally verifying the learnt model against the design specification. This approach has been fruitfully used to model-check unknown, *black-box* systems [9] and to learn unknown environments to assist in compositional verification of systems [7]. Both these efforts apply to non-deterministic, discrete systems and have not been extended to more general stochastic systems. While there are several machine learning algorithms on grammatical inference [6, 1, 8, 19, 5] that have been successfully applied to pattern recognition, speech recognition, natural language processing and several other domains, there are no algorithms in the literature that can learn the real-time, stochastic models that are typically used to model systems in formal verification. In this paper, we address this problem by presenting an algorithm that given execution traces (possibly obtained by running the deployed system during testing) of the system, infers a CTMC model that could have generated the traces according to the observed distribution. The learned CTMC can then be used by existing probabilistic model-checking [17, 12, 21, 24] and performance evaluation tools [14, 4] for further analysis and thereby helping to find bugs in post-implementation phase. The learning algorithm may also potentially be used to perform automatic compositional verification (as in [7]) for stochastic systems. A closely related work is given in [23] where they learn continuous-time hidden Markov models to do performance evaluation. However, they fix the size of the continuous-time hidden Markov model before learning. This can be restrictive if the system cannot be modelled by a continuous-time hidden Markov model of given size. In our approach there is no such restriction.

We present an algorithm and show that it correctly identifies the CTMC model in the limit [8] when it is given samples drawn from a distribution generated by a CTMC. One technical difficulty when talking about samples drawn from a CTMC is that traditionally CTMCs are unlabeled, and so they only have runs, which are sequences of states that are traversed and not traces. However, the problem is that when samples are drawn from an implementation get-

ting information that uniquely identifies states is expensive and impractical, and can lead to the construction of a very large model which does not collapse equivalent states. To address this difficulty, we introduce the model of an *Edge Labeled Continuous-time Markov Chain* ($CTMC_L$) where edges are labeled from a finite set of alphabet and traces are sequences of edge labels which are given to the learning algorithm.

Our algorithm is based on the state merging paradigm introduced and used in RPNI [19] and ALERGIA [6]. The samples provided to the learning algorithm are used to construct what we call a *prefix-tree continuous-time Markov chain*. Such Markov chains are the simplest CTMC that are consistent with the samples. The algorithm then progressively generalizes this model (i.e., produces models with additional behaviors) by merging pairs of states about whose "equivalence" the sample has evidence. Since the traces do not have complete state information about the original CTMC states, statistical information present in the samples is used to distinguish states. Our key algorithmic insight is in determining the statistical tests that can be used to conclude the equivalence of states with a given confidence. The candidate states that are tested for equivalence by the algorithm are done in a carefully chosen order (as in RPNI and ALERGIA) to ensure that the algorithm runs in time polynomial in sample size. The algorithm terminates when it has tested for all possible merges. Like all algorithms that learn in the limit, we show that this algorithm learns the correct CTMC given a sufficiently large sample. Our proof that the algorithm learns in the limit relies on a novel method to bound the error probability of our statistical tests. The CTMC that the algorithm learns may be much smaller than the implementation, since it merges all potentially equivalent states, and it only generates the *reachable* portion of the implementation. This can be particularly beneficial in the context of formal verification; the running time and space requirements of verification algorithms depend on the size of the reachable portion of the model. We have implemented our algorithm in Java and experimented by learning some example systems encountered in practice.

The rest of the paper is organized as follows. We give the preliminary definitions and notations in Section 2, followed by the learning algorithm in Section 3. In Section 4 we prove that the learned CTMC converges to the original CTMC in the limit. We report our initial experimental results in Section 5 and conclude in Section 6.

2. Preliminaries

We recall some basic concepts related to CTMCs. Our presentation of this material is slightly non-standard in that we consider CTMCs that have labels both on the edges and the states. In what follows we assume AP to be a finite set of atomic propositions that are used in describing reliability and performance constraints.

Definition 1 *An* Edge Labeled Continuous-time Markov Chain *($CTMC_L$) is a tuple $\mathcal{M} = (S, \Sigma, s_0, \delta, \rho, L)$ where*

- *S is a finite set of states,*
- *Σ is a finite alphabet of edge labels,*
- *$s_0 \in S$ is the initial state,*
- *$\delta\colon S \times \Sigma \to S$ is a partial function which maps a state and an alphabet to the next state,*
- *$\rho\colon S \times \Sigma \to \mathbb{R}_{\geq 0}$ is a function which returns a positive real, called* rate*, associated with the transition. We assume that $\rho(s,a) = 0$ if and only if $\delta(s,a)$ is not defined.*
- *$L\colon S \to 2^{AP}$ is a function which assigns to each state $s \in S$ the set $L(s)$ of atomic propositions that are valid in s.*

A $CTMC_L$ defined as above is deterministic in the sense that for a given state $s \in S$ and an alphabet $a \in \Sigma$ the state reached from s by the edge labeled a is unique if it exists. Intuitively, the probability of moving from state s to state s' along the edge labeled a within time t is given by $(1 - e^{-\rho(s,a)t})$. This probability corresponds to the cumulative probability of an exponential distribution with rate $\rho(s,a)$. For a given state s, if there are more than one alphabet $a \in \Sigma$ such that $\rho(s,a) > 0$ then there is a competition between the transitions. More precisely, for each transition $s \to \delta(s,a)$ from s for which $\rho(s,a) > 0$, a random time t is sampled from the exponential distribution with rate $\rho(s,a)$. Then the transition corresponding to the lowest sampled time is taken. The probability to move from a state s to another state, along the edge a, within t time units i.e. the time sampled for the transition corresponding to $s \to \delta(s,a)$ is minimum, is given by

$$\mathbf{P}(s,a,t) = \frac{\rho(s,a)}{\mathbf{E}(s)}\left(1 - e^{-\mathbf{E}(s)t}\right)$$

where $\mathbf{E}(s) = \sum_{a \in \Sigma} \rho(s,a)$ is the total rate at which any transition from the state s is taken. In other words, the probability of leaving the state s within t time units is $(1 - e^{-\mathbf{E}(s)t})$. This is because the distribution for the minimum time among all edges is exponential with rate $\mathbf{E}(s)$. Thus we can see the probability of moving from a state s along the edge a is the probability of staying at the state s for less than t time units times the probability of taking the edge a. The probability of taking the edge a from state s is thus given by

$$\mathbf{P}(s,a) = \frac{\rho(s,a)}{\mathbf{E}(s)}$$

When $\mathbf{E}(s) = 0$, we define $\mathbf{P}(s,a) = 0$ for every a.

Paths and Probability Space A *path* τ starting at state s is a finite or infinite sequence $l_0 \xrightarrow{(a_1,t_1)} l_1 \xrightarrow{(a_2,t_2)} l_2 \xrightarrow{(a_3,t_3)} \cdots$ such that there is a corresponding sequence $v_0 \xrightarrow{a_1} v_1 \xrightarrow{a_2} v_2 \xrightarrow{a_3} \cdots$ of states with $s = v_0$, $L(v_i) = l_i$, $t_i \in \mathbb{R}_{\geq 0}$, $\delta(v_i, a_{i+1}) = v_{i+1}$ for all i. For a path τ from s, $\tau[s,i] = v_i$ is the i^{th} state of the path and $\eta[\tau, s, i] = t_i$ is the time spent in state v_{i-1}. A *maximal* path starting at state s is a path τ starting at s such that it is either infinite or (if finite) $\mathbf{E}(\tau[s,n]) = 0$, where n is the length of τ. The set of all maximal paths starting at state s is denoted by $Path(s)$; the set of all (maximal) paths in a CTMC $\mathcal{M}$, denoted by $Path(\mathcal{M})$, is taken to be $Path(s_0)$, where s_0 is the initial state.

Let $\pi = l_0 \xrightarrow{a_1} l_1 \xrightarrow{a_2} \cdots \xrightarrow{a_k} l_k$ be a finite sequence such that there is a sequence of states $s_0, s_1, \ldots s_k$ such that s_0 is the initial state, $L(s_i) = l_i$ and $\delta(s_i, a_{i+1}) = s_{i+1}$; $\pi[i]$ is used to denote state s_i in the sequence corresponding to π. Let $I_1, I_2, \ldots I_k$ be non-empty intervals in $\mathbb{R}_{\geq 0}$. Then $C(l_0, (a_1, I_1), l_1, \ldots (a_k, I_k), l_k)$ denotes a *cylinder set* consisting of all paths $\tau \in Path(s_0)$ such that $\tau[s_0, i] = \pi[i]$ (for $i \leq k$) and $\eta[\tau, s_0, i] \in I_i$. Let $\mathcal{B}$ be the smallest σ-algebra on $Path(s_0)$ which contains all the cylinders $C(l_0, (a_1, I_1), l_1, \ldots (a_k, I_k), l_k)$. The probability measure over $\mathcal{B}$ is the unique measure inductively defined as $Path(C(l_0)) = 1$, if $L(s_0) = l_0$ and for $k > 0$ as

$$\begin{aligned} &Prob(C(l_0, (a_1, I_1), l_1, \ldots l_{k-1}, (a_k, I_k), l_k)) \\ &\quad = Prob(C(l_0, (a_1, I_1), l_1, \ldots l_{k-1})) \cdot \\ &\qquad \mathbf{P}(s_{k-1}, a_k) \cdot \left(e^{-\mathbf{E}(s_{k-1})\ell} - e^{-\mathbf{E}(s_{k-1})u}\right) \end{aligned}$$

where $\ell = \inf I_k$ and $u = \sup I_k$, and $s_{k-1} = \pi[k-1]$

3. Learning Edge Labeled CTMCs

The learning problem considered in this paper falls under the category of stochastic grammatical inference [6, 16, 20, 5]. In stochastic grammatical inference, samples are taken from a stochastic language. Given these samples the stochastic language is learned by finding statistical regularity among the samples. The parameters for the different distributions determining the language are also estimated from the relative frequencies of the samples. For most of these learning algorithms it has been shown that they can learn the stochastic language in the *limit* i.e. if the number of samples tends towards infinity then the learned language is the same as the language that generated the sample. All these algorithms essentially follow the same technique: they build a prefix-tree automata which stores exactly the same samples and then they test and merge possibly equivalent states.

We present the algorithm for learning edge labeled CTMCs. We first consider the issue of how to generate and reason about behaviors (visible execution traces) of finite length, given that traditionally the behaviors are assumed to be of infinite length. We then present some concepts that are used in the algorithm. After this we present the algorithm, whose proof of correctness appears in Section 4.

3.1. Generating Samples

In this paper we consider the problem of learning $CTMC_L$ from examples generated by simulating a system under investigation. The way $CTMC_L$ is formally defined in Section 2, all behaviors are *maximal* executions, and maximal executions are typically infinite. This creates a technical difficulty namely what the samples appropriate for learning are. To overcome this problem we define a finitary version of $CTMC_L$ called *Finitary Edge Labeled Continuous-time Markov Chains* ($CTMC_L^f$) which is a $CTMC_L$, with a non-zero stopping probability in any state. This allows one to generate and reason about behaviors of finite length. It is important to note however, that use of $CTMC_L^f$ is merely a technical tool. Our primary goal is to learn the underlying $CTMC_L$ and as we shall see in Proposition 3, we can achieve this by learning the $CTMC_L^f$. Moreover in this effort, the specific value of the stopping probability that we use does not influence the correctness of the result. We present the formal definition of a $CTMC_L^f$.

Definition 2 *A* Finitary Edge Labeled Continuous-time Markov Chain *($CTMC_L^f$) is a pair $\mathcal{F} = (\mathcal{M}, p)$ where $\mathcal{M}$ is a $CTMC_L$ and p denotes the stopping probability in any state s of $\mathcal{M}$*

There exists a trivial surjection $\Theta\colon (\mathcal{M}, p) \mapsto \mathcal{M}$.

A finite sequence $\tau = l_0 \xrightarrow{(a_1,t_1)} l_1 \xrightarrow{(a_2,t_2)} l_2 \cdots \xrightarrow{(a_n,t_n)} l_n$ is a path of the $CTMC_L^f$ $\mathcal{F} = (\mathcal{M}, p)$ starting from a state s iff it is path (not necessarily maximal) of $\mathcal{M}$ starting from s. The set of paths starting from state s is denoted by $Path(s)$. The i^{th} state of path τ from s and the time spent in the i^{th} state are defined similarly, and are denoted by $\tau[s,i]$ and $\eta[\tau, s, i]$, respectively.The σ-field corresponding to a $CTMC_L^f$ is defined analogously to that of a $CTMC_L$. For the path τ from state s, the probability that the $CTMC_L^f$ exhibits such a path is given by

$$\begin{aligned} &Prob_{\mathcal{F}}(\tau, s) = \\ &\quad (1-p) \cdot \mathbf{P}(\tau[s,0], a_1, t_1) \cdot (1-p) \cdot \mathbf{P}(\tau[s,1], a_2, t_2) \\ &\qquad \cdots (1-p) \cdot \mathbf{P}(\tau[s, n-1], a_n, t_n) \cdot p \end{aligned}$$

Given a $CTMC_L$ we extend it to a $CTMC_L^f$ by associating a known probability $p_\perp$ (say $p_\perp = 0.1$) as the stopping probability. The $CTMC_L^f$ thus obtained is then simulated to get a multi-set of finite samples which we treat as the multi-set of examples for learning. In our algorithm we will assume that we are given a finite multi-set of examples from a $CTMC_L$ $\mathcal{M}$ extended with a known stopping probability p to a $CTMC_L^f$. Our goal will be to learn $\mathcal{M}$ from the multi-set of examples.

Note that for a given implemented system, which can be seen as a software program, an example can be generated in the following way. Let s_0 be the initial state of the program. Then add $l_0 = L(s_0)$ to the example sequence. We set a probability $p_0 = 0.1$. With probability p_0 return the current sequence as an example. With probability $1 - p_0$ execute the next instruction of the program. If the execution of the instruction a_i takes time t_i and results in the change of state from s_{i-1} to s_i then add $\stackrel{(a_i,t_i)}{\longrightarrow} L(s_i)$ to the example sequence.

3.2. Definitions

We next define the notations and the concepts that we will use to describe the learning algorithm. Given a $CTMC_L$ $\mathcal{M} = (S, \Sigma, s_0, \delta, \rho, L)$ we can extend the definition of δ as follows:

$\delta(s, \lambda) = s$ where λ is the empty string
$\delta(s, xa) = \delta(\delta(s, x), a)$ where $x \in \Sigma^*$ and $a \in \Sigma$
$\delta(s, a) = \perp$ if $\delta(s, a)$ is not defined
$\delta(s, xa) = \perp$ if $\delta(s, x) = \perp$ or $\delta(\delta(s, x), a)$ is undefined

For a given example $\tau = l_0 \stackrel{(a_1,t_1)}{\longrightarrow} l_1 \stackrel{(a_2,t_2)}{\longrightarrow} l_2 \cdots \stackrel{(a_n,t_n)}{\longrightarrow} l_n$, let $\tau|_\Sigma$ be the string $a_1 a_2 \ldots a_n$. We use $Pr(\tau)$ to denote the set $\{x \mid \exists y\colon xy = \tau|_\Sigma\}$, that is, $Pr(\tau)$ is the set of all prefixes of $\tau|_\Sigma$. Given a multi-set I^+ of examples, let $Pr(I^+)$ be the set $\bigcup_{\tau \in I^+} Pr(\tau)$. If there exists an example $l_0 \stackrel{(a_1,t_1)}{\longrightarrow} l_1 \stackrel{(a_2,t_2)}{\longrightarrow} l_2 \cdots \stackrel{(a_i,t_i)}{\longrightarrow} l_i \cdots \stackrel{(a_n,t_n)}{\longrightarrow} l_n$ in I^+ such that $x = a_1 a_2 \ldots a_i$ then we define $L(x, I^+) = l_i$.

Let $n(x, I^+)$ be the number of $\tau \in I^+$ such that $x \in Pr(\tau)$ and let $n'(x, I^+)$ be $n(x, I^+)$ minus the number of $x \in I^+$. Thus $n(x, I^+)$ counts the number of examples in I^+ for which x is a prefix and $n'(x, I^+)$ is the number of examples in I^+ for which x is prefix and length of x is less than the length of the example. For $\tau = l_0 \stackrel{(a_1,t_1)}{\longrightarrow} l_1 \stackrel{(a_2,t_2)}{\longrightarrow} l_2 \cdots \stackrel{(a_i,t_i)}{\longrightarrow} l_i \cdots \stackrel{(a_n,t_n)}{\longrightarrow} l_n$, if $x = a_1 a_2 \ldots a_{i-1}$ and $a = a_i$, then $\theta(x, a, \tau) = t_i$ and 0 otherwise; in other words, $\theta(x, a, \tau)$ denotes the time spent in the state reached after x in τ. We define $\hat{\theta}(x, I^+)$ and $\hat{p}(x, a, I^+)$ as follows:

$$\hat{\theta}(x, I^+) = \begin{cases} \frac{\sum_{a \in \Sigma} \sum_{\tau \in I^+} \theta(x,a,\tau)}{n'(x, I^+)} \text{ if } n'(x, I^+) > 0 \\ 0 \text{ otherwise} \end{cases}$$

$$\hat{p}(x, a, I^+) = \begin{cases} \frac{n(xa, I^+)}{n'(x, I^+)} \text{ if } n'(x, I^+) > 0 \\ 0 \text{ otherwise} \end{cases}$$

Note that $\hat{\theta}(x, I^+)$ gives an estimate of $1/\mathbf{E}(s)$ where s is the state $\delta(s_0, x)$ and $\hat{p}(x, a, I^+)$ gives an estimate of $\mathbf{P}(s, a)$.

Given a multi-set I^+ of examples we first construct a prefix-tree $CTMC_L^f$ defined as follows.

Definition 3 *The prefix-tree $CTMC_L^f$ for a multi-set of examples I^+ is a $CTMC_L^f$ $PCTMC(I^+) = ((S, \Sigma, s_0, \delta, \rho, L), p)$, where*

1. $S = Pr(I^+)$
2. $s_0 = \lambda$ *(the empty string)*
3. $\delta(x, a) = \begin{cases} xa \text{ if } xa \in S \\ \perp \text{ otherwise} \end{cases}$
4. $\mathbf{E}(x) = 1/\hat{\theta}(x, I^+)$
5. $\mathbf{P}(x, a) = \hat{p}(x, a, I^+)$
6. $\rho(x, a) = \mathbf{P}(x, a)\mathbf{E}(x)$
7. $L(x) = L(x, I^+)$
8. p *is the stopping probability associated with the $CTMC_L^f$ that generated the examples.*

A $PCTMC(I^+)$ is an $CTMC_L^f$ consistent with the examples in I^+ in the sense that for every example in I^+ there is a corresponding path in the $CTMC_L^f$.

The learning algorithm proceeds by generalizing the initial guess, $PCTMC(I^+)$, by merging *equivalent* states. The formal definition of when two states are equivalent is now presented.

Definition 4 *Given a $CTMC_L$ $\mathcal{M} = (S, \Sigma, s_0, \delta, \rho, L)$, a relation $R \subseteq S \times S$ is said to be* stable relation *if and only if for any $s, s' \in S$ such that $(s, s') \in R$, we have*

a) $L(s) = L(s')$

b) $\mathbf{E}(s) = \mathbf{E}(s')$

c) for all $a \in \Sigma$ if there exists $t \in S$ such that $\delta(s, a) = t$ then there exists a $t' \in S$ such that $\delta(s', a) = t'$, $\mathbf{P}(s, a) = \mathbf{P}(s', a)$ and $(t, t') \in R$, and conversely

d) for all $a \in \Sigma$ if there exists $t' \in S$ such that $\delta(s', a) = t'$ then there exists a $t \in S$ such that $\delta(s, a) = t$, $\mathbf{P}(s', a) = \mathbf{P}(s, a)$ and $(t', t) \in R$.

Two states s and s' in $CTMC_L$ M are said to be equivalent *($s \equiv s'$) if and only if there is a stable relation R such that $(s, s') \in R$.*

The correctness of learning algorithm crucially depends on the fact that merging two equivalent states results in a $CTMC_L$ that generates the same distribution. But before we state and prove this formally we make a simple observation about equivalent states.

Lemma 1 *Let $\mathcal{F} = (\mathcal{M}, p)$ be an $CTMC_L^f$ and $s \equiv s'$. τ is a path starting from s iff τ is a path starting from s'. Moreover $Prob_{\mathcal{F}}(\tau, s) = Prob_{\mathcal{F}}(\tau, s')$.*

Proof: Let $\tau = l_0 \stackrel{(a_1,t_1)}{\longrightarrow} l_1 \stackrel{(a_2,t_2)}{\longrightarrow} l_2 \cdots \stackrel{(a_n,t_n)}{\longrightarrow} l_n$ be a path starting from s. There is a sequence $v_0 \stackrel{a_1}{\longrightarrow} v_1 \stackrel{a_2}{\longrightarrow} \cdots v_n$ of states such that $v_0 = s$. Since $s \equiv s'$ and $\delta(s, a_1) = v_1$, there must be a state u_1 such that $\delta(s', a_1) = u_1$ and $u_1 \equiv v_1$. Continuing inductively, we can construct a sequence of states $u_0 \stackrel{a_1}{\longrightarrow} u_1 \stackrel{a_2}{\longrightarrow} \cdots u_n$, such that $u_0 = s'$

and $u_i \equiv v_i$. Hence τ is also a path starting from s'. Furthermore, since $u_i \equiv v_i$, we know that $\mathbf{E}(u_i) = \mathbf{E}(v_i)$, and $\mathbf{P}(u_i, a_{i+1}) = \mathbf{P}(v_i, a_{i+1})$ and hence $Prob_{\mathcal{F}}(\tau, s) = Prob_{\mathcal{F}}(\tau, s')$. For paths starting from s', the argument is symmetric.

Definition 5 *Two $CTMC_L^f$s $\mathcal{F}$ and $\mathcal{F}'$ with initial states s_0 and s_0', respectively, are said to be* equivalent *if $Path(\mathcal{F}) = Path(\mathcal{F}')$ and for every $\tau \in Path(\mathcal{F})$, $Prob_{\mathcal{F}}(\tau, s_0) = Prob_{\mathcal{F}'}(\tau, s_0')$.*

For a $CTMC_L$ $\mathcal{M} = (S, \Sigma, s_0, \delta, \rho, L)$, the *minimal* $CTMC_L$ is defined to be the quotient of $\mathcal{M}$ with respect to the equivalence relation on states. Formally, the minimal $CTMC_L$ is $\mathcal{M}' = (S', \Sigma, s_0', \delta', \rho', L')$ such that

1. S' are the equivalence classes of S with respect to $\equiv$,
2. $s_0' = [s_0]$, the equivalence class of s_0
3. $\delta'([s], a) = [s']$ iff $\delta(s, a) = s'$
4. $\rho'([s], a) = \rho(s, a)$ and
5. $L'([s]) = L(s)$

Observe, that this is well-defined, because of the way $\equiv$ is defined.

Proposition 2 *Let $\mathcal{F} = (\mathcal{M}, p)$ be a $CTMC_L^f$. Then $\mathcal{F}' = (\mathcal{M}', p)$ is equivalent to $\mathcal{F}$ where $\mathcal{M}'$ is the minimal $CTMC_L$ corresponding to $\mathcal{M}$*

Proof: The proof is a straightforward consequence of the definition of the minimal $CTMC_L$ $\mathcal{M}'$. It relies on the observation that for a path τ, $v_0 \xrightarrow{a_1} v_1 \xrightarrow{a_2} \cdots v_n$ is a sequence of states visited in $\mathcal{M}$ iff $[v_0] \xrightarrow{a_1} [v_1] \xrightarrow{a_2} \cdots [v_n]$ in $\mathcal{M}'$. Furthermore, since $\rho'([s], a) = \rho(s, a)$, the probabilities are also the same.

We conclude the section with the observation that for equivalent $CTMC_L^f$s with the same stopping probability, the associated $CTMC_L$s define the same probability space on the set of paths. This next proposition together with Proposition 2 shows that given any $CTMC_L$, we can always construct a smaller equivalent $CTMC_L$ by merging equivalent states, thus providing mathematical justification for our algorithm.

Proposition 3 *Let $\mathcal{F} = (\mathcal{M}, p)$ and $\mathcal{F}' = (\mathcal{M}', p)$ be two $CTMC_L^f$s with the same stopping probability p. Then the probability spaces defined by $\mathcal{M}$ and $\mathcal{M}'$ are the same.*

We skip the proof of this proposition in the interests of space. However, we would like to point out some of the important consequences of Proposition 3. First is that if we learn an $CTMC_L^f$ that has the same stopping probability as the one that was used to generate the samples from the system, then the underlying $CTMC_L$s are also equivalent in terms of the distribution on traces they generate. Second, the specific value of the stopping probability plays no role in proving the correctness of our learning algorithm. It may have an effect in terms of the length of traces produced and the number of traces needed to learn. The right choice of the stopping probability is thus one that is determined by the empirical constraints that one is working in.

3.3. Learning Algorithm

algorithm *learnCTMC*
Input: I^+ : a multi-set of examples
α : confidence level
Output: $CTMC_L$
begin
  $A \leftarrow PCTMC(I^+)$
  for $i = 2$ to $|A|$ do
    for $j = 1$ to $i - 1$ do
      if *compatible*(s_i, s_j, α, I^+) then
        $A \leftarrow$ *merge*(A, s_i, s_j)
        $A \leftarrow$ *determinize*(A)
        exit j-loop
      endif
  return A
end

Figure 1. Algorithm to learn $CTMC_L^f$

The algorithm for $CTMC_L$ learning, described in Figure 1, first constructs the prefix-tree $CTMC_L^f$ $A = PCTMC(I^+)$ from the multi-set of examples I^+. We assume that the states in A are ordered in lexicographic order.[1] Let $|A|$ be the number of states in A. The algorithm then tries to merge pairs of states in A that are equivalent in a quadratic loop, i.e. for all i from 1 to $|A|$ the algorithm tries to merge s_i with the states $s_1, s_2, \ldots, s_{i-1}$ in that order. If two states s_i and s_j are equivalent they are merged using the method *merge*(A, s_i, s_j). The smallest state in a block of merged states is used to represent the whole block. After every merge of state s_i and s_j the resulting $CTMC_L^f$ may be non-deterministic. However, equivalence of s_i and s_j implies that each successor of s_i is equivalent to the corresponding successor of s_j. This means that those successors should also get merged. To ensure this the method *determinize*(A) described in Figure 2 is invoked which removes the non-determinism in A by a sequence of merges. After every merge the probabilities $\mathbf{P}(s, a)$ and the rates $\mathbf{E}(s)$ are re-computed for every state as there is more information available at every state. The algorithm stops when no more merging is possible.

1 For $\Sigma = \{a, b\}$, the lexicographic ordering is $\lambda, a, b, aa, ab, ba, bb, aaa, \ldots$

algorithm *determinize*
Input: A
Output: $CTMC_L$
begin
  while($\exists s, a \in A\colon s', s'' \in \delta(s,a)$) do
    $A \leftarrow$ *merge*(A, s', s'')
  return A
end

Figure 2. *determinize* removes non-determinism

Now the observations Proposition 2 and 3 together suggest that the above algorithm would be correct if indeed we could test for equivalence of two states. This, however, is not the case, as A is built from experimental data. However, we approximately check the equivalence of two states recursively through statistical hypothesis testing [15, 18]. We say that two states s_i and s_j are compatible, denoted by $s_i \approx s_j$, if $L(s) = L(s')$, $\mathbf{E}(s) \sim \mathbf{E}(s')$, for all $a \in \Sigma$, $\mathbf{P}(s_i, a) \sim \mathbf{P}(s_j, a)$, and $\delta(s_i, a) \approx \delta(s_j, a)$, where $\mathbf{E}(s) \sim \mathbf{E}(s')$ means that $\mathbf{E}(s)$ and $\mathbf{E}(s')$ are equal within some statistical uncertainty and similarly for $\mathbf{P}(s_i, a) \sim \mathbf{P}(s_j, a)$. The decision for compatibility is made using the function *compatible* described in Figure 3.

algorithm *compatible*
Input: x, y, I^+, α
Output: boolean
begin
  if $L(x, I^+) \neq L(y, I^+)$ then
    return FALSE
  if *differentExpMeans*$(\hat{\theta}(x, I^+), n'(x, I^+),$
                $\hat{\theta}(y, I^+), n'(y, I^+), \alpha)$ then
    return FALSE
  for $\forall a \in \Sigma$
    if *differentBerMeans*$(\hat{p}(x, a, I^+), n(xa, I^+),$
                $\hat{p}(y, a, I^+), n(ya, I^+), \alpha)$ then
      return FALSE
    if not *compatible*$(\delta(x, a), \delta(y, a), I^+, \alpha)$ then
      return FALSE
  endfor
  return TRUE
end

Figure 3. *compatible* checks if two two states are approximately equivalent

The check for $\mathbf{E}(s_i) \sim \mathbf{E}(s_j)$ is performed by the function *differentExpMean*, described in Figure 4, which uses statistical hypothesis testing. The function actually checks if the means $1/\mathbf{E}(s_i)$ and $1/\mathbf{E}(s_j)$ of two exponential distributions are different. Given two exponential distributions with means θ_1 and θ_2 we want to check if $\theta_1 = \theta_2$ against the fact fact that $\theta_1 \neq \theta_2$. This is equivalent to checking $\frac{\theta_1}{\theta_2} = 1$ against the fact that $\frac{\theta_1}{\theta_2} \neq 1$. In statistical terms we call $\frac{\theta_1}{\theta_2} = 1$ as the null hypothesis (denoted by H_0) and $\frac{\theta_1}{\theta_2} \neq 1$ as the alternate hypothesis (denoted by H_a). To test the hypothesis H_0 against H_a we draw n_1 samples, say $x_1, x_2, \ldots, x_{n_1}$, from the exponential distribution with mean θ_1 and n_2 samples, say $y_1, y_2, \ldots, y_{n_2}$, from the exponential distribution with mean θ_2. We estimate θ_1 and θ_2 by $\hat{\theta_1} = \frac{\sum_{i=1}^{n_1} x_i}{n_1}$ and $\hat{\theta_2} = \frac{\sum_{i=1}^{n_2} y_i}{n_2}$ respectively. Then we use the ratio $\frac{\hat{\theta_1}}{\hat{\theta_2}}$ to check H_0 against H_a as follows:

We can say that $x_1, x_2, \ldots, x_{n_1}$ are random samples from the random variables $X_1, X_2, \ldots, X_{n_1}$ where each X_i has an exponential distribution with mean θ_1. Similarly, $y_1, y_2, \ldots, y_{n_2}$ are random samples from the random variables $Y_1, Y_2, \ldots, Y_{n_2}$ where each Y_i has an exponential distribution with mean θ_2. Then it can be shown by methods of moment generating function that the random variables $\frac{2\sum X_i}{\theta_1}$ and $\frac{2\sum Y_i}{\theta_2}$ have $\chi^2(2n_1)$ and $\chi^2(2n_2)$ distributions respectively. This implies that the ratio $\frac{(2\sum X_i)/(2n_1\theta_1)}{(2\sum Y_i)/(2n_2\theta_2)}$ or $\frac{\sum X_i/n_1}{\theta_1} / \frac{\sum Y_i/n_2}{\theta_2}$ has F distribution with $(2n_1, 2n_2)$ degrees of freedom. Assuming that H_0 holds $\frac{\sum X_i/n_1}{\sum Y_i/n_2}$ has $F(2n_1, 2n_2)$ distribution. Let us introduce the random variables Θ_1 and Θ_2 where $\Theta_1 = \frac{\sum X_i}{n_1}$ and $\Theta_2 = \frac{\sum Y_i}{n_2}$. Our experimental value of $\frac{\hat{\theta_1}}{\hat{\theta_2}}$ gives a random sample from the random variable $\frac{\Theta_1}{\Theta_2}$. Let the random variable $Z = \frac{\Theta_1}{\Theta_2}\frac{\theta_2}{\theta_1}$. Then Z has F distribution with $(2n_1, 2n_2)$ degrees of freedom. Given $\theta_1 = \theta_2$, from Chebyshev's inequality, we get

$$Prob\left[\left|\frac{\Theta_1}{\Theta_2} - \mu\right| \geq \frac{\sigma}{\sqrt{\alpha}}\right] = Prob\left[|Z - \mu| \geq \frac{\sigma}{\sqrt{\alpha}}\right] \leq \alpha$$

where $\mu = \frac{n_2}{n_2 - 1}$ is the mean of $F(2n_1, 2n_2)$ and $\sigma = \sqrt{\frac{n_2^2(n_1+n_2-1)}{n_1(n_2-1)^2(n_2-2)}}$ its standard deviation. Thus, taking $\check{r} = \mu - \frac{\sigma}{\sqrt{\alpha}}$ and $\hat{r} = \mu + \frac{\sigma}{\sqrt{\alpha}}$, we get

$$Prob\left[\check{r} \leq \frac{\Theta_1}{\Theta_2} \leq \hat{r}\right] > 1 - \alpha \qquad (1)$$

If $\frac{\hat{\theta_1}}{\hat{\theta_2}} > 1$ then we calculate the probability of our observation given $\theta_1 = \theta_2$, called the p-value, as

$$p\text{-value} = Prob\left[\frac{\Theta_1}{\Theta_2} > \frac{\hat{\theta_1}}{\hat{\theta_2}}\right] = Prob\left[Z > \frac{\hat{\theta_1}}{\hat{\theta_2}}\right]$$

Similarly, if $\frac{\hat{\theta_1}}{\hat{\theta_2}} < 1$, the p-value is given by

$$p\text{-value} = Prob\left[\frac{\Theta_1}{\Theta_2} < \frac{\hat{\theta_1}}{\hat{\theta_2}}\right] = Prob\left[Z < \frac{\hat{\theta_1}}{\hat{\theta_2}}\right]$$

If the calculated p-value in both cases together is less than α we say we have enough evidence to reject the null hypoth-

esis $\theta_1 = \theta_2$. This is equivalent to say that we reject H_0 if $\frac{\hat{\theta_1}}{\hat{\theta_2}} \notin [\check{r}, \hat{r}]$.

---

algorithm *differentExpMeans*
Input: $\hat{\theta_1}, n_1, \hat{\theta_2}, n_2, \alpha$
Output: boolean
begin
  if $n_1 = 0$ or $n_2 = 0$ then
    return FALSE
  return $\frac{\hat{\theta_1}}{\hat{\theta_2}} \notin [\check{r}, \hat{r}]$
end

Figure 4. *differentExpMeans* checks if two estimated exponential means are different; the parameter α is used in calculating $\check{r}$ and $\hat{r}$

The check for $\mathbf{P}(s_i, a) \sim \mathbf{P}(s_j, a)$ is performed by the function *differentBerMeans* (see Figure 5) using Hoeffding bounds similar to that in [6]. The method checks if the means p_1 and p_2 of two Bernoulli distributions are statistically different or not. If f_1 tries are 1 out of n_1 tries from a Bernoulli distribution with mean p_1 and f_2 tries are 1 out of n_2 tries from a Bernoulli distribution with mean p_2, then we say that p_1 and p_2 are statistically same if

$$\left| \frac{f_1}{n_1} - \frac{f_2}{n_2} \right| < \sqrt{\frac{1}{2} \log \frac{2}{\alpha}} \left(\frac{1}{\sqrt{n_1}} + \frac{1}{\sqrt{n_2}} \right)$$

Note that it is possible to use other tests, such as multinomial test [16], to compare two means of Bernoulli distributions.

---

algorithm *differentBerMeans*
Input: $\hat{p_1}, n_1, \hat{p_2}, n_2, \alpha$
Output: boolean
begin
  if $n_1 = 0$ or $n_2 = 0$ then
    return FALSE
  return $|\hat{p_1} - \hat{p_2}| > \sqrt{\frac{1}{2} \log \frac{2}{\alpha}} \left(\frac{1}{\sqrt{n_1}} + \frac{1}{\sqrt{n_2}} \right)$
end

Figure 5. *differentBerMeans* checks if two estimated Bernoulli means are different

---

3.4. Complexity

The worst case running complexity of the algorithm is cubic in the sum of the length of all samples. However, in our experiments we found that the running time grows almost linearly with the sum of length of sample lengths. The parameter α influences the size of the sample needed for converging on the right model. The exact dependence of sample size on α is an open problem that needs investigation.

4. Learning in the Limit

In order to prove correctness of our algorithm, we need to show that the $CTMC_L$ that the learning algorithm produces is eventually equivalent to the model that was used to generate the samples. Our proof proceeds in two steps. First we show that the learning algorithm will eventually be presented what is usually called a *structurally complete* sample. A structurally complete sample I^+ is a multi-set of traces such that the traces visit every (reachable) state and every transition. More formally, for every state s of the target $CTMC_L$, there is a trace $\tau \in I^+$ such that s is one of the states visited when trace τ was produced, and for every (reachable) transition (s, a) there is a trace $\tau \in I^+$ such that (s, a) is traversed by τ. Observe that if $I^+ \subseteq I'^+$ and I^+ is structurally complete then I'^+ is also structurally complete. The second step of the proof involves showing that if we keep adding samples to a structurally complete set, then we will eventually learn the right $CTMC_L$. These two steps together show that our algorithm will learn the target $CTMC_L$ in the limit [8].

The first thing to observe that for any $CTMC_L$ $\mathcal{M}$ there is a finite structurally complete sample set. Let Γ be a structurally complete sample set and let $\mathcal{F} = (\mathcal{M}, p)$ be a $CTMC_L^f$ (with any stopping probability p). Now observe that for any $\tau \in \Gamma$, $p = Prob_{\mathcal{F}}(\tau, s_0)$ is finite and non-zero. Thus, the probability that τ is not among the first k samples generated by $\mathcal{F}$ is $(1-p)^k$, and this tends to 0 as k increases. Hence, every string in Γ is eventually generated, and so the sample given to the learning algorithm is eventually structurally complete.

The main challenge in the proof of correctness is to show that once we have a structurally complete sample, we will eventually learn the right $CTMC_L$. In what follows, we simply assume that whenever we refer to a sample I^+, we mean that I^+ is structurally complete. Observe that for a (structurally complete) sample I^+, the right $CTMC_L$ is one that results from merging equivalent states of $PCTMC(I^+)$. However, since we can only check compatibility (and not exact equivalence) the only errors the algorithm makes can result when we check the compatibility of two states. There are two types of errors in this context.

1. Type I error : *compatibility* returns false when two states are actually equivalent, and
2. Type II error : *compatibility* returns true when two states are not equivalent

Our goal is to reduce these two errors as much as possible. We show that as $s = |I^+|$ goes to infinity, the global contribution of these two errors tend to zero. Observe that, if t is

the number of states in $PCTMC(I^+)$, then t cannot grow as fast as s does. If m be the number of states in the target $CTMC_L^f$, then the number of merges performed by the algorithm before giving the correct $CTMC_L^f$ is $t-m$. Further recall that the p-value of the tests performed by the functions *differentExpMeans* and *differentBerMeans* is at most α. Hence, global Type I error, e_α is bounded by $\alpha(|\Sigma|+1)t$. This error can be made negligible and independent of the size of the $PCTMC(I^+)$ by taking $\alpha = kt^{-1}$ for some very small constant k.

Thus, by making α small we can ensure that the learning algorithm always merges equivalent states. Then the errors of the learning algorithm can be confined to those resulting from merging inequivalent states. In the absence of Type I errors, the learning algorithm always outputs a $CTMC_L^f$, whose states form a partition of the target $CTMC_L^f$. Thus an upper bound on Type II errors is given by the probability that an error occurs when comparing two states of the target $CTMC_L^f$. Taking β to be the probability of merging two non-equivalent states, we get the Type II error $e_\beta \leq \frac{1}{2}\beta m(m-1)(|\Sigma|+1)$. Thus if we show that β tends to 0 as the sample size grows, then we know that the algorithm will eventually not make any errors.

Observe that the probability of merging a pair of non-equivalent states is bounded by the probability of either *differentExpMeans* or *differentBerMeans* returning TRUE when the actual means are different. Hence, in order to show that the learning algorithm eventually gives the right answer, we need to show that the probability that *differentExpMeans* and *differentBerMeans* make an error when the means are different tends to 0. We will consider each of the procedures separately and bound their error.

Case *differentExpMeans*: Assume that $r = \frac{\theta_2}{\theta_1} \neq 1$, i.e., $\mathbf{E}(s_i) \neq \mathbf{E}(s_j)$. Recall that for a given α, taking $\check{r} = \mu - \frac{\sigma}{\sqrt{\alpha}}$ and $\hat{r} = \mu + \frac{\sigma}{\sqrt{\alpha}}$, where μ is the mean and σ the standard deviation of $F(2n_1, 2n_2)$, we say that an observation $\frac{\hat{\theta_1}}{\hat{\theta_2}}$ provides evidence for $\mathbf{E}(s_i) \sim \mathbf{E}(s_j)$ when $\frac{\hat{\theta_1}}{\hat{\theta_2}} \in [\check{r}, \hat{r}]$. Now, we have

$$\begin{aligned} Prob\left[\check{r} \leq \tfrac{\Theta_1}{\Theta_2} \leq \hat{r}\right] &= Prob\left[\check{r}r \leq \tfrac{\Theta_1}{\Theta_2}\tfrac{\theta_2}{\theta_1} \leq \hat{r}r\right] \\ &= Prob\left[\check{r}r \leq Z \leq \hat{r}r\right] \end{aligned}$$

where, Z has distribution $F(2n_1, 2n_2)$. Once again by Chebyshev's inequality, we know that

$$Prob\left[|Z-\mu| \geq \frac{\sigma}{\sqrt{\beta}}\right] \leq \beta$$

Thus, if $r\check{r} \geq \mu + \frac{\sigma}{\sqrt{\beta}}$ or $r\hat{r} \leq \mu - \frac{\sigma}{\sqrt{\beta}}$ then

$$Prob\left[\check{r} \leq \frac{\Theta_1}{\Theta_2} \leq \hat{r}\right] = Prob\left[\check{r}r \leq Z \leq \hat{r}r\right] \leq \beta$$

Taking

$$\beta = \left(\frac{\sigma\sqrt{\alpha}}{|r-1|\mu\sqrt{\alpha} + r\sigma}\right)^2$$

we observe that if $r < 1$ then $r\check{r} \geq \mu + \frac{\sigma}{\sqrt{\beta}}$ and if $r \geq 1$ then $r\hat{r} \leq \mu - \frac{\sigma}{\sqrt{\beta}}$. Finally, plugging in the values of μ, σ and α, we observe that β tends to 0.

Case *differentBerMeans*: Let $\mathbf{P}(s_i, a) = p_1 \neq p_2 = \mathbf{P}(s_j, a)$. Let F_1 be a random variable that is the mean of n_1 Bernoulli trials with mean p_1 and F_2 the mean of n_2 Bernoulli trials with mean p_2. Recall that we say $\mathbf{P}(s_i, a) \sim \mathbf{P}(s_j, a)$ if some observation $\hat{f}_1$ of variable F_1 and observation $\hat{f}_2$ of F_2 are such that $|\hat{f}_1 - \hat{f}_2| < \epsilon$ where

$$\epsilon = \sqrt{\frac{1}{2}\log\frac{2}{\alpha}}\left(\frac{1}{\sqrt{n_1}} + \frac{1}{\sqrt{n_2}}\right)$$

We will try to bound the probability that this happens. Observe that $E(F_1 - F_2) = p_1 - p_2$ and

$$\begin{aligned} \mathrm{Var}(F_1 - F_2) &= \mathrm{Var}(F_1) + \mathrm{Var}(F_2) \\ &= \tfrac{p_1(1-p_1)}{n_1} + \tfrac{p_2(1-p_2)}{n_2} \geq \tfrac{1}{4n_1} + \tfrac{1}{4n_2} \end{aligned}$$

Now $\beta = Prob(|F_1 - F_2| < \epsilon) < Prob(|(F_1 - F_2) - (p_1 - p_2)| > |p_1 - p_2| - \epsilon)$. By Chebyshev's inequality, $\beta \leq B$ where

$$B = \begin{cases} \frac{\mathrm{Var}(F_1-F_2)}{(|p_1-p_2|-\epsilon)^2} & \text{if } \epsilon < |p_1 - p_2| \text{ and} \\ & \mathrm{Var}(F_1 - F_2) < ((p_1 - p_2) - \epsilon)^2 \\ 1 & \text{otherwise} \end{cases}$$

First observe that n_1 and n_2 grow linearly as $s = |I^+|$ increases, and so even if α is kt^{-1} as needed in order to eliminate Type I errors, since t is bounded by s, ϵ tends to 0 as s grows. Next $\mathrm{Var}(F_1 - F_2)$ also tends to 0 as s grows. Hence B vanishes with s, proving that in the limit Type II errors are eliminated.

5. Tool and Experiments

We have implemented the learning algorithm in Java as a sub-component of the tool VESTA (Verification based on Statistical Analysis).[2] The tool takes a multi-set of examples generated from the simulation of a system having an unknown $CTMC_L$ model. Based on these examples the tool learns the underlying $CTMC_L$ for a given value of α. The learned model can be used for verification of CSL formulas either using the statistical model-checker of VESTA [21] or other model-checkers such as PRISM [17], ETMCC [12], Ymer [24] etc. We tested the performance of our tool on several $CTMC_L$ models. For each $CTMC_L$ model we performed discrete-event simulation to get a large number of examples and then learned a $CTMC_L$ based on these examples. Finally we checked if the learned $CTMC_L$ is equivalent as the original $CTMC_L$, that generated the examples. We found that the learned $CTMC_L$ is equivalent to the original $CTMC_L$ in all our experiments provided that the number of samples is large enough. In all our experiments we assumed that the set of atomic propositions at any state

2 Available from `http://osl.cs.uiuc.edu/~ksen/vesta/`

are same. This is assumed to show the working of the statistical tests. If we take atomic propositions into consideration then the learning becomes faster because atomic propositions will be sufficient in distinguishing certain states. We next report the results of our experiments performed on a Pentium III 1.2GHz laptop with 620 MB SDRAM.

Symmetric CTMC The $CTMC_L$ in Figure 6, which is carefully selected, contains four states. The probability of taking the edges labeled a and b from the states 0 and 2 are same; however, the total rates at which transitions take place from the two states are different. Therefore, to distinguish these two states comparison of the total rates is required. Similar is the case for states 1 and 3. On the other hand, the total rates for the states 0 and 1 are same; however, the probability of taking the edges a and b are different which is used to distinguish the two states. The same is true for the states 2 and 3. Thus this example shows the effectiveness of both the functions *differentBerMeans* and *differentExpMeans* during learning. We use the $CTMC_L$ to generate samples which are then used to learn a $CTMC_L$. We found that for more than 600 samples and $\alpha = 0.0001$ the $CTMC_L$ learned is same as the original $CTMC_L$.

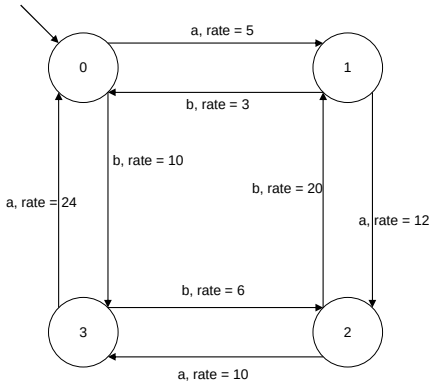

Figure 6. Symmetric CTMC

Triple Modular Redundant System Our next example is the $CTMC_L$ in Figure 7 representing the model of a *Triple Modular Redundant System* (TMR). The example is taken from [10, 2]. We ignore the atomic propositions that are true at each state to show the effectiveness of our statistical test. Although we generated the samples through the discrete event simulation of the $CTMC_L$ in Figure 7, we can as well assume that the samples are coming from the actual running system. In figure 7 we plot the average number of states in the learned $CTMC_L$ and time taken in learning against the number of samples used in learning. The number of states converges to five when the sample is large enough. The time taken for learning grows almost linearly with the number of samples. With $\alpha = 0.0001$ and 1000 samples the algorithm learns the same $CTMC_L$ in less than 2 seconds. For small number of samples, due to lack of sufficient information, the algorithm tends to generalize more resulting in less number of states in the learned $CTMC_L$.

Tandem Queuing Network In this more practical example we considered a M/Cox$_2$/1-queue sequentially composed with a M/M/1-queue. This example is taken from [13]. For $N = 3$, where N denotes the capacity of the queues, the algorithm learned a $CTMC_L$ model with 15 states. However, the number of samples required in this case to learn the underlying $CTMC_L$ is quite large (around 20,000). This particular experiment suggests that learning underlying $CTMC_L$ for large systems may require a large number of samples. Therefore, a more effective technique would be to verify the approximate model learnt from small number of samples. However, because the model learnt is approximate, the result of verification would also be approximate. This suggests that the confidence in verification should be quantified reasonably. How to do such quantification remains an open problem.

6. Conclusion and Future Work

We have presented a novel machine-learning algorithm to learn the underlying edge-labeled continuous-time Markov chains of deployed stochastic systems for which we do not know the model before-hand. An important aspect of the learning algorithm is that it can learn a formal stochastic model from the traces generated during testing or executing the deployed system. The learnt $CTMC_L$ can be used to verify the deployed systems using existing probabilistic model-checking tools. Moreover, one can also check if the learnt model is F-bisimilar [2, 3] to the model given in a specification. This allows us to check if the deployed system correctly implements a specification with respect to a set of formulas F. Finally, we provide an implementation of the algorithm which can be used with various other tools.

One of the limitations of our work is that it may not scale up for systems having large underlying CTMC model. Therefore, one needs to develop techniques that can perform approximate verification as the model is learnt. The accuracy of such verification technique should increase with the increase in the number of samples. The difficult part in developing such an approach is to correctly quantify the confidence (or accuracy) in verification. Such a technique will make verification of "black-box" systems a very practical approach that can co-exist with testing based on discrete event simulation.

Acknowledgement

The work is supported in part by the DARPA IPTO TASK Award F30602-00-2-0586, the DARPA/AFOSR MURI Award F49620-02-1-0325, the ONR Grant N00014-02-1-0715, and the Motorola Grant MOTOROLA RPS #23 ANT. Our work has benefitted considerably from our collaboration with Abhay Vardhan

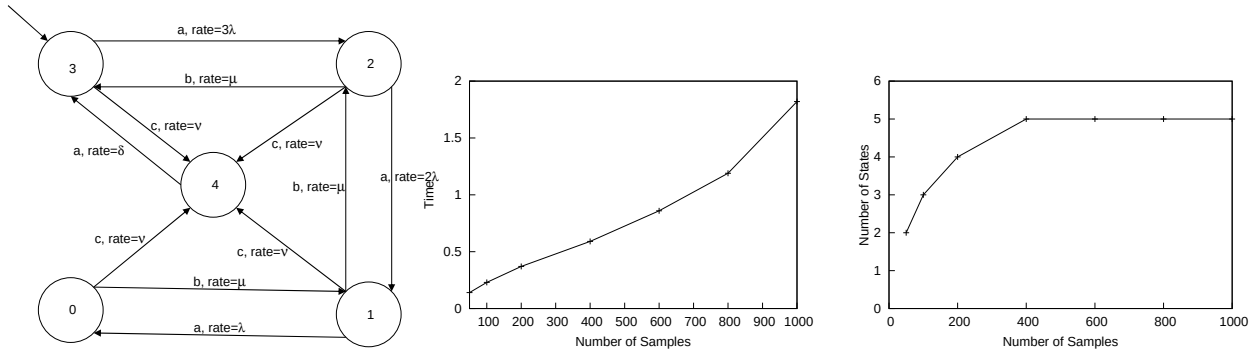

Figure 7. Learning TMR

on "learning to verify" framework for verifying infinite state systems. We would like to thank Tom Brown for giving us feedback on a previous version of this paper.

References

[1] D. Angluin. Learning regular sets from queries and counterexamples. *Infor. and Comp.*, 75(2):87–106, 1987.

[2] C. Baier, B. Haverkort, H. Hermanns, and J. Katoen. Model checking continuous-time Markov chains by transient analysis. In *Computer Aided Verification (CAV'00)*, volume 1855 of *LNCS*, pages 358–372. Springer, 2000.

[3] C. Baier, J. Katoen, H. Hermanns, and B. Haverkort. Simulation for continuous-time Markov chains. In *13th International Conference on Concurrency Theory (CONCUR'02)*, volume 2421 of *LNCS*, pages 338–354. Springer, 2002.

[4] G. Balbo, G. Conte, S. Donatelli, G. Franceschinis, and M. A. Marsan. *Modelling with Generalized Stochastic Petri Nets*. John Wiley & Sons, 1995.

[5] L. Baum, T. Petrie, G. Soules, and N. Weiss. A maximization technique occuring in the statistical analysis of probabilistic functions of Markov chains. *Annals of Mathematical Statistics*, 41(1):164–171, 1970.

[6] R. C. Carrasco and J. Oncina. Learning stochastic regular grammars by means of a state merging method. In *International Colloquium Grammatical Inference and Applications (ICGI'94)*, volume 862 of *LNCS*. Springer, 1994.

[7] J. M. Cobleigh, D. Giannakopoulou, and C. S. Pasareanu. Learning assumptions for compositional verification. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS'03)*, volume 2619 of *LNCS*, pages 331–346.

[8] E. M. Gold. Language identification in the limit. *Information and Control*, 10:447–474, 1967.

[9] A. Groce, D. Peled, and M. Yannakakis. Adaptive model checking. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS'02)*, volume 2280 of *LNCS*, pages 357–371, 2002.

[10] B. Haverkort. *Performance of Computer Communication Systems: A Model-Based Approach*. Wiley, 1998.

[11] H. Hermanns, U. Herzog, and J. Katoen. Process algebra for performance evaluation. *Theoretical Computer Science*, 274(1–2):43–87, 2002.

[12] H. Hermanns, J. P. Katoen, J. Meyer-Kayser, and M. Siegle. A tool for model-checking Markov chains. *Software Tools for Technology Transfer*, 4(2):153–172, 2003.

[13] H. Hermanns, J. Meyer-Kayser, and M. Siegle. Multi-terminal binary decision diagrams to represent and analyse continuous-time Markov chains. In *Workshop on the Numerical Solution of Markov Chains (NSMC'99)*, 1999.

[14] J. Hillston. *A Compositional Approach to Performance Modelling*. Cambridge University Press, 1996.

[15] R. V. Hogg and A. T. Craig. *Introduction to Mathematical Statistics*. Macmillan, New York, NY, USA, 1978.

[16] C. Kermorvant and P. Dupont. Stochastic grammatical inference with multinomial tests. In *Grammatical Inference: Algorithms and Applications*, volume 2484 of *Lecture Notes in Artificial Intelligence*. Springer, 2002.

[17] M. Z. Kwiatkowska, G. Norman, and D. Parker. PRISM: Probabilistic symbolic model checker. In *Computer Performance Evaluation, Modelling Techniques and Tools (TOOLS'02)*, volume 2324 of *LNCS*, pages 200–204.

[18] W. Nelson. *Applied Life Data Analysis*. Wiley, 1982.

[19] J. Oncina and P. Garcia. Inferring regular languages in polynomial update time. In *Pattern Recognition and Image Analysis*, volume 1 of *Series in Machine Perception and Artificial Intelligence*, pages 49–61. 1992.

[20] D. Ron, Y. Singer, and N. Tishby. On the learnability and usage of acyclic probabilistic finite automata. *Journal of Computer and System Sciences*, 56(2):133–152, 1998.

[21] K. Sen, M. Viswanathan, and G. Agha. Statistical model checking of black-box probabilistic systems. In *16th conference on Computer Aided Verification (CAV'04)*, LNCS (To Appear). Springer, July 2004.

[22] W. J. Stewart. *Introduction to the Numerical Solution of Markov Chains*. Princeton, 1994.

[23] W. Wei, B. Wang, and D. Towsley. Continuous-time hidden Markov models for network performance evaluation. *Performance Evaluation*, 49(1–4):129–146, September 2002.

[24] H. L. S. Younes, M. Kwiatkowska, G. Norman, and D. Parker. Numerical vs. statistical probabilistic model checking: An empirical study. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS'04)*, volume 2988 of *LNCS*. Springer, 2004.