Accelerating Markovian Analysis of Asynchronous Systems using String-based State Compression †

Aiguo Xie and Peter A. Beerel

EE-Systems Department
University of Southern California
Los Angeles, CA 90089

Abstract

This paper presents a methodology to speed up the stationary analysis of large Markov chains that model asynchronous systems. Instead of directly working on the original Markov chain, we propose to analyze a smaller Markov chain obtained via a novel technique called string-based state compression. Once the smaller chain is solved, the solution to the original chain is obtained via a process called expansion. The method is especially powerful when the Markov chain has a small feedback vertex set, which happens often in asynchronous systems. Experimental results show that the method can yield reductions of more than an order of magnitude in run time and facilitate the analysis of larger systems than possible using traditional techniques.

1 Introduction

Driven by market demands for low-power and high-performance, tools to estimate power and performance of a system have become particularly important. In an asynchronous system, the randomness caused by varying input data rate and data processing delays, makes these estimation tasks particularly challenging. The work in this paper is applicable to both areas, however, to simplify the exposition we focus on performance analysis.

There are currently two main categories of approaches for performance analysis. Approaches belonging to the first category use *bounding techniques* to analyze system models with delays specified by upper and lower bounds. Hulgaard et al. use a delay-annotated Petri-net model to obtain upper/lower bounds for the time-separations between arbitrarily indexed occurrences of two events [1, 2]. Also belonging to this category are Greenstreet and Steiglitz's work [3] on throughput bounds of pipelines and Ebergen and Berks' work [4] on bounding the response time for FIFOs. Approaches in the second category target metrics that are averaged over a long time period, e.g., average time separation between two events, average throughput, and average latency. These approaches analyze probabilistic models which assume arbitrarily distributed delays and allow arbitrary system behaviors such as unfair choices and conflicts. Kudva et al. [5] and Xie and Beerel's [6] work belong to this category. It should be noted that in the case where all delays are fixed, bounding techniques are often able to give exact average performance metrics for some classes of systems (e.g., those modeled by marked graphs or extended marked graphs) [1, 3, 7, 8, 9]. Average performance metrics may also be derived directly for some systems (e.g., with FIFO-like structures) if all delays are exponentially distributed so that classic queuing theory can be applied [3].

Bounding techniques are often faster than probabilistic techniques, but the average performance measure may be more useful to guide design. This is particularly true if the bound is quite wide and the mean lies close to one end of the bound, which can easily occur in practice.

The most time consuming step in probabilistic approaches to performance analysis is Markovian analysis. In Kudva et al.'s approach, systems are specified in a generalized timed Petri-net model [10] whereas in Xie and Beerel's approach, they are specified using an SMV [11] based discrete-time model. Both approaches convert their models into finite state Markov chains, i.e., random processes with short memory (whose current state completely determines its future

†This work is funded in part by a NSF CAREER Award MIP-9502386, a gift from the Intel Corporation, and a research seed grant from the James H. Zumberge Faculty Research and Innovation Fund at USC.

evolution). Here, states represent the valuations of system signals plus delay signals (corresponding to transition firing duration in Kudva et al.'s model and auxiliary variables capturing the probabilistic interval delays in Xie and Beerel's model). This conversion is followed by reachability analysis which finds all reachable states of the system. The derived Markov chain is then solved for the stationary distribution, the long run probabilities of all reachable states. Xie and Beerel further use this stationary distribution to compute the *sojourn time* [12] of a subset of the state space from which the *average time separation of events* can be obtained. Fortunately, this latter part of analysis is computationally inexpensive.

The bottleneck in the Markovian analysis is the computation of the stationary distribution of the derived Markov chain. The problem is equivalent to solving a linear system with as many unknowns as the number of states in the Markov chain. The generally preferred technique is the well known *Power method* which iterates from some initial distribution (or initial guess) until it converges according to some user-specified accuracy requirement. Since the Markov chains resulting from real designs can possess huge state spaces, e.g, with hundreds of thousands of states, the Power method can take intolerably long before getting a good approximation of the stationary distribution. A promising way to speed up this phase of the analysis is to use *symbolic techniques* [6, 13]. Although it is then possible to handle some systems with hundreds of signals, it can still take hours to complete. Therefore, techniques that can significantly speed up Markovian analysis are of great demand.

There are many proposed techniques to speed up iterative approaches such as the Power method for large linear systems. Examples are *pre-conditioning* [14] and *orthogonal multi-vector iteration* [15]. It is unclear whether these sophisticated techniques are likely to give a significant improvements, however, because they impose significant overhead. In addition, there is an ongoing research trend in probability theory on the *lumpability* of Markov chains [16, 17]. The basic idea in lumping states is to explore the *similarity* (if any) among the behavior of the Markov chain when it is in different states. A subset of states can be lumped if when treated as a single state, the resulting random process still has the Markov property and thus may be analyzed as a smaller Markov chain. However, the condition for a subset of states to be lumpable is generally very restrictive. Consequently, it is unlikely for real designs to possess sufficient regularity of this kind to benefit from the incorporation of the lumpability theory currently available [18].

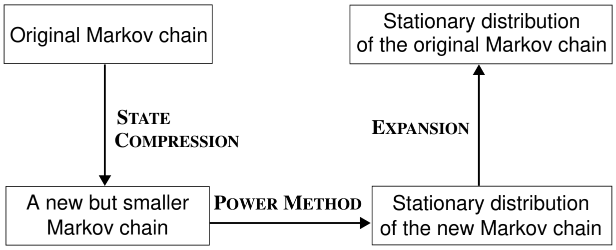


Figure 1: Block diagram of our method.

The method we propose here takes advantage of a notion of *limited feedback* inherent in many asynchronous systems in which the occurrence of some states are always preceeded by the occurrences of some other states. More precisely, we partition the (reachable) state space S into two subsets, B and $\overline{B}$, such that any two consecutive occurrences of any state in $\overline{B}$ is always interleaved by at least one occurrence of some state in B. In graph terminology, B is a *feedback vertex set* (FVS) of states. It turns out that many asynchronous system models possess a FVS that is significantly smaller than their corresponding reachable state space. This common feature of asynchronous systems motivates us to speed up the Markovian analysis by focusing on the FVS.

Specifically, we create a new Markov chain which has the FVS as its state space. One of the key properties of this new Markov chain is that these feedback states keep the same relative occurrence frequencies as they have in the original Markov chain. This is the first step of our analysis and we call it *state compression*. Next, the stationary distribution of the new Markov chain is computed by the Power method or even by direct (non-iterative) methods if the new Markov chain is sufficiently small. Finally, the stationary distribution of the original Markov chain is expanded from that of the new Markov chain just computed. We call this third step *expansion*. Figure 1 depicts an overview of our method.

Ideally, we might like to find a minimum FVS in order to make the state space of the new Markov chain small. However, this is a well-known *NP*-complete problem [19]. Many heuristic algorithms to find a *minimal FVS* [20, 21, 22] exist, but they are all too expensive for large graphs. We give two simple heuristics to search for a good FVS which is not necessarily the minimum. Application to several real designs have demonstrated over an order of magnitude reduction of the overall CPU time. Moreover, we demonstrate that our method can analyze larger systems than the method in [6].

In Section 2, we review the basics of Markov chains and the Power method for obtaining stationary distributions. Section 3 gives detailed description of our state-compression-based stationary analysis. Section 4 describes our heuristic algorithms to find a good feedback set to form the state space of the new Markov chain. Implementation issues are noted in Section 5 and experimental results are given in Section 6. We conclude this paper and outline some possible future work in Section 7. Due to space constraints, the proofs of all the theorems are reported in [23].

2 Preliminaries

2.1 Markov chains

Consider a *finite state space* $S = \{1, 2, \cdots, N\}$. A random process X in *discrete time* on state space S is a sequence of random variables each taking values in S, i.e., $X = \{X_n \in S : n \geq 0\}$. The value assumed by random variable X_n is called the state of X at time instant n. Process X is a *Markov chain* (or is *Markovian*) if it satisfies the *Markov property*, i.e, its future evolution depends only on the current state. More formally, X is a Markov chain if for all $n \geq 1$ and all $j, i, x_{n-1}, \cdots, x_1, x_0 \in S$, $Pr(X_{n+1} = j \mid X_n = i, (X_{n-1} = x_{n-1}, \cdots, X_1 = x_1, X_0 = x_0)) = Pr(X_{n+1} = j \mid X_n = i)$, where $Pr(A \mid B)$ denotes the conditional probability that A occurs given that B occurs. A Markov chain X is *time-homogeneous* if $Pr(X_{n+1} = j \mid X_n = i)$ is independent of time n. In this case, we define a matrix P whose element at the i-th row and j-th column, $P_{ij} = Pr(X_{n+1} = j \mid X_n = i) = Pr(X_1 = j \mid X_0 = i)$. Matrix P is called the *(1-step) transition matrix* of X. Since Markov chains in our models are always time-homogeneous [6], assume X denotes a time-homogeneous Markov chain.

A *sample path* of X with length $n > 0$ is a sequence of states $(s_0, s_1, \cdots, s_{n-1})$ visited by X during a particular run. States s_0 and s_{n-1} are called the *head* and the *tail* state, respectively. By definition, we must have $Pr(X_{k+1} = s_{k+1} \mid X_k = s_k) > 0$ for all $0 \leq k < n-1$. A *cycle* is a sample path of length at least 2 in which the head and tail states are identical. If no other two states are identical, the cycle is *simple*.

A state is called *recurrent* if the chain starting from that state will return to it with probability 1. Otherwise, it is called *transient*. Hence, once visited, a recurrent state is expected to be re-visited infinitely often whereas a transient state is expected to be re-visited only a finite number of times. Two states *communicate* (or are *strongly connected*) if there exists a cycle containing both states. Similarly, a subset of states communicate if any two states of the subset communicate. A *recurrent class* is a maximal communicating subset that contains only recurrent states. If S contains transient states or it has multiple recurrent classes, X is called *reducible*. Otherwise, X is *irreducible*. Not that all states of an irreducible Markov chain X communicate. Finally, a state is called *periodic* with a period d if d is the largest integer such that once visited the state is re-visited only after a multiple of d time steps. If a state has period 1, it is said to *aperiodic*. States in the same recurrent class have the same period. X is called periodic (aperiodic) if all its states are periodic (aperiodic).

Let $M = (S, P)$ be an irreducible finite state Markov chain in discrete time where S is the state space and P is the transition matrix. If M is aperiodic, it tends to stabilize in the sense that the probability for it to be in any state $i \in S$ converges. Formally, $\lim_{n\to\infty} P\{M_n = i\}$ converges to a constant, π_i, the stationary probability of state i. If M is periodic, the average probability of M being in state i over the time, i.e., $\lim_{n\to\infty} \frac{1}{n}\sum_{k=0}^{n} P\{M_k = i\}$, converges to a constant. Let this constant be denoted by π_i as well. In both cases, M has the following property:

$$\pi = \pi P \tag{1}$$

where π is the row vector, $(\pi_1, \pi_2, \cdots, \pi_{|S|})$ satisfying $\sum_{i=1}^{|S|} \pi_i = 1$. π is called the *stationary distribution* because M_n has the same distribution for all $n \geq 1$ if M_0 has distribution π.

In the sequel, we assume the Markov chains are irreducible for the following reasons. The state classification technique developed in [24] can be used to efficiently identify transient states and all recurrent classes. If the original Markov chain has only one recurrent class (which is often the case [13, 24]), all transient states can be removed, reducing the problem to that of an irreducible Markov chain. Otherwise, the stationary analysis can be restricted to each of its recurrent classes. If necessary, one may compute the limiting probability for each of the recurrent classes to be hit from a given initial distribution by performing a transient analysis. Our implementation supports both these analyses.

2.2 The Power method

The Power method is one among many well known methods to compute an *eigenvector* of a square matrix A corresponding to its *dominant* (largest in magnitude) eigenvalue. In general, matrix A has N eigenvalues (not necessarily distinct) denoted by

$\lambda_1, \lambda_2, \cdots, \lambda_N$ if N is the number of its columns (or rows). Eigenvalue λ is *m-multiple* if A has $m > 0$ eigenvalues equal to λ. In the case where $m = 1$, λ is a distinct eigenvalue of A and is called *simple*.

Suppose one of the following two conditions holds:
Condition 1 A has a dominant eigenvalue which is simple, i.e., $|\lambda_1| > |\lambda_2| \geq \cdots \geq |\lambda_N|$,
Condition 2 : A has a dominant eigenvalue which is real and *m-multiple*, i.e., $\lambda_1 = \lambda_2 = \cdots = \lambda_m > |\lambda_{m+1}| \geq \cdots \geq |\lambda_N|$.
The Power method can then be used to compute an eigenvector corresponding to the dominant eigenvalue λ_1 according to the following iterative procedure:

$$\mathbf{x}^{(n+1)} = \mathbf{x}^{(n)} A, \ n = 0, 1, \cdots$$

where $\mathbf{x}^{(0)}$ is an arbitrary non-zero vector. The sequence $\{\frac{\mathbf{x}^{(n)}}{\lambda_1^n} : n = 0, 1, \cdots\}$ converges to a desired eigenvector denoted by $\mathbf{x}$. The convergence is *geometric* in the sense that the norm 1 distance between $\mathbf{x}^{(n)}$ and $\mathbf{x}$ is bounded from above by some geometrically decreasing quantity, i.e.,

$$||\frac{\mathbf{x}^{(n+1)}}{\lambda_1^n} - \mathbf{x}|| \leq \mathcal{C}e^{-\alpha n}$$

where $\mathcal{C}, \alpha > 0$ are constants. The convergence rate, $e^{-\alpha}$, is determined by $|\frac{\lambda_{m+1}}{\lambda_1}|$. The smaller this ratio, the faster the convergence. To avoid possible overflow (in case $|\lambda_1| > 1$) or underflow (in case $|\lambda_1| < 1$), the resulting vector may be normalized after several iterations. The procedure terminates when a user-defined convergence criterion is satisfied.

2.3 Computing π using the Power method

For an irreducible aperiodic Markov chain $M = (S, P)$, the transition matrix P has a simple dominant eigenvalue which is 1. That is, eigenvalues of P satisfies: $1 = \lambda_1 > |\lambda_2| \geq \cdots |\lambda_{|S|}|$ [25]. From (1), we see the stationary distribution of M is an eigenvector of P corresponding to the simple dominant eigenvalue 1. Thus, π can be iteratively computed using the Power method as follows:

$$\pi^{(n+1)} = \pi^{(n)} P, \ n = 0, 1, \cdots \quad (2)$$

where $\pi^{(0)}$ is an initial distribution. The convergence is geometric with rate determined by $|\frac{\lambda_2}{\lambda_1}| = |\lambda_2|$.

If there are no rounding errors, this iteration process needs no normalization in principle since $\lambda_1 = 1$. However, due to finite machine precision, normalization is still necessary after a large number of iterations, e.g., 1,000 iterations, to ensure that any unexpected drift will be scaled away.

If the irreducible Markov chain M has a period $d > 1$, the transition matrix P has not only eigenvalue 1 but also $d - 1$ other eigenvalues scattered on the unit circle. Thus, neither *Condition 1* nor *Condition 2* is satisfied. Therefore, the Power method cannot be directly used to compute π.

However, the eigenvalues of matrix P^d satisfy *Condition 2* where $1 = \lambda_1 = \lambda_2 = \cdots = \lambda_d > |\lambda_{d+1}| \geq \cdots \geq |\lambda_{|S|}|$. Thus, one may use the Power method to compute all d independent eigenvectors of P^d corresponding to eigenvalue 1. Moreover, it can be shown that the average of these (normalized) d eigenvectors equals to π. In fact, π can be computed iteratively as follows:

$$\overline{\pi}^{(n)} = \frac{1}{d} \sum_{r=0}^{d-1} \pi^{(nd+r)} \quad (3)$$

It simply takes the average of every d iteration results from the Power method defined by (2). The computed sequence $\{\overline{\pi}^{(n)} : n \geq 0\}$ converges to π geometrically with a rate determined by the ratio $|\frac{\lambda_{d+1}}{\lambda_1}| = |\lambda_{d+1}|$. A detailed proof can be found in [23].

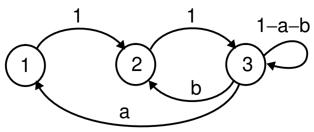
Note that by setting $d = 1$, the above iterative procedure reduces to the simple Power method used in the aperiodic case. The period of M is found using a symbolic version of a simple algorithm described in [26] which is very effective in practice.

3 State-compression-based analysis

For what follows, let $M = (S, P)$ be an irreducible Markov chain. We speed up its (stationary) analysis by focusing on part of the state space rather than the entire space. Precisely, we restrict M to a subset of states, S' ($S' \subseteq S$), obtaining a new Markov chain $M' = (S', P')$, a step referred to as *state compression*. Next, the stationary distribution π' of M' is computed using the Power method. Finally, the stationary distribution π of M is expanded from π', which we call the *expansion* step. The following subsections describe these three steps in order.

3.1 State compression

The first part of this section describes the construction of a new Markov chain M' with a (hopefully) smaller state space S'. The remainder of this section develops the criterion for the selection of S' by taking advantage of properties of typical asynchronous systems such that the overhead for the construction and the subsequent analysis of M' is nearly minimized.



(a) STG

$$P = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ a & b & 1-a-b \end{pmatrix}$$

(b) Transition matrix P

Figure 2: A 3-state Markov chain.

3.1.1 Constructing a new Markov chain

We start by constructing a new random process X from the original Markov chain M such that X has a (hopefully) smaller state space and some special property. Then, we show that X is a time-homogeneous irreducible Markov chain and that there is a simple relationship between the stationary distributions of X and M.

Suppose S has N states, i.e, $S = \{1, 2, \cdots, N\}$. Suppose further that S' has $N' \leq N$ states chosen from S. Without loss of generality, let us assume S' consists of the first N' states of S such that $S' = \{1, 2, \cdots, N'\}$. Let us denote by $\overline{S'}$ the remaining states in S. That is, $\overline{S'} = \{N'+1, N'+2, \cdots, N\}$. These two subsets, S' and $\overline{S'}$, form a partition of S. They imply a decomposition of the transition matrix P into four sub-matrices and a decomposition of the stationary distribution π into two sub-vectors:

$$P = \begin{pmatrix} P_{S'} & P_{S'\overline{S'}} \\ P_{\overline{S'}S'} & P_{\overline{S'}} \end{pmatrix} \quad \text{and} \quad \pi = (\pi_{S'}, \pi_{\overline{S'}}).$$

Sub-matrix $P_{S'}$ describes the 1-step transition probabilities between any two states in subset S'. Sub-matrix $P_{S'\overline{S'}}$ describes the 1-step transition probabilities from a state in S' to a state in $\overline{S'}$. The other two sub-matrices can be interpreted similarly. Finally, sub-vectors $\pi_{S'}$ and $\pi_{\overline{S'}}$ denote the stationary probabilities of the states in S' and in $\overline{S}$, respectively.

Example (Decomposition of P and π): The transition matrix of a Markov chain is usually depicted as a *labeled state transition graph* (STG) with vertices denoting the states and labels on the edges denoting the transition probabilities. Figure 2 shows the STG and the transition matrix of a 3-state Markov chain.

Assume S' contains the first two states of S, i.e., $S' = \{1, 2\}$, and of course, $\overline{S'} = \{3\}$. Then, we have following decomposition of P and π.

$$\begin{array}{lll} P_{S'} = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} & & P_{S'\overline{S'}} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \\ P_{\overline{S'}S'} = \begin{pmatrix} a & b \end{pmatrix} & & P_{\overline{S'}} = \begin{pmatrix} 1-a-b \end{pmatrix} \\ \pi_{S'} = (\pi_1, \pi_2) & & \pi_{\overline{S'}} = (\pi_3) \quad \square \end{array}$$

While the purpose of our construction for a new Markov chain M' is to reduce the state space that the Power method has ultimately to deal with, we hope it also guarantees a simple relationship between π' and $\pi_{S'}$ so that we may easily expand π' to obtain the entire vector π. Perhaps the simplest such relationship is that for every state $i \in S'$, π_i can be obtained from π'_i by a simple scaling. That is, $\pi_i = \mu \pi'_i$, where the scaling factor μ is the same for all $i \in S'$. Or in vector form, $\pi_{S'} = \mu \pi'$.

For example, suppose M has a state space $S = \{1, 2, 3\}$. If the compressed Markov chain contains the first two states of S, i.e., $S' = \{1, 2\}$. Then, we would hope there is a simple relation: $(\pi_1, \pi_2) = \mu\pi' = \mu(\pi'_1, \pi'_2)$ where μ is a constant. Indeed, this is true and we will later show that if we can get $\pi_{S'}$, then $\pi_{\overline{S'}}$, $\{\pi_3\}$ in this example, can be easily computed from π'.

To construct the compressed Markov chain with above mentioned simple scaling property, one possible way is to construct a new random process X that keeps track of all time instances when the original Markov chain M is in a state of S' and skips all other time instances. Note that the states of S' keep the same relative occurrence frequency in process X as they have in Markov chain M. This idea may be best described by a small example.

Example (Defining a new random process): A *sample path* of a random process $M = \{M_n : n \geq 0\}$ is a particular run of M over the time. Let M be the 3-state Markov chain described above. Again, let the new random process $X = \{X_n : n \geq 0\}$ have state space $S' = \{1, 2\}$. Suppose a sample path of M is:

$$\begin{array}{r|cccccccc c} M_n: & 1 & 2 & 3 & 3 & 2 & 3 & 1 & 2 & \cdots \\ \text{time } n: & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & \cdots \end{array}$$

For the new random process X to be created, its corresponding sample path must be:

$$\begin{array}{r|cccccccc c} X_n: & 1 & 2 & & & 2 & & 1 & 2 & \cdots \\ \text{time } n: & 0 & 1 & & & 2 & & 3 & 4 & \cdots \end{array}$$

Note that starting from 0, time for process X advances by one only when Markov chain M is in a state of S' at the next time instant. Or equivalently, time for process X stops when M is not in S'. $\square$

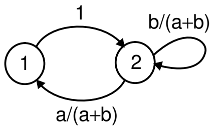
More precisely, let us define process X as follows:

$$X = \{X_n = M_{\tau(n)} : n \geq 0\} \tag{4}$$

where function τ is defined recursively:

$$\tau(0) = \min\{k \mid k \geq 0, M_k \in S'\},$$
$$\tau(i) = \min\{k \mid k > \tau(i-1), M_k \in S'\} \text{ for all } i > 0.$$

Theorem 3.1 *If M is a time-homogeneous irreducible Markov chain, then process X defined by (4) is also a time-homogeneous irreducible Markov chain.*



(a) STG

$$P' = \begin{pmatrix} 0 & 1 \\ \frac{a}{a+b} & \frac{b}{a+b} \end{pmatrix}$$

(b) transition matrix P'

Figure 3: The compressed Markov chain M'.

To be consistent, let us denote this new Markov chain by M' instead of X. As we know, M' must have a unique stationary distribution. Let this stationary distribution be denoted by π'.

Corollary 3.1 *$\pi_{S'} = \mu\pi'$, where $\mu > 0$ is a constant.*

From Corollary 3.1, we see that that expansion for $\pi_{S'}$ from π' can be done by a simple scaling as expected. Later in this section, we will show how to determine the constant μ.

To compute π', the transition matrix P' of the new Markov chain M' needs to be determined.

Theorem 3.2 *M' has a transition matrix*

$$P' = P_{S'} + P_{S'\overline{S'}}(I - P_{\overline{S'}})^{-1}P_{\overline{S'}S'}. \qquad (5)$$

Example (State compression): Continuing with the previous example, we know that (4) defines a new Markov chain $M' = (S', P')$ where $S' = \{1, 2\}$ and P' is determined by (5), i.e.,

$$\begin{aligned} P' &= \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} + \\ &\quad \begin{pmatrix} 0 \\ 1 \end{pmatrix} \left[\begin{pmatrix} 1 \end{pmatrix} - \begin{pmatrix} 1-a-b \end{pmatrix} \right]^{-1} \begin{pmatrix} a & b \end{pmatrix} \end{aligned}$$

Figure 3 shows the corresponding STG. One can use any direct or iterative method to obtain $\pi' = (\pi'_1, \pi'_2) = \frac{1}{2a+b}\begin{pmatrix} a & a+b \end{pmatrix}$. On the other hand, if one computes π directly from P, $\pi = (\pi_1, \pi_2, \pi_3)$ $= \frac{1}{1+2a+b}\begin{pmatrix} a & a+b & 1 \end{pmatrix}$. From this, we check $\pi_{S'} = \frac{1}{1+2a+b}\begin{pmatrix} a & a+b \end{pmatrix} = \frac{2a+b}{1+2a+b}\left[\frac{1}{2a+b}\begin{pmatrix} a & a+b \end{pmatrix}\right] = \mu\pi'$, where $\mu = \frac{2a+b}{1+2a+b}$. $\square$

3.1.2 Finding a proper S'

We have now established that given a time-homogeneous irreducible Markov chain $M = (S, P)$, we can create a new Markov chain $M' = (S', P')$ with a (hopefully) smaller state space $S' \subseteq S$ so that $\pi_{S'}$ can be expanded from π'.

However, when determining the transition matrix for M' using Theorem 3.2, a matrix inversion, i.e, $(I - P_{\overline{S'}})^{-1}$, has to be performed. This is not desired because the size of $\overline{S'}$ can be very large, close to the size of S if the new Markov chain has much fewer states than M so that this inversion may take an intolerably long time. The remainder of this subsection show how we may avoid this problem by choosing a proper state space S' for the new Markov chain.

It is shown [23] that $\lim_{k\to\infty} P^k_{\overline{P'}} = 0$ whenever S' is a proper subset of S. This implies that matrix $I - P_{\overline{S'}}$ is invertible and can be determined as:

$$(I - P_{\overline{S'}})^{-1} = I + P_{\overline{S'}} + P^2_{\overline{S'}} + \cdots$$

Let us call this quantity the *probabilistic reflective transitive closure* of $P_{\overline{S'}}$ and denote it by $P^*_{\overline{S'}}$.

If the term $P^k_{\overline{S'}} = \mathbf{0}$ when k is sufficiently large, e.g., $k \geq K$ for some fixed K, the inversion of $I - P_{\overline{S'}}$ may be replaced by finite number of matrix multiplications and summations, i.e.,

$$(I - P_{\overline{S'}})^{-1} = I + P_{\overline{S'}} + P^2_{\overline{S'}} + \cdots + P^{K-1}_{\overline{S'}} \qquad (6)$$

However, this is not generally true. In particular, if there is a cycle in $\overline{S'}$ (recall a cycle is a path with its tail and head states being identical), then there is a path of infinite length because a cycle can be traversed infinitely many times. This means that the term $P^k_{\overline{S'}}$ will never be zero for any fixed number k.

In fact, if S' is chosen in such away that there is no cycle in $\overline{S'}$, then, $(P_{\overline{S'}})^k = \mathbf{0}$ for all $k \geq K$ where K is the length of the longest path in $\overline{S'}$. Consequently, the inversion of $I - P_{\overline{S'}}$ can be determined by (6) without an error. This is a key observation of this paper.

Ensuring there is no cycle in $\overline{S'}$ is equivalent to finding a subset S' of S such that every cycle in S has at least one state in S'. The latter is the well known *feedback vertex set* (FVS) problem in graph terminology where a FVS is a subset of vertices of a graph and by removing all the edges connected with the vertices in this subset, the graph becomes acyclic.

Example (Choosing S'): In the previous example, we took $S' = \{1, 2\}$. It is not a FVS since the cycle $(3, 3)$ remains unbroken. If states in S' and those in $\overline{S'}$ switch, we get a FVS for $S' = \{3\}$ since every cycle must contain state 3. Meanwhile, $\overline{S'} = \{1, 2\}$ and $P_{\overline{S'}} = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}$.

Since the longest path in $\overline{S'}$, i.e., $(1, 2)$, has a length of 2, we can compute $(I - P_{\overline{S'}})^{-1}$ as follows:

$$(I - P_{\overline{S'}})^{-1} = I + P_{\overline{S'}} = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \square$$

One might think it advantageous to make S' as small as possible. There are several reasons, however, which suggest that we should not search for the minimum FVS to be S'. First, finding the minimum FVS is NP-complete [19]. Secondly, as we will further explain, there is a trade-off associated with the total analysis time between the size of S' and $\overline{S'}$. In Section 4, we propose simple heuristics to find a good FVS for S'.

3.2 Computing π'

One we obtain the compressed Markov chain M', its stationary distribution π' is computed iteratively using the Power method described in Section 2.3.

It is generally hard to give a good initial guess. The simplest way to build an initial guess is to concentrate all probability to a particular state and set all other states to 0 probability. Alternatively, one may distribute the probability evenly to all the states. Our experiments show that there exist cases where either of them can be better than the other. In our implementation, we set the initial guess to be equiprobable by default. In this case, $\pi'^{(0)}$ is set to be an all-one vector, i.e., $\pi'^{(0)} = (1, 1, \cdots, 1)$ rather than $\{\frac{1}{|S'|}, \frac{1}{|S'|}, \cdots, \frac{1}{|S'|}\}$ where $|S'|$ denotes the size of S' and do normalization when necessary.

We know from Section 2.3 that for a Markov chain (S, P) with period d it is the magnitude of $d+1$-th eigenvalue of matrix P^d that determines the convergence rate. To see how state compression might change the magnitude of this eigenvalue, let us assume the Markov chain is aperiodic, i.e., $d = 1$.

It is possible that state compression can increase the magnitude of the second largest eigenvalue, but our experiments on several real examples suggest this rarely happens. One possible explanation to this may be stated as follows. Any given Markov chain $M = (S, P)$ may be considered as a random sample from the family of all possible Markov chains with the same state space as that of M. Consequently, one may think of the eigenvalues of P as uniformly distributed in the interval $[0, 1)$ except the largest eigenvalue 1. We may further assume that the eigenvalues of P', the transition matrix after state compression, are also uniformly distributed in the interval $[0, 1)$. Then, we deduce the probability that $|\lambda'_2| > |\lambda_2|$ to be very small when $|S'|$ is significantly smaller than $|S|$. A similar argument holds for the case where the Markov chain is periodic.

3.3 Expanding π' to π

The last step in the analysis is to expand the stationary distribution π' of M' to π of M. We know from Corollary 3.1 how π' relates to $\pi_{S'}$. The following theorem further relates π' to $\pi_{\overline{S'}}$.

Theorem 3.3 *If π and π' are the stationary distributions of M and M', respectively, then $\pi_{\overline{S'}} = \mu\pi' P_{S'\overline{S'}} P^*_{\overline{S'}}$, where μ is the normalization factor which is the same as in Corollary 3.1 and is equal to:*

$$\mu = \frac{1}{1 + \pi' P_{S'\overline{S'}} P^*_{\overline{S'}} \mathbf{1}^T} \tag{7}$$

and $\mathbf{1}^T$ is a properly sized all-one column vector.

Example (Expansion): Let us continue with the previous example where $S' = \{3\}$ for M'. Since S' has only one state, following (5), we have $M' = (1)$ and thus $\pi' = (1)$.

By the result in the previous example computed for $P^*_{\overline{S'}}$, we may expand π' to π as below.

$$\begin{cases} \pi_{S'} = (\pi_3) = \mu\,(1) \quad \text{(by Corollary 3.1)} \\ \pi_{\overline{S'}} \;=\; (\pi_1, \pi_2) = \mu\,(1) \begin{pmatrix} a & b \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \\ \qquad\;\; = \; \mu \begin{pmatrix} a & a+b \end{pmatrix} \text{(by Theorem 3.3).} \end{cases}$$

Following (7), we compute $\mu = (1 + 2a + b)^{-1}$. □

In fact, the value for μ is typically not important. What is important is the *relative values* computed for all the states in S by the expansion step. These relative values represent the *relative frequencies* of the corresponding states to be visited by the original Markov chain M in the long run which is generally sufficient for further analysis of performance metrics (as well as power estimation [13]).

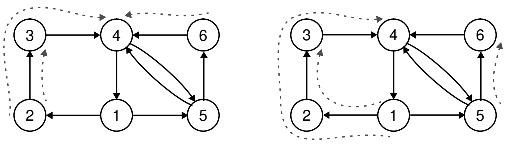
4 State compression heuristics

For the description of our heuristic algorithms, we need the following definitions. Let us assume an irreducible Markov chain M has a state space S and a transition matrix P.

Definition Given two states $i, j \in S$ for which $P_{ij} > 0$, we say state j is a *next state* of state i and state i is a *previous state* of state j.

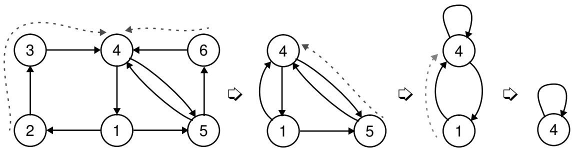
Definition A state is called a *decision state* if it has multiple next states. Otherwise, it is a *non-decision state*. A state is called a *merging state* if it has multiple previous states. Otherwise, it is called a *non-merging state*.

Note that non-decision state can reach other states only through its sole next state. Similarly, a non-merging state can be reached only through its sole previous state.



(a) Forward-strings (b) Reverse-strings

Figure 4: Examples of strings.



compress (2,3,4) and (6,4) compress (5,4) compress (1,4)

Figure 5: Forward-string-based state compression.

Definition A *forward-string* is a sample path which contains only non-decision states except possibly for its tail state. A *reverse-string* is a sample path which contains only non-merging states except possibly for its head state. A *maximal forward-string* is one which is not contained in any other forward-string. A *maximal reverse-string* is defined similarly.

Note that a forward-string is maximal *iff* it starts with a next state of some decision state and ends with a decision state. Similarly, a reverse-string is maximal *iff* it starts with a merging state and ends with a previous state of a merging state.

Example (Forward- and reverse-strings): Consider the STG shown in Figure 4. By definition, every single state is both a forward-string and a reverse-string. Example forward-strings of length more than 1 are $(2,3)$, $(2,3,4)$ and $(6,4)$. The latter two are also maximal. Example reverse-strings are $(1,2,3)$, $(1,2,3,4)$, and $(5,6)$. The latter two are also maximal.

Finally, we define a *loop* to be a simple cycle that is both a forward-string and a reverse-string.

Our first heuristic algorithm for state compression is based the forward-string concept whereas the second heuristic algorithm combines both forward- and reverse-string concepts.

4.1 Forward-string-based compression

Lemma 4.1 *An irreducible Markov chain (S, P) has a maximal forward-string iff it has decision states. If so, such a string has a length no more than $|S|$.*

Lemma 4.2 *Suppose σ is a forward-string containing a state s. For any FVS S_1 such that $s \in S_1$, $S_2 = (S_1 - \{s\}) \cup \{tail\ of\ \sigma\}$ is also a FVS.*

Algorithm 4.1 State_Compression_1
input: *A Markov chain $M = (S, P)$.*
output: *A compressed Markov chain $M' = (S', P')$.*
begin
  $M' :=$ **String_Compress**$(S, P, FORWARD)$;
end

procedure **String_Compress**(S, P, D)
begin
  if $D = FORWARD$ **then**
    Let $S' :=$ the set of decision states in S;
  else
    Let $S' :=$ the set of merging states in S;
  if $S' = \emptyset$ **then**
    Take s as an arbitrary state of S;
    $S' := \{s\}$; $P' := (1)$;
    return (S', P');
  endif
  if $S' = S$ **then**
    return (S, P);
  compute P' from P and S' by (5) and (6);
  String_Compress(S', P', D);
end

Figure 6: Compression based on forward-strings.

The above lemmas state that if S has maximal forward-strings, their tails form a FVS of S. In fact, we have following theorem.

Theorem 4.1 *If S has decision states, there is a FVS containing only decision states. Otherwise, S forms a loop, and hence any single state is a FVS.*

Our first heuristic for state compression is thus to search for all the decision states of S. If there is no decision state, any single state of S forms a FVS which must be the smallest. Otherwise, we select all the decision states as a FVS. The heuristics can be applied iteratively. That is, each time the heuristic is applied, a new Markov chain is created. Since a decision state in the original Markov chain may become a non-decision state in the new Markov chain, the heuristic may be applied again to get an even smaller Markov chain. This process continues until the heuristic fails to compress the Markov chain any further. Figure 5 illustrates this idea.

The corresponding algorithm is shown in Figure 6. The iterative procedure **String_Compress** with the *direction* parameter set to key word FORWARD performs the iterative compression. It returns immediately if all states currently left are decision states. If not, it checks whether they are all non-decision states (forming a loop) in which case it takes an arbitrary state as a FVS and returns a Markov chain containing only this state. Otherwise, it takes all the decision states as a FVS, creates a new Markov chain with the FVS as its state space and contructs its transition matrix. Then the procedure calls itself to try compression on the new Markov chain.

4.2 Further compression based on reverse-string

It is not difficult to see that the compressed Markov chain returned from Algorithm 4.1 does not contain any forward-string of length more than 1. In other words, our forward-string-based heuristics compresses forward-strings down to their tail states. This same idea can be applied on the reverse-strings. That is, we may compress all reverse-strings up to their head states (by invoking procedure **String_Compress** with the direction parameter set to REVERSE). These two compression ideas can be combined.

To avoid possible conflict of the two compression strategies on a forward-string that is also a reverse-string, we do not apply them concurrently but rather *sequentially*. The new algorithm starts by performing forward-string compression followed by reverse-string compression, followed by another forward-string compression and so forth. This procedure terminates when no more strings are compressed during most recent two compressions.

The compressed Markov chain returned by the new algorithm will be no larger than that returned by Algorithm 4.1. One fact worth noticing, however, is during each of its iteration, states in $\overline{S'}$, i.e., those not in the compressed Markov chain M', are all non-decision states with respect to the current Markov chain. This means that the sub-matrix, $P_{\overline{S'}}$, contains only 0's and 1's and hence is a Boolean matrix. This will speed up the computation of $P_{\overline{S'}}$ when implemented using symbolic techniques (as will be explained in the following section). Consequently, in some cases Algorithm 4.1 can result in less overall CPU time even though it involves analyzing a larger Markov chain.

5 Implementation

We embed the method discussed in the previous sections in a modified version of the symbolic performance analysis tool described in [6]. It serves as an alternative for the stationary Power method previously adopted by the tool. In particular, we make several notes as follows.

To reduce the time spent in the state compression step, we compute only the Boolean transition matrix instead of the probabilistic transition matrix for the newly created Markov chain during each iteration. It is sufficient in the sense that it does not affect the final FVS to be found. Only after obtaining the final FVS S', i.e., the smallest FVS that string-based compression can get, the probabilistic transition matrix P' of the final compressed Markov chain M' is computed according to (5) and (6). In addition, when computing P', we do not compute the reflective probabilistic transitive closure $P'^{*}_{\overline{S'}}$ independently. Instead, we first left multiply $P_{\overline{S'}}$ by the quantity $P_{S'\overline{S'}}$ and then iteratively compute for $P'^{*}_{\overline{S'}}$. This is because the FVS S' is typically much smaller than the remain state space $\overline{S'}$ so that the above treatment saves significant computation time.

Secondly, the convergence is checked after each iteration for both the standard Power method and our state-compression-based method. If the Markov chain under consideration is aperiodic, then the iterative procedure terminates after iteration n if $||\pi^{(n)} - \pi^{(n-1)}|| \leq \epsilon$, where $\pi^{(i)}$ is the probability vector after the i-th iteration and ϵ is a user defined constant. That is, we check the norm 1 distance between the probability vectors resulting from two consecutive iterations. It can be shown that this distance decreases not only *monotonically* but also *geometrically*. In case the Markov chain is periodic periodic with period d, criterion similar to that in the aperiodic case is:

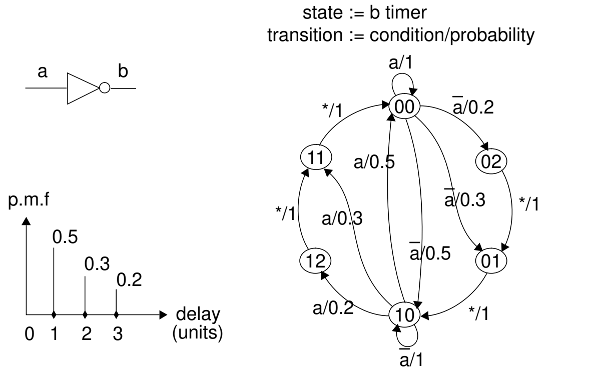
$$||\overline{\pi}^{(n)} - \overline{\pi}^{(n-1)}|| \leq \epsilon,$$

where $\overline{\pi}^{(n)}$ is defined by (3).

There are other convergence criteria recommended in the literature. For instance, one might take the relative supremum norm [27] as an alternative,

$$\sup_{i \in S} \left| \frac{\pi_i^{(n)} - \pi_i^{(n-1)}}{\pi_i^{(n)}} \right| \leq \epsilon.$$

This criterion is generally stronger than the one based on the norm 1 distance that we adopted. Nevertheless, choosing the convergence criterion is rather arbitrary and should depend on the particular applications one has [28]. In fact, it can be shown that for any finite state Markov chain, the above relative supremum norm and the norm 1 measures are both geometric and have the same convergence rate [28]. The difference is a constant factor. This is true for all other usually referenced norms as well. In our applications which target the average metrics of performance or power (i.e., emphasizing the stationary probabilities of subsets of the state space rather than those of individual states), we observe that the criterion based on the norm 1 measure is generally sufficient.



(a) an inverter and its delay distribution

(b) its component STG.

Figure 7: An example of component STGs.

6 Experiments

We evaluated our method on a number of Markov chain models of asynchronous systems some of which were previously analyzed for performance [6]. These models were first used for formal verification purposes and converted to those applicable to probabilistic performance analysis by annotating non-deterministic choices with probability distributions. Unlike some verification models that use continuous-time semantics for interval delays, our models use discrete-time semantics in which time is quantized. Developing probabilistic interval delay models without time discretization is a topic of current research.

In the discrete-time models, each circuit component is modeled by a State Transition Graph (STG) annotated with probabilities. The state bits in the STG may consist of internal state variables, output signals and timers. For example, the STG of an inverter with a probabilistic interval delay is shown in Figure 7. The system model is then obtained by synchronous composition of all component STGs in which the probability of a system state transition is the product of all corresponding transition probabilities in the component STGs. The specification format of the individual component STGs and the proof that the system STG is stochastic is outside the scope of this paper.

6.1 The FIFO

We first consider a FIFO with 6 identical stages and trivial environments as shown in Figure 8. The C-element has unit delay. The delay line in each stage has an interval delay distributed as $(1'0.1, 2'0.1, 3'0.8)$, i.e., with probability 0.1, 0.1 and 0.8, it has a delay of 1 , 2 and 3 units, respectively. The environment

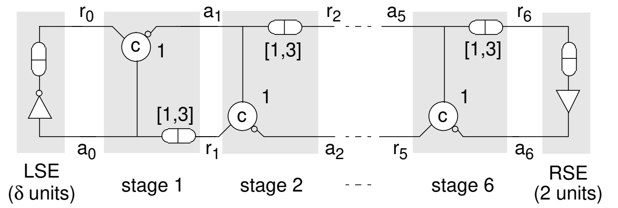


Figure 8: A 6-stage FIFO with simple environments.

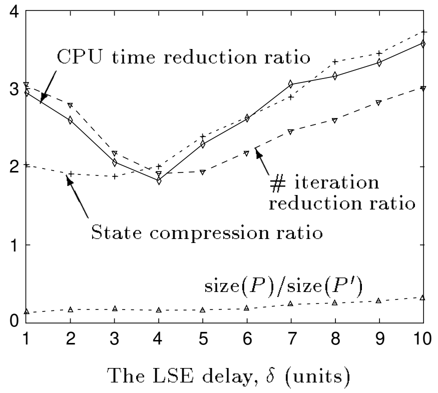


Figure 9: Speedup in FIFO analysis.

at the right side (RSE) acknowledges requests with a fixed delay of 2 units upon requested. The environment at left side (LSE hereafter) requests with a fixed delay of $\delta \geq 1$ units upon acknowledged.

Our experiment varies the delay of the LSE from 1 unit to 10 units. If we think of an empty stage (free of data) as a bubble, the average number of non-empty FIFO stages at any time instance decreases with an increase in the LSE delay. This suggests that the FIFO behaves more "sequentially" as the LSE gets slower, implying a sparser state space with a smaller feedback set. In the extreme case (not shown), when the LSE is very slow, only one stage contains data. In this case, the shortest cycle in the state space corresponds to a datum traveling through the entire FIFO in which each FIFO stage has its shortest delay. Indeed, in Figure 9 we see a trend of increasing state compression ratio as the delay of the LSE increases.

We note, however, that the overall state compression ratio in this example is relatively small. This may be because the circuit structure has a high density of cycles which seems to correlate with the presence of numerous cycles in the state space.

Figure 9 also plots the speedup in stationary analysis using our state-compression-based method over the standard Power method. The CPU time reduction ratio closely tracks with the state compression ratio when the LSE has a relatively large delay (over 4 units). When the LSE is relatively fast (with a delay less than 4 units), the CPU time reduction ratio tracks more closely with the number of iterations needed for convergence. Another interesting fact is the ratio between the sizes (in number of ADD [29] nodes) of the transition matrices P and P'. Clearly, the ADD for P' is significantly more complex than the ADD for P. However, the larger size of P' does not prevent the state-compression-based from achieving a significant reduction in overall run time. On the whole, we observe a speedup of close to or over a factor of 2. The model with the LSE set to 4 units requires the longest CPU time to solve (slightly over 15 minutes on a Sparc 20 with 256M of memory). In all cases, the time for state compression and expansion are negligible.

6.2 The differential equation solver

The second example which has been analyzed for performance metrics in [6] is the behavioral model of the differential equation solver [30]. As in [6], we consider its estimated and back-annotated versions. The latter is more accurate in that it accounts for some wire delays.

Table 1 gives the sizes of their reachable state space and the set of recurrent states. The back-annotated version achieves over three times the state compression ratio of the estimated version which is much higher than that of the FIFOs. This is largely because the modules in the DIFFEQs are *loosely coupled* and rather sequential. Due to the state compression, the number of power iterations to convergence is dramatically reduced. The curves in Figure 10 illustrate convergence of the distance of the probability vectors from two consecutive iterations. They clearly suggest that the Markov chains derived from both estimated and back-annotated versions possess a much higher convergence rate after state compression.

Table 2 lists the CPU times required. The state-compression-based method achieves over one order of magnitude in the overall run time reduction. Compared with the FIFOs, the DIFFEQ models require a larger portion of the run time for state compression mainly because their structures allow for more state compression than in the FIFOs. We conjecture that the higher state compression ratio is attributed to the fact that the circuit structure of the DIFFEQ solver has relatively fewer and larger cycles than the FIFOs.

6.3 A pausible clocking interface

Figure 11 shows two asynchronous modules (the senders) communicating with a synchronous module (the receiver) through a variation of a *pausible clocking interface* (PCI) [31] which is built inside the receiver and guarantees failure-free synchronization.

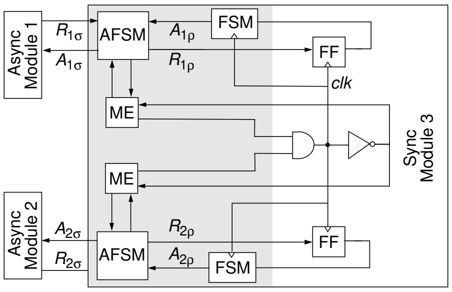
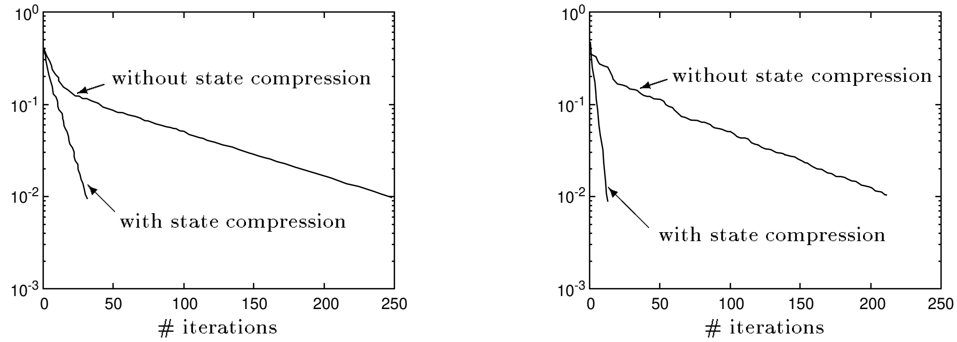


Figure 11: A pausible clocking interface.

When there is no request from either of the senders, the receiver is freely clocked by the ring oscillator (formed by an inverter, two MEs and an AND gate). It continues to run freely after receiving requests from the senders if there is a clear difference between the arrival time of the requests and that of the clock edge. However, if this time difference is significantly small so that synchronization failure may occur, either the clock is *paused* by stretching its falling edge or the corresponding AFSM delays its request out (a transition of signal $R_{x\rho}$). This is done so by the corresponding ME element. The AND gate ensures that the next clock edge is delayed until all possible synchronization failures have been resolved.

We make following timing assumptions. The AND gate and the inverter have delays of 4 units and 3 units, respectively. The two senders are identical. They have a geometrically distributed delay with a parameter of 0.9. That is, the probability of a request occuring $n \geq 1$ time units after acknowledgment is $0.1^{n-1} * 0.9$. Similarly, the delay at the receiver side is also geometrically distributed with a parameter of 0.9. The mutual exclusion element *mutex* has a unit delay and is assumed to be fair with simultaneously arriving requests. Table 1 lists the sizes of the reachable state space, the recurrent state set and the state space after compression. The model achieves a state compression ratio close to 6. A slightly closer study of the PCI behavior suggests that any cycle in its state space represents at least one complete clock cycle as generated by the ring oscillator, a possible explanation for the PCI to have a high state compression ratio.

We compare the overall CPU times required to compute the stationary distribution by the standard Power method and our method. The standard Power method fails to reach the convergence after 12 hours of CPU time whereas our method using state-compression converges within 3 minutes yielding a speedup of over two orders of magnitude. In par-



(a) The estimated version (b) The backannotated version.

Figure 10: Convergence of $||x^{(n)} - x^{(n-1)}||$ in the DIFFEQ model.

examples	# states				# iterations			size of transition matrix in number of ADD nodes		
	reachable	recurrent	after compress	compress ratio	w/o compress	w/ compress	reduction ratio	w/o compress	w/ compress	ratio
DiffeqEst.	7,632	4,870	614	7.9	247	39	7.5	3,384	3,258	1.04
DiffeqBka.	11,036	4,860	194	25.1	210	14	15	4,456	3,323	1.34
PCI2to1	51,376	51,376	8,704	5.9	> 4,287	35	>122	8,915	10,040	0.89

Table 1: State compression and iteration number reduction in the DIFFEQ and PCI analyses.

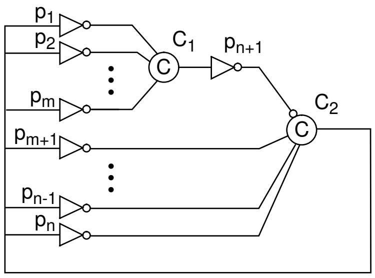


Figure 12: Synchronized processes.

ticular, we see that in our state-compression-based method, time spent in expansion is again negligible, but the time spent in state compression takes over half of the total CPU time in this case.

6.4 Synchronized processes

Our last example is a set of synchronized concurrently running processes. Figure 12 shows a simple configuration of $n+1$ ($n > 1$) processes from $\mathtt{p}_1$ to $\mathtt{p}_{n+1}$ each depicted by an inverter. All processes except $\mathtt{p}_{n+1}$ start running simultaneously. Process $\mathtt{p}_{n+1}$ initiates only after processes $\mathtt{p}_1$ through $\mathtt{p}_m$ ($0 < m \leq n$) complete and are synchronized at the C-element $\mathtt{C}_1$. The C-element $\mathtt{C}_2$ waits until processes $\mathtt{p}_{m+1}$ through $\mathtt{p}_{n+1}$ terminate, and then triggers processes $\mathtt{p}_1$ through $\mathtt{p}_n$, making the entire procedure repeat.

Let us assume all processes take some independent identically distributed random processing times as follows: with probability 0.1, a process takes a delay of 1 unit, with probability 0.4, a delay of 2 units, with probability 0.4, a delay of 3 units and with probability 0.1, a delay of 4 units. Further, both C-elements take a fixed delay of 1 unit.

It might be useful to know performance metrics such as the average cycle time of the entire procedure (average time separation of any two consecutive output transitions of the element $\mathtt{C}_2$), or the average waiting (idle) time for each of the processes. One might then perform a Markovian analysis in order to obtain these measures [6].

The state space of this simple configuration, unfortunately, increases roughly by a factor of 4 with each added process. Taking $n = 24$ processes, we face a state space of over 10^{15} states, excluding the possibility to use any explicit approaches on currently available machines. However, due to the symmetry involved in the configuration, we can successfully employ the symbolic technique [6] to obtain the stationary probability of each of the state using standard Power method with memory usage less than 30 Mbytes. Unfortunately, it takes over 14 hours to complete on a Sparc20 assuming the Power method

Examples	CPU time without compression (sec)	CPU time with state compression (sec)				Speedup
	Power iteration	Compression	Power iteration	Expansion	Total	
DiffeqEst.	762	11.5	48.3	3.0	62.8	12.9
DiffeqBka.	702	24.5	20.9	4.7	50.1	14.0
PCI2to1	incomplete @ 12h	84.4	50.0	2.6	137	> 315

Table 2: Speedup in the DIFFEQ and PCI analyses using state compression.

starts with the initial distribution where each state is equiprobable.

By the technique proposed in this paper, one may compress the state space down to a single state for every possible configuration in Figure 12 (i.e., for every possible value of m). In fact, that single state is always one of the two global synchronization states where all processes are idle and element $\mathtt{C}_2$ is about to make a transition. It is worth noticing from this simple example that high concurrency does not usually limit the state compression ratio, nor does generally the synchronization (e.g., at element $\mathtt{C}_1$). Rather, the main reason for us to achieve high state compression ratio is that there is only one feedback signal (i.e., the output of element $\mathtt{C}_2$), which implies the system has a small feedback set in its state space.

The Power method on the compressed chain converges immediately. It turns out that the compression step takes most of the run times. For n = 24, it takes slightly over 20 minutes at most when $m = 1$, and less than 10 minutes at least when $m = 24$. The time for expansion is negligible. Thus, one achieves a reduction of over a factor of 40 in CPU time. Figure 13(a) shows that the overall run time is a small polynomial function of the system size while the number of states grows exponentially.

Figure 13 plots the average cycle time of the entire procedure computed as a function of m. Further, in this example the sum of the average waiting time and the busy time of each process equals the average cycle of the system. Since the average busy times are the same for all processes (i.e., 2.5 units), their average waiting times are also the same which is 2.5 units less than the average cycle time for a fixed m.

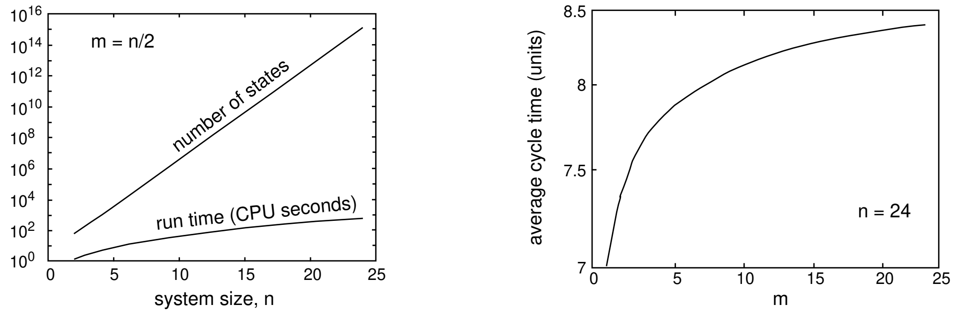
7 Conclusion and future work

Markov chain models currently formalized from specifications of asynchronous systems require intensive CPU time to find their stationary distributions. We propose to speedup this analysis by *mapping* the original Markov chain to a new Markov chain that has a smaller state space by compressing its state space followed by a post-processing step called expansion. We further propose the *FVS* to be the state space of the new Markov chain to reduce the computation associated with state compression. Simple *string-based* heuristics have been given to find a proper FVS. Experiments show that many asynchronous systems possess a FVS which is much smaller than their reachable state spaces. As a consequence, our method dramatically reduces the CPU time required, which prevents the Markovian analysis from being the bottleneck of performance and power analysis in many cases.

Studying more powerful mappings might be interesting future work. For instance, new state compression heuristics may be helpful to further accelerate the analysis by obtaining higher state compression ratio in system models such as FIFOs. In addition, it might be possible to construct a small Markov chain that is sufficient for targeted analysis without performing complete reachability analysis, the likely bottleneck in the future. Finally, due to the state explosion problem, any approach that constructs and then analyzes the *flat* (entire) underlying Markov chain (including our previous work in [6] which exploits symbolic techniques) is unlikely to have solved all the problems. However, the technique presented in this paper should be applicable to future hierarchical approaches.

Acknowledgment

We wish to acknowledge Prof. P. Baxendale at USC for his support on stochastic analysis and Prof. L. Holden at Norwegian Computing Center for discussions on convergence of Markov chains. We would also like to thank anonymous reviewers for their valuable comments. Special thanks go to Prof. M. Greenstreet at UBC for his insightful discussions on state compression and several corrections regarding the proofs.



(a) State space size v.s. overall run time

(b) System performance.

Figure 13: Experimental results of synchronized processes.

References

[1] H. Hulgaard and S. M. Burns. An algorithm for exact bounds on time separation of events in concurrent systems. *IEEE Trans. Computers*, 44(11), Nov. 1995.

[2] H. Hulgaard, T. Amon, S. M. Burns, and G. Borriello. Timing analysis of timed graphs with bounded delays using algebraic techniques. In *Proc. IEEE Conf. Decision and Control*, 1994.

[3] M. R. Greenstreet and K. Steiglitz. Bubbles can make self-timed pipelines fast. *J. VLSI Signal Processing*, 2(3):139–148, Nov. 1990.

[4] J. Ebergen and R. Berks. Response time properties of some asynchronous circuits. In *Proc. Int. Symp. Advanced Research in Asynchronous Circuits and Systems*, 1997.

[5] P. Kudva, G. Gopalakrishnan, E. Brunvand, and V. Akella. Performance analysis and optimization of asynchronous circuits. In *Proc. Int. Conf. Computer Design*, October 1994.

[6] A. Xie and P. A. Beerel. Symbolic techniques for performance analysis of asynchronous systems based on average time separations of events. In *Proc. Int. Symp. Advanced Research in Asynchronous Circuits and Systems*, 1997.

[7] S. Burns. *Performance Analysis and Optimization of Asynchronous Circuits.* PhD thesis, CIT, 1991.

[8] T. Lee. *A General Approach to Performance Analysis and Optimization of Asynchronous Circuits.* PhD thesis, CIT, 1995.

[9] J. Teich, L. Ghiele, S. Sriram, and M. Martin. Performance analysis and optimization of mixed asynchronous synchronous systems. *IEEE Trans. Computer-Aided Design*, May 1997.

[10] M. Holiday and M. Vernon. A generalized timed Petri net model for performance analysis. *IEEE Trans. Software Eng.*, 13:1297–1310, Dec. 1987.

[11] K. L. McMillan. *Symbolic Model Checking.* Kluwer Academic Publishers, 1993.

[12] G. Rubino and B. Sericola. Sojourn times in finite Markov process. *J. Appl. Prob.*, 27:744–756, 1989.

[13] G. D. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Markovian analysis of large finite state machines. *IEEE Trans. Computer-Aided Design*, Dec. 1996.

[14] V. D. Ploeg. Preconditioning techniques for large sparse, non-symmetric matrices with arbitrary sparsity patterns. In *Proc. IMACS Symp. Iterative Methods in Linear Algebra*, 1991.

[15] G. W. Stewart. Methods of simultaneous iteration for computing invariant subspaces of non-ruminating matrices. *Numerical Mathematics*, 25:123–136, 1976.

[16] A. M. Abdel-Moneim and F. W. Leysieffer. Weak lumpability in finite Markov chains. *J. Appl. Prob.*, 19:685–691, 1982.

[17] G. Rubino and B. Sericola. On weak lumpability in Markov chains. *J. Appl. Prob.*, 26:446–457, 1989.

[18] Y. Ho and X. Cao. *Perturbation analysis of discrete event dynamic systems.* Kluwer Academic Publishers, 1991.

[19] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, 1979.

[20] Jr. G. W. Smith and R. B. Walford. The identification of a minimal feedback vertex set of a directed graph. *IEEE Trans. Circuits and Systems*, 22(1):9–15, Jan. 1975.

[21] K. Cheng and V. D. Agrawal. A partial scan method for sequential circuits and feedback. *IEEE Trans. Computers*, 39(4):544–548, Apr 1990.

[22] S. Park and S. B. Akers. A graph theoretic approach to partial scan design by k-cycle elimination. In *Proc. IEEE Int. Test Conf.*, 1992.

[23] A. Xie and P. A. Beerel. *Accelerating Markovian analysis of synchronous systems using string-based state compression.* Tech. Rep., CENG-97-21, Univ. Southern Calif., 1997.

[24] A. Xie and P. A. Beerel. *Efficient state classification of finite state Markov chains.* Tech. Rep., CENG-97-20, Univ. Southern Calif., 1997.

[25] J. G. Kemeny and J. L. Snell. *Finite Markov Chains.* Springer, 1976.

[26] V. G. Kulkarni. *Modeling and Analysis of Stochastic Systems.* Chapman & Hall, 1995.

[27] W. J. Stewart. *An Introduction to the Numerical Solution of Markov chains.* Princeton Univ. Press, 1994.

[28] L. Holden. *Research Director*, Norwegian Computing Center at Univ. of Oslo. *private communication*, Dec. 1997.

[29] R. I. Bahar, E. A. Frohm, C. M. Gaona, and G. D. Hachtel. Algebraic decision diagrams and their applications. In *Proc. Int. Conf. Computer-Aided Design*, 1993.

[30] K. Y. Yun, P. A. Beerel, V. Vakilotojar, A. E. Dooply, and J. Arceo. A low-control-overhead asynchronous differential equation solver. In *Proc. Int. Symp. Advanced Research in Asynchronous Circuits and Systems*, 1997.

[31] K. Y. Yun and R. P. Donohue. Pausible clocking: A first step toward heterogeneous systems. In *Proc. Int. Conf. Computer Design*, Oct. 1996.