Accelerating Markovian Analysis of Asynchronous Systems Using String-based State Compression

Aiguo Xie and Peter A. Beerel

CENG 97-21

Department of Electrical Engineering - Systems
University of Southern California
Los Angeles, California 90089-2562
(213) 740-4481

November 1997

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 behavior 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 recovered via a process called *expansion*. The method is especially powerful when the Markov chain is *nearly deterministic*, which happens often in asynchronous systems. Our experimental results show that the method can yield reductions of more than an order of magnitude in CPU 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. Such tools can help guide design choices at high levels of abstraction, thereby shortening time-to-market, as well as validate design characteristics at lower levels of abstraction, thereby providing higher confidence in the design. 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 can improve the state-of-the-art in both areas, however, to simplify the exposition we focus on performance analysis.

There are currently two important 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 and Burns *et al* use a delay-annotated Petri-net model to obtain upper/lower bounds for the time-separations between arbitrarily indexed occurrences of two signal 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 results (average metrics) [1, 3, 7, 8, 9] for some classes of systems (e.g., those modeled by marked graphs or extended marked graphs). Average performance metrics may also be derived directly for some systems [3] (e.g., with FIFO-like structures) if all delays are exponentailly distributed so that the classic queuing theory may be applied.

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. In addition, probabilistic techniques can often handle a much wider class of systems and can be used to estimate higher moments (e.g., variance) of performance metrics.

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 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 evolution). Here, states represent the valuations of system signals plus delay signals (corresponding to transition firing dura-

tion 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 so called *sojourn time* [12], the average duration the system remains in 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 systems with hundreds of signals, it can still take hours to complete. Therefore, techniques that can significantly speed up Markovian analysis is of great demand.

There are many proposed techniques to speed up the iterative approaches such as 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 intensive ongoing research trend in probability theory on *lumpability* of Markov chains [16, 17]. The basic idea to lump 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 keeps the Markovian 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 the incorporation of the lumpability theory currently available [18].

The method we propose here takes advantage of a generalized notion of the *determinism* inherent in 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 set* of states. It turns out that many asynchronous system models possess a feedback set that is significantly smaller than their corresponding reachable state space. This distinct feature of asynchronous systems motivates us to speed up the Markovian analysis by focusing on the feedback set.

Specifically, we create a new Markov chain which has the feedback set 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.

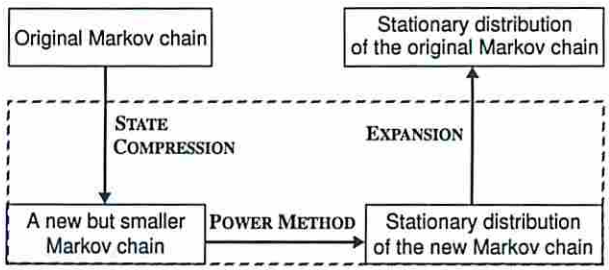


Figure 1: Block diagram of our method.

Ideally, we would like to find a minimum feedback set in order to make the state space of the new Markov chain small. However, this is a *NP*-complete problem known as the *minimum feedback vertex set (MFVS)* 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 quickly search for a good feedback set 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 system than possible using the method in [6].

In Section 2, we briefly review the basics of Markov chains and the Power method for obtaining stationary distributions. Section 3 gives detailed description of our method as well as the theorems we have proved to check the correctness of our method. Section 4 de-

scribes 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. The proofs of all theorems are put in the appendix.

2 Preliminaries

2.1 Markov chains

Consider a *finite state space* $S = \{1, 2, \cdots, N\}$. In the *discrete time* domain, a random process X on state space S is a sequence of random variables each taking values in S, i.e., $X = \{X_n \in S : n \geq 0\}$. We call the value assumed by random variable X_n the state of X at time instant n. X is *Markovian* (or a *Markov chain*) if its future evolution depends only on current state. More formally, if X is Markovian, then for all $i, j \in S$ and whatever history X has before time $n > 0$[1], $Pr(X_{n+1} = j \mid X_n = i, (X_{n-1}, \cdots, X_0)) = Pr(X_{n+1} = j \mid X_n = i)$.

If the probabilities governing X are independent of time, X is *time homogeneous*. So, if X is Markovian and $Pr(X_{n+1} = j \mid X_n = i)$ is independent of time n, then X is time homogeneous. 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* or simply the *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 in the cycle are identical, the cycle is called *simple*.

States that may be visited by X infinitely often over the time are called *recurrent*. The remaining states of S are called *transient*. 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*. Notice that all states of an irreducible Markov chain X communicate.

Let us assume $M = (S, P)$ is an irreducible finite state Markov chain in discrete time where S is its state space and P is the *1-step transition matrix*. For any given state of S, there is a positive integer such that M can be in the given state only at time instants which is multiple of that positive integer. The maximum such integer is called the *period* of M.

If M has period 1 it is called *aperiodic*, and it tends to stabilize in the sense that the probability for M to be in any state $i \in S$ converges as time progresses. Formally, $\lim_{n\to\infty} P\{X_n = i\}$ converges to a constant called the stationary probability of state i denoted by π_i. If M has period $d > 1$, it is called *periodic* and the average of probability of M being in state i over the time, i.e., $\lim_{n\to\infty} \frac{1}{n} \sum_{k=0}^{n} P\{X_k = i\}$, converges to a constant. Let this constant be denoted by π_i as well. Regardless of its periodicity (periodic or aperiodic), M has the following nice property: $\pi = \pi P$, where π is the row vector, $(\pi_1, \pi_2, \cdots, \pi_N)$, and $\sum_{i=1}^{|S|} \pi_i = 1$. π is called the *stationary distribution* of M.

2.2 The Power method

The Power method is one among many well known methods to compute the largest *eigenvalue* (in magnitude) and its associated *eigenvector* of a square matrix.

Let us first recall some terminologies in basic matrix theory since they will be encountered several times in this paper. An eigenvalue of square matrix P is a root of the equation $det(\lambda I - P) = 0$. If λ is an eigenvalue of P, equation $\mathbf{x}(\lambda I - P) = 0$ has a solution which is unique up to a non-zero multiplicative factor. This solution is called the eigenvector of P associated with λ. Matrix P has N eigenvalues if N is the number of its columns (or rows). Let λ_i, $1 \leq i \leq N$, be the N eigenvalues of P and let them be ordered in the non-increasing order of their magnitude such that $|\lambda_i| \geq |\lambda_{i+1}|$ for all $1 \leq i \leq N - 1$.

To compute the *largest* eigenvalue and its corresponding eigenvector, the Power method iterates as follows until it satisfies some convergence criterion:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} P \text{ for } k > 0 \qquad (1)$$

where $\mathbf{x}^{(0)}$ is the initial guess. To avoid possible overflow (in case $|\lambda_0| > 1$) or underflow (in case $|\lambda_0| < 1$) during iteration, the resulting vector may have to be normalized after each or several iterations.

From a matrix point of view, π, the stationary distribution of an irreducible Markov chain $M = (S, P)$,

[1] No history if $n = 0$.

is the eigenvector of P corresponding to the eigenvalue 1 since $\pi = \pi P \Longleftrightarrow \pi(I - P) = \mathbf{0}$. Moreover, P has $|S|$ eigenvalues satisfying $1 = \lambda_1 > \lambda_1 \geq \lambda_2 \cdots \lambda_{|S|} \geq 0$. Or we may say, π is the normalized eigenvector of P corresponding to the largest eigenvalue λ_1. Therefore, the Power method may be used to iteratively compute the stationary distribution of M. Moreover, since $\lambda_1 = 1$, no normalization is necessary throughout the iteration process.

3 State compression based analysis

The goal of this paper is to speed up Markovian analysis by focusing on part of the state space rather than the entire space. More precisely, let $M = (S, P)$ be the original Markov chain with large state space. In order to obtain its stationary distribution π, we first restrict M to a subset of states, S' ($S' \subset S$), obtaining a new Markov chain $M' = (S', P')$. This step is called *state compression*. Then, 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 three 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 smaller state space S' whose stationary distribution has important but simple relationship to part of the stationary distribution of M. 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.

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 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:

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

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 1-step transition probabilities. Figure 2 shows the *STG* and the transition matrix of a Markov chain with 3 states.

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.

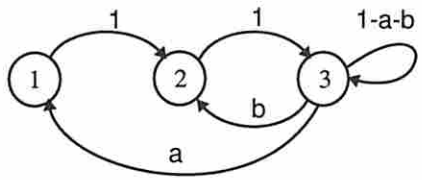
$$\begin{array}{lcllcl} P_{S'} & = & \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} & P_{S'\overline{S'}} & = & \begin{pmatrix} 0 \\ 1 \end{pmatrix} \\ P_{\overline{S'}S'} & = & \begin{pmatrix} a & b \end{pmatrix} & P_{\overline{S'}} & = & \begin{pmatrix} 1-a-b \end{pmatrix} \\ \pi_{S'} & = & (\pi_1, \pi_2)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

□

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. Later, we will 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 skip all other time instances when M is not in any state of S'. Note



(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.

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 4-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:

M_n :	1	2	3	3	2	3	1	2	$\cdots$
time n :	0	1	2	3	4	5	6	7	$\cdots$

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

X_n :	1	2	2	1	2	$\cdots$
time n :	0	1	2	3	4	$\cdots$

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

More precisely, let us define the new random process X as below:

$$X = \{X_n = M_{\tau(n)} : n \geq 0\} \tag{2}$$

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 *Process X is a time-homogeneous irreducible Markov chain.*

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 μ is a positive 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*

$$P' = P_{S'} + P_{S'\overline{S'}}(I - P_{\overline{S'}})^{-1}P_{\overline{S'}S'}. \tag{3}$$

Example: State compression. Continuing previous example, we know that Equation 2 defines a new Markov chain $M' = (S', P')$ where $S' = \{1, 2\}$ and its transition matrix P' is determined by Equation 3. That is,

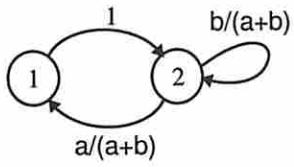
$$\begin{aligned} P' &= P_{S'} + P_{S'\overline{S'}}(I - P_{\overline{S'}})^{-1}P_{\overline{S'}S'} \\ &= \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} + \\ &\quad \begin{pmatrix} 0 \\ 1 \end{pmatrix}\left[\begin{pmatrix} 1 \end{pmatrix} - \begin{pmatrix} 1-a-b \end{pmatrix}\right]^{-1}\begin{pmatrix} a & b \end{pmatrix} \\ &= \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ \frac{a}{a+b} & \frac{b}{a+b} \end{pmatrix} \\ &= \begin{pmatrix} 0 & 1 \\ \frac{a}{a+b} & \frac{b}{a+b} \end{pmatrix} \end{aligned}$$

We may draw the labeled *STG* for M' as shown in Figure 3. Moreover, we can use direct method (or power method) to obtain $\pi' = (\pi'_1, \pi'_2) = \frac{1}{2a+b}\begin{pmatrix} a & a+b \end{pmatrix}$. In fact, if we compute π of M 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, one may verify that $\pi_{S'} = \frac{1}{1+2a+b}\begin{pmatrix} a & a+b \end{pmatrix} = \frac{2a+b}{1+2a+b}\left[\frac{1}{2a+b}\begin{pmatrix} a & a+b \end{pmatrix}\right] = \mu\pi'$, where $\mu = \frac{2a+b}{1+2a+b}$. □

3.1.2 Finding a proper S'

We have now established that given a time-homogeneous irreducible Markov chain [2] $M = (S, P)$,

[2] Let us ignore the degenerated case where the given Markov chain $M = (S, P)$ has only one state, i.e., $|S| = 1$ where M stays in that state with probability one all the time.



(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'.

we may create a new Markov chain $M' = (S', P')$ with a smaller state space $S' \subset S$ 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 can take intolerably long CPU time and form a new bottleneck. The remainder of this subsection show how we may avoid this problem by choosing a proper state space S' for the new Markov chain.

First, let us note that in the proof of Theorem 3.2 we have shown $\lim_{k\to\infty} P^k_{\overline{S'}} = 0$. 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 $P^*_{\overline{S'}}$.

Should 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.,

$$(I - P_{\overline{S'}})^{-1} = I + P_{\overline{S'}} + P^2_{\overline{S'}} + \cdots + P^{K-1}_{\overline{S'}} \qquad (4)$$

However, this is not generally true. Specifically, 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.

From the above discussion, we see that if S' is chosen in such away that there is no possible 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 Equation 4 without an error. This is a key observation of this paper.

Ensuring there is no cycle in subset S' is equivalent to finding a subset of S such that every simple cycle in S contains 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 the graph and by removing all the edges connected with the vertices in this subset, the graph becomes acyclic.

Example: Choosing S' as a feedback set. In the previous example, we took $S' = \{1, 2\}$. It is not a feedback set since there is a simple cycle $(3, 3)$ remains unbroken. If states in S' and those in $\overline{S'}$ switch, we get a feedback set 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:

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

□

Generally, we would like S' to be as small as possible. There are several reasons, however, which suggest us not to 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 $\overline{S'}$. Therefore, we propose simple heuristics to efficiently find a good FVS for S' in Section 4.

3.2 Computing π'

Once we obtain the compressed Markov chain M', its stationary distribution π' is computed iteratively using Power method:

$$\pi'^{(k+1)} = \pi'^{(k)} P' \text{ for } k > 0,$$

where $\pi'^{(0)}$ is the initial distribution called the initial guess and $\pi'^{(k)}$ is the probability (distribution) vector after n-th iteration.

The number of iterations needed before the method converges depends on two factors: the quality of the initial guess, and the eigenvalues of P'. The Power

method shows geometric convergence [23], i.e.,

$$|\pi' - \pi'^{(k)}| = \mathcal{C}e^{-\alpha k} \tag{5}$$

where $\mathcal{C}$ is a constant related to the initial distribution and α is a positive constant related to the distribution of the eigenvalue of P'. The term $e^{-\alpha}$ is called the *convergence rate*. With the same constant $\mathcal{C}$, the smaller the convergence rate, the faster the convergence speed.

It is generally very 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 have 0 probability. Alternatively, one may distribute the probability evenly among all the states and make them equally probable. 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 as the default. In this case, we set $\pi'^{(0)}$ to be an all-one vector, i.e., $\pi'^{(0)} = (1, 1, \cdots, 1)$ rather than $\left\{\frac{1}{|S'|}, \frac{1}{|S'|}, \cdots, \frac{1}{|S'|}\right\}$ where $|S'|$ denotes the size of S' and do normalization when necessary.

Since the distribution of the eigenvalues relates to the exponential constant α in Equation 5, it often dominates the convergence speed of the Power method. Let us discuss this fact in a little more detail. Transition matrices P' has $|S'|$ eigenvalues. Let them be denoted by $\left\{\lambda'_1, \lambda'_2, \cdots, \lambda'_{|S'|}\right\}$ in the decreasing order of their magnitude. Moreover, 1 is a unique eigenvalue, and the magnitude of all other eigenvalues are less than 1. That is, $1 = |\lambda'_1| > |\lambda'_2| \geq \cdots \geq |\lambda'_{|S'|}|$. It is the ratio $\frac{|\lambda_1|}{|\lambda_2|} = \frac{1}{|\lambda_2|}$ that generally determines the value of α. Thus, the smaller the second largest (in terms of the magnitude) eigenvalue of P', the faster the convergence speed.

Unfortunately, obtaining the distribution of the transition matrices P and P' is very hard for it is essentially the same problem as the computation of the stationary distribution π and π'.

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 assume the eigenvalues of P are uniformly distributed in the interval $[0, 1]$ except the largest eigenvalue 1. We may further assume that the eigenvalue of P', the transition matrix after state compression, are also uniformly distributed in the interval $[0, 1]$. Then, we may deduce the probability that $|\lambda'_2| > |\lambda_2|$ to be very small when $|S'|$ is significantly smaller than $|S|$.

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 have seen Corollary 3.1 relates π' to $\pi_{S'}$. The following theorem relates π' further to $\pi_{\overline{S'}}$.

Theorem 3.3 *If π and π' are the stationary distributions of M and M', respectively, then*

$$\pi_{\overline{S'}} = \mu\pi' P_{S'\overline{S'}} P^*_{\overline{S'}} \tag{6}$$

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.

Theorem 3.3 combined with Corollary 3.1 enables us to expand the stationary distribution of M' to that of M. Recall that $P^*_{\overline{S'}}$ in Equation 6 is the reflective transitive closure of $P_{\overline{S'}}$ which has been computed during state compression step.

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

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

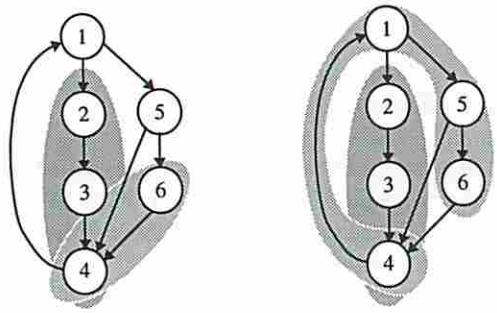
$$\begin{cases} \pi_{S'} = (\pi_3) = \mu\,(1) & \text{(by Corollary 3.1)} \\ \pi_{\overline{S'}} = (\pi_1, \pi_2) = \mu\,(1) \begin{pmatrix} a & b \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \\ \qquad\quad = \mu \begin{pmatrix} a & a+b \end{pmatrix} \text{(by Theorem 3.3).} \end{cases}$$

Following Equation 7, we compute $\mu = 1 + 2a + b$. □

In fact, the value for μ is not important. What is important is the *relative values* computed for all the state 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 Heuristics for state compression

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.



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

Figure 4: Forward- and reverse-string examples.

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 *pervious 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 visit all other states only through its sole next state. Similarly, a non-merging state can be reached only through its sole previous state.

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 has a decision state as its tail. Similarly, a reverse-string is maximal *iff* it has a merging state as its head.

Example: Forward-strings and reverse-strings. Suppose the *STG* of some Markov chain is as shown in Figure 4. By definition, every single state is both a forward-string and a reverse-string. We can list several forward-strings of length more than 1. Examples are $(2, 3)$, $(2, 3, 4)$ and $(6, 4)$. The latter two are also maximal. Example reverse-strings are $(1, 5, 6)$, $(4, 1, 5, 6)$ and $(2, 3, 4)$. 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

Let us first give two simple lemmas.

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

Lemma 4.2 *Suppose σ is a forward-string containing a state s. For every feedback set S_1 that contains s, there is another feedback set S_2 containing the same states as S_1 with s replaced by the tail of σ, i.e, $S_2 = (S_1 - \{s\}) \cup \{tail\ of\ \sigma\}$.*

Lemmas 4.1 and 4.2 effectively state that if S has maximal forward-strings, covering the tails of all maximal forward-strings results a feedback set of S. In fact, we have following theorem.

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

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 feedback set which must be the smallest. Otherwise, take all the decision states as a feedback set. The heuristics can be applied hierarchically. 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 hierarchical 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 feedback set and returns a Markov chain containing only this state. Otherwise, it takes all the decision states as a feedback set, creates a new Markov chain with the feedback set as its state space and contructs its transition matrix. Then the procedure calls itself to try compression on the new Markov chain.

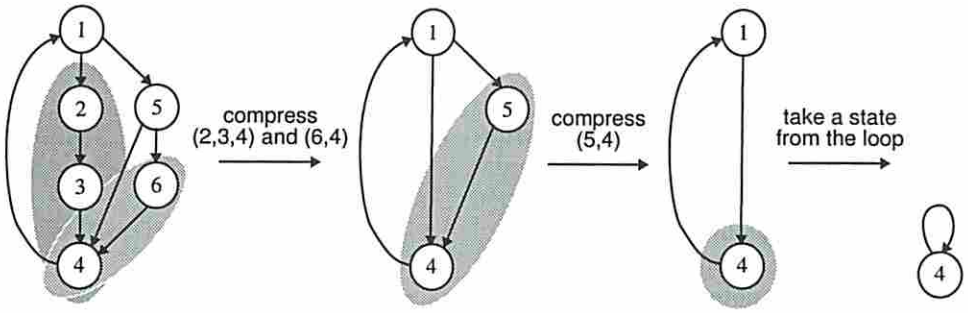


Figure 5: Forward-string-based hierarchical state compression.

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 Eqns. 3 and 4;
  String_Compress(S', P', D);
end

Figure 6: State compression based on forward-string concept.

Algorithm 4.2 State_Compression_2
input: A Markov chain $M = (S, P)$.
output: A compressed Markov chain $M' = (S', P')$.
begin
  $S'_1 := S$; $P'_1 := P$;
  D := FORWARD;
  do
    $S'_0 := S'_1$; $P'_0 := P'_1$;
    (S'_1, P'_1) := **String_Compress**(S'_0, P'_0, D);
    if D = FORWARD **then**
      D := REVERSE;
    else
      D := FORWARD;
    endif
  while$(S'_1 \neq S'_0)$;
  $M' := (S'_1, P'_1)$;
end

Figure 7: State compression based on both forward-string and reverse-string concepts.

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. These two compression ideas can be combined.

To avoid the confliction of the two compressions on a forward-string that is also a reverse-string, we do not applied them concurrently but rather *sequentially*. Figure 7 shows the algorithm that adopts both forward-string-based and reverse-string-based compressions.

The compressed Markov chain returned by Algorithm 4.2 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). Thus, in some cases Algorithm 4.1 can result less overall CPU time than Algorithm 4.2 even though it requires 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] as an alternative for the computation of stationary distribution of Markov chain models in contrast to the standard Power method previously adopted by the tool. In particular, we make several notes as follows.

First, the tool has been recently extended to handle any kind of finite state Markov chains via an efficient symbolic reducibility check. Specifically, all transient states (if any) are symbolically identified and excluded from further analysis. If there are multiple recurrent classes, the tool also performs a *transient analysis* for the limiting probability for each of them to be hit from the initial distribution. Therefore, the computation of stationary distribution is effectively done for individual recurrent classes.

Secondly, we note some technical details related to improving the speed of state compression step. Rather than computing the probabilistic transition matrix for a new Markov chain during each iteration of state compression, we compute its Boolean transition matrix which is sufficient for further compression in the sense that it is guaranteed to find the same feedback set. Only after obtaining the final feedback set S', i.e., the smallest feedback set that string-based compression can get, the probabilistic transition matrix P' of the final compressed Markov chain M' is computed according to Equations 3 and 4. 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 feedback set S' is typically much smaller than the remain state space $\overline{S'}$ so that the above treatment saves significant computation time.

Lastly, the convergence is checked after each iteration for both the standard Power method and our state-compression-based method. The iterative procedure terminates after iteration n if $\mid \pi^{(n)} - \pi^{(n-1)} \mid \leq \epsilon$, where $\pi^{(i)}$ is the probability vector after i-th iteration and ϵ is a user defined constant. That is, we check the 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 under consideration is periodic with period d, the probability vectors computed from the most recent d iterations is averaged, i.e., $\overline{\pi}^{(n)} = \frac{1}{d}\sum_{i=n-d+1}^{n} \pi^{(i)}$. It can be checked that $\lim_{n\to\infty} \overline{\pi}^{(n)} = \lim_{n\to\infty} \frac{1}{n}\sum_{i=0}^{n} \pi^{(n)}$. A convergence criterion similar to that in the aperiodic case is: $\mid \overline{\pi}^{(n)} - \overline{\pi}^{(n-1)} \mid \leq \epsilon$. Or equivalently, $\frac{1}{d} \mid \pi^{(n)} - \pi^{(n-d)} \mid \leq \epsilon$.

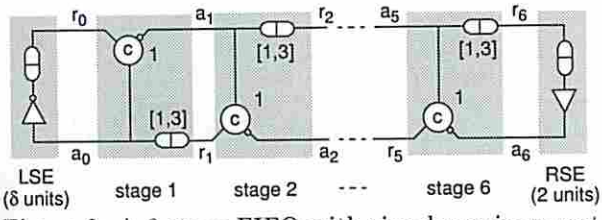


Figure 8: A 6-stage FIFO with simple environments.

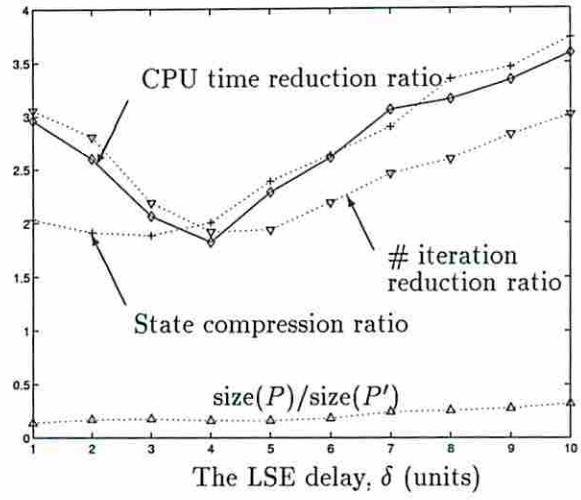


Figure 9: Speedup in FIFO analysis.

6 Experimental results

6.1 The FIFO

We first consider a FIFO with 6 stages and trivial environments as shown in Figure 8. All stages are assumed to be identical. The C-element has a 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 at the right side (RSE in sequel) acknowledges requests with a fixed delay of 2 units upon requested. The environment at left side (LSE in sequel) requests with a fixed delay of $\delta \geq 1$ units upon acknowledged. Intuitively, the shorter the delay of LSE, the longer the latency for a particular data to travel through the FIFO because it is more likely to be stalled between stages. If we think of an empty stage (free of data) as a bubble, the average number of bubbles at any time instance tends to increase as LSE gets larger delay. This suggests that the FIFO behaves more "sequentially" (toward our general notion of determinism) as LSE gets slower.

Our experiment varies the delay of the LSE from 1 unit to 10 units. We see in Figure 9 a trend of increasing state compression ratio as the delay of LSE increases. This is consistent with our intuition that the feedback set gets *relatively* smaller as the system becomes more deterministic.

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 the state compression ratio when the LSE of the FIFO has a relatively large delay (over 4 units). When the LSE is relatively fast (with a delay less than 4 units), the speedup tracks more closely the reduction ration of the number of iterations to convergence. Another interesting fact is the ratio between the sizes (in number of ADD [24] nodes) of the transition matrices P and P'. Clearly, the ADD for P' is significantly more complex than the ADD for P. However, we see the larger size of P' does not prevent the state-compression-based from achieving a significant reduction in overall CPU time. On the whole, we observe a speedup of close to or over a factor of 2. On a Sparc20 with 256M of memory, the model requires the longest CPU time (slightly over 15 minutes) to solve when the LSE assumes a delay of 4 units. 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 [25]. As in [6], we consider its estimated and back-annotated versions. The latter has more accurate specifications that account 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. 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 in both estimated and back-annotated versions possess a much higher convergence rate after state compression.

Table 2 lists the CPU times consumed. The state-compression-based method achieves over one order of magnitude in the overall CPU time reduction. Compared with the FIFO models, the DIFFEQ models requires a larger portion of the CPU time for state compression mainly because their structures allow more state compression than the FIFO models.

6.3 The pausible clocking interface

Figure 11 shows two asynchronous modules (the senders) communicating with a synchronous module (the receiver) through a variation of the *pausible clocking interface* (PCI) [26] which is built inside the receiver and guarantees failure-free synchronization.

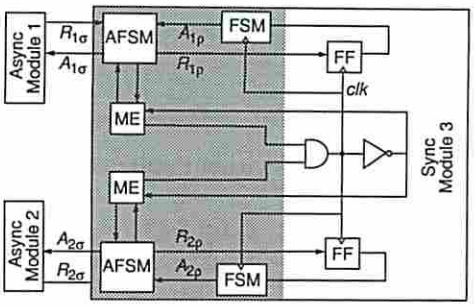
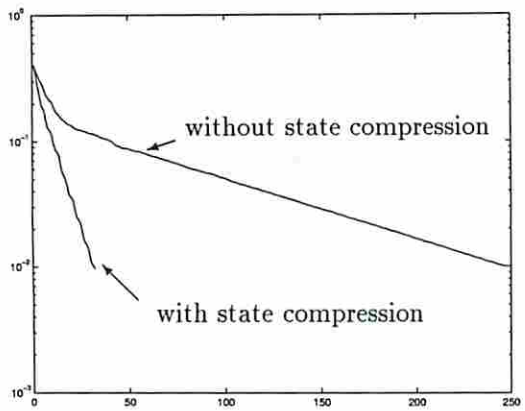


Figure 11: A pausible clocking interface.

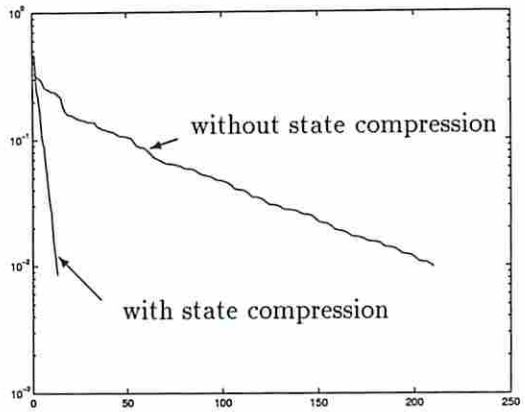
When there is no request from either of the senders, the receiver is freely clocked by the ring oscillator (formed by an inverter, two MEs and an AND gate). It continues to run freely after receiving requests from the senders if there is a clear difference between the arrival time of the requests and 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 on the model. 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, with probability 0.9 the sender issues a new request one time unit later upon acknowledged. Similarly, the delay at the receiver side is also geometrically distributed with a parameter of 0.9. That is, with probability 0.9 the receiver issues an acknowledgment at next clock edge upon requested from the Flip-flop FF. The mutual exclusion element *mutex* has a unit delay and is assumed to be fair on the 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.

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 particular, we see that in our state-compression-based method, time spent in expansion is again negligible,



(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.

but the time spent in state compression takes over half of the total CPU time as in the back-annotated version of the DIFFEQ model.

The above experiments also show that when the system model achieves a high state compression ratio (e.g., the both versions of the DIFFEQ, and the PCI models), the Power iteration is no longer the bottleneck in terms of CPU time using state-compression-basis analysis.

All the models in the above examples assume delays to be distributed in *discrete time*. However, our preliminary experiments on models that do not involve time discretization suggest that similar performance can be achieved by using our technique. In the future, we will investigate more on the performance of the technique on continuous-delay models.

7 Conclusion and future work

Markov chain models currently formalized from specifications of asynchronous system require intensive CPU time to find its stationary distribution. 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 the original state space followed by a post-processing step called expansion. We further propose the *feedback state set* to be the state space of the compressed Markov chain to reduce the amount of computation associated with state compression. Simple *string-based* heuristics have been given to find a proper feedback state set. We experimentally show that many asynchronous systems possess feedback state sets that occupy a small portion of their reachable state spaces. As a consequence, our method dramatically reduces the CPU time required, which effectively prevents the Markovian analysis from being the bottleneck of performance and power analysis in many cases.

Studying more powerful mapping 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

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.

that is sufficient for targeted analysis without performing complete reachability analysis, the likely bottleneck in the future. Finally, since the size of the state space generally grows exponentially as the system increases linearly, 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.

References

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

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

[3] Mark R. Greenstreet and Kenneth Steiglitz. Bubbles can make self-timed pipelines fast. *Journal of VLSI Signal Processing*, 2(3):139–148, November 1990.

[4] J. Ebergen and R. Berks. Response time properties of some asynchronous circuits. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC)*, pages 76–86. IEEE Computer Society Press, 1997.

[5] P. Kudva, G. Gopalakrishnan, E. Brunvand, and V. Akella. Performance analysis and optimization of asynchronous circuits. In *Proc. International Conf. Computer Design (ICCD)*, pages 221–225, 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. International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC)*, pages 64–75. IEEE Computer Society Press, 1997.

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

[8] S. Burns. *A General Approach to Performance Analysis and Optimization of Asynchronous Circuits.* PhD thesis, California Institute of Technology, 1995.

[9] J. Teich, L. Ghiele, S. Sriram, and M. Martin. Performance analysis and optimization of mixed asynchronous synchronous systems. *IEEE Transactions on Computer-Aided Design*, pages 473–484, May 1997.

[10] M. Holiday and M. Vernon. A generalized timed Petri net model for performance analysis. *IEEE Transactions on Software Engineering*, 13:1297–1310, December 1987.

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

[12] G. Rubino and B. Sericola. Sojourn times in finite markov process. *Journal of Applied Probability*, 27:744–756, 1989.

[13] G. D. Hachtel, E. Macii, A. pardo, and F. Somenzi. Markovian analysis of large finite state machines. *IEEE Transactions on Computer-Aided Design*, pages 1479–11493, December 1996.

[14] V. D. Ploeg. Preconditioning techniques for large sparse, non-symmetric matrices with arbitrary sparsity patterns. In *Proc. IMACS Symposium on Iterative Methods in Linear Algebra*, pages 173–179, 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. *Journal of Applied Probability*, 19:685–691, 1982.

[17] G. Rubino and B. Sericola. On weak lumpability in markov chains. *Journal of Applied Probability*, 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 Transactions on Circuits and Systems*, 22(1):9–15, January 1975.

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

[22] S. Park and S. B. Akers. A graph theoretic approach to partial scan design by k-cycle elimination. In *Proc. IEEE International Test Conference*, pages 303–311, 1992.

[23] J. S. Rosenthal. Convergence rates for markov chains. *SIAM Review*, 37:387–405, 1995.

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

[25] K. Y. Yun, P. A. Beerel, V. Vakilotojar, A. E. Dooply, and J. Arceo. A low-control-overhead asynchronous differential equation solver. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC)*, pages 210–223. IEEE Computer Society Press, 1997.

[26] K. Y. Yun and R. P. Donohue. Pausible clocking: A first step toward heterogeneous systems. In *Proc. International Conf. Computer Design (ICCD)*, pages 118–123. IEEE Computer Society Press, October 1996.

Appendix: Proofs

Theorem 3.1 *Process X is a time-homogeneous irreducible Markov chain.*

Proof: With the aid of the definition of τ, we have

$$\begin{aligned}
&Pr(X_n = j \mid X_{n-1} = i, X_{n-2}, \cdots, X_0)\\
&= Pr(M_{\tau(n)} = j \mid M_{\tau(n-1)} = i, M_{\tau(n-2)}, \cdots, M_{\tau(0)})\\
&= Pr(M_{\tau(n)} = j \mid M_{\tau(n-1)} = i)\\
&\quad \text{(Markovian property of } M)\\
&= Pr(X_n = j \mid X_{n-1})
\end{aligned}$$

Hence, X is Markovian.

To show it is also time-homogeneous, we prove $Pr(X_n = j \mid X_{n-1} = i) = Pr(X_1 = j \mid X_0 = i)$. Indeed,

$$\begin{aligned}
&Pr(X_n = j \mid X_{n-1} = i)\\
&= Pr(M_{\tau(n-1)+1} = j \mid M_{\tau(n-1)} = i) +\\
&\quad \textstyle\sum_{k=1}^{\infty} Pr(M_{\tau(n-1)+1+k} = j,\\
&\quad (M_{\tau(n-1)+k}, \cdots, M_{\tau(n-1)+1}) \notin S' \mid M_{\tau(n-1)} = i)\\
&= Pr(M_{\tau(0)+1} = j \mid M_{\tau(0)} = i) +\\
&\quad \textstyle\sum_{k=1}^{\infty} Pr(M_{\tau(0)+1+k} = j,\\
&\quad (M_{\tau(0)+k}, \cdots, M_{\tau(0)+1}) \notin S' \mid M_{\tau(0)} = i)\\
&\quad \text{(Time-homogeneity of } M)\\
&= Pr(M_{\tau(1)} = j \mid M_{\tau(0)} = i)\\
&= Pr(X_1 = j \mid X_0 = i). \quad \square
\end{aligned}$$

Corollary 3.1
$\pi_{S'} = \mu\pi'$, where μ is a positive constant.

Proof: For any state $i \in S'$, let $N_i(n)$ and $N'_i(n)$ be the number of visits to it up to time n by M and M', respectively. Following the definition of X, $N'_i(n) = N_i(n)$ for all time instance n. The stationary probability of state i, i.e, π', equals to the proportion of time M' has spent in state i in the limit. That is, $\pi' = \lim_{n\to\infty} \frac{1}{n} N'_i(n)$. Similarly for M, we have $\pi = \lim_{n\to\infty} \frac{1}{n} N_i(n)$. Thus, for any given states $i, j \in S'$, we have

$$\frac{\pi'_i}{\pi'_j} = \lim_{n\to\infty} \frac{N'_i(n)}{N'_j(n)} = \lim_{n\to\infty} \frac{N_i(n)}{N_j} = \frac{\pi_i}{\pi_j}. \qquad (8)$$

Moreover, by Theorem 3.1, $i, j \in S'$ are recurrent, meaning $\pi'_i, \pi'_j > 0$. Then, claim of the corollary is equivalent to Equation 8. $\square$

The proof of Theorem 3.2 relies on following lemma.

Lemma 3.1 *Let i and j be two arbitrary states of $\overline{S'}$. Given that M is currently in state i, the probability that it will not leave subset $\overline{S'}$ and after $k (k \geq 0)$ time steps it is in state j is:*

$$Pr(M_k = j \in \overline{S'}, \text{ if } k > 1, (M_{k-1}, \cdots, M_1) \in \overline{S'} \mid M_0 = i \in \overline{S'}) = (P^k_{\overline{S'}})_{ij}.$$

Moreover, this probability converges to 0 as $k \to \infty$.

Proof: (By induction)

In the cases $k = 0, 1$, the equality is trivially true. Suppose the equality holds in the cases up to $k = m \geq 1$. Then, in the case $k = m + 1$, we have

$$\begin{aligned}
&Pr(M_{m+1} = j \in \overline{S'}, (M_m, \cdots, M_1) \in \overline{S'} \mid\\
&\quad M_0 = i \in \overline{S'})\\
&= \textstyle\sum_{u\in\overline{S'}} Pr(M_{m+1} = j, M_m = u,\\
&\quad (M_{m-1}, \cdots, M_1) \in \overline{S'} \mid M_0 = i \in \overline{S'})\\
&= \textstyle\sum_{u\in\overline{S'}} Pr(M_{m+1} = j \mid M_m = u,\\
&\quad (M_{m-1}, \cdots, M_1) \in \overline{S'}, M_0 = i \in \overline{S'}) \cdot\\
&\quad Pr(M_m = u, (M_{m-1}, \cdots, M_1) \in \overline{S'} \mid M_0 = i \in \overline{S'})\\
&= \textstyle\sum_{u\in\overline{S'}} (P^m_{\overline{S'}})_{iu} (P_{\overline{S'}})_{uj} \text{ (Markovian property of } M\\
&\quad \text{and assumption on the case } k = m)\\
&= (P^{m+1}_{\overline{S'}})_{ij}.
\end{aligned}$$

Further, to assume $\lim_{k\to\infty}(P^k_{\overline{S'}})_{ij} > 0$ is to contradict the fact that M is irreducible since it would then be possible for state i not to communicate with any state in the non-empty subset S'. This completes the proof. □.

Theorem 3.2 *M' has a transition matrix*

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

Proof: From Lemma 3.1, $P^k_{\overline{S'}} \to 0$ as $k \to 0$, which implies $(I - P_{\overline{S'}})$ is invertible such that $(I - P_{\overline{S'}})^{-1} = \sum_{k=0}$. Therefore, it is sufficient to show that

$$\begin{aligned} &Pr(M'_1 = j \mid M'_0 = i) \\ &= (P_{S'})_{ij} + (P_{S'\overline{S'}}(\textstyle\sum_{k=0}^{\infty} P^k_{\overline{S'}})P_{\overline{S'}S'})_{ij} \end{aligned}$$

for all $i, j \in S'$. Indeed,

$$\begin{aligned} &Pr(M'_1 = j \mid M'_0 = i) \\ &= Pr(M_{\tau(1)} = j \mid M_{\tau(0)} = i) \\ &= Pr(M_{\tau(0)+1} = j \mid M_{\tau(0)=i}) + \\ &\quad Pr(\tau(1) > \tau(0)+1, M_{\tau(1)} = j \mid M_{\tau(0)} = i) \\ &= Pr(M_1 = j \mid M_0 = i) + \textstyle\sum_{k=\tau(0)+2}^{\infty} Pr(M_k = j, \\ &\quad (M_{k-1}, \cdots, M_{\tau(0)+1}) \notin S' \mid M_{\tau(0)} = i) \\ &= (P_{S'})_{ij} + \textstyle\sum_{k=2}^{\infty} Pr(M_{k-\tau(0)} = j, \\ &\quad (M_{k-\tau(0)-1}, \cdots, M_1) \notin S' \mid M_0 = i) \\ &\quad \text{(time homogeneous property of } M) \\ &= (P_{S'})_{ij} + \textstyle\sum_{k=2}^{\infty} Pr(M_k = j, \\ &\quad (M_{k-1}, \cdots, M_1) \notin S' \mid M_0 = i) \\ &= (P_{S'})_{ij} + \textstyle\sum_{k=2}^{\infty} \sum_{u,v\in\overline{S'}} \\ &\quad Pr(M_k = j \mid M_{k-1} = v) \cdot \\ &\quad Pr(M_{k-1} = v \in S' \mid M_1 = u) \cdot \\ &\quad Pr(M_1 = u \in S' \mid M_0 = i) \\ &\quad \text{(condition } M \text{ at time } k-1 \text{ and } 1, \\ &\quad \text{and apply its Markovian property)} \\ &= (P_{S'})_{ij} + \textstyle\sum_{u,v\in\overline{S'}} \\ &\quad Pr(M_1 = j \mid M_0 = v) \cdot \\ &\quad \textstyle\sum_{k=0}^{\infty} Pr(M_k = v \in S' \mid M_0 = u) \cdot \\ &\quad Pr(M_1 = u \in S' \mid M_0 = i) \\ &\quad \text{(exchange summations,} \\ &\quad \text{and apply time-homogeneous property of } M) \\ &= (P_{S'})_{ij} + (P_{S'\overline{S'}}(\textstyle\sum_{k=0}^{\infty} P^k_{\overline{S'}})P_{\overline{S'}S'})_{ij} \\ &\quad \text{(Lemma 3.1)} \end{aligned}$$

□

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:

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

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

Proof: Partitioning π into $(\pi_{S'}, \pi_{\overline{S'}})$ and P into $\begin{pmatrix} P_{S'} & P_{S'\overline{S'}} \\ P_{\overline{S'}S'} & P_{\overline{S'}} \end{pmatrix}$, we have $\pi_{\overline{S'}} = \pi_{\overline{S'}}P_{\overline{S'}} + \pi_{S'}P_{S'\overline{S'}}$ from the fact that $\pi = \pi P$. That is to say,

$$\pi_{\overline{S'}}(I - P_{\overline{S'}}) = \pi_{S'}P_{S'\overline{S'}}.$$

But $(I - P_{\overline{S'}})$ is invertible and it equals to $P^*_{\overline{S'}}$, the reflective transitive closure of $P_{\overline{S'}}$, we get

$$\pi_{\overline{S'}} = \pi_{S'}P_{S'\overline{S'}}P^*_{\overline{S'}} = \mu\pi' P_{S'\overline{S'}}P^*_{\overline{S'}}$$

The second half of the above equation is due to Corollary 3.1.

The proof concludes with the fact that the normalization requirement $\pi\mathbf{1}^T = 1$ is satisfied if and only if μ is as defined. □

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

Proof: Suppose σ is a maximal forward-string of M. Then, M has at least one decision state which is the tail of σ. Next, the length of σ must be bounded by $|S|$ because otherwise there is a state appears multiple times on σ such that σ contains a loop. Unfolding that loop makes σ not a maximal forward-string, contradicting the assumption. Finally, if s is a decision state of M, then very string ending with s is maximal with respect to its head state. The irreducibility assumption on M requires s be visited infinitely often so that there is a forward-string leading to s. □

Lemma 4.2 *Suppose σ is a forward-string containing a state s. For every feedback set S_1 that contains s, there is another feedback set S_2 containing the same states as S_1 with but s replaced by the tail of σ, i.e, $S_2 = (S_1 - \{s\}) \cup \{tail\ of\ \sigma\}$.*

Proof: Every cycle of S containing s also contains the tail of σ. □

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

Proof: If S has no decision states, the result is trivial. Suppose S has decision states. Suppose further a feedback set of S has a non-decision state s. By Lemma 4.1, all maximal forward-strings in S (including those containing s) have a bounded length with a decision tail state. The proof concludes immediately with Lemma 4.2. □