Accelerating Markovian Analysis of Asynchronous Systems using State Compression

Aiguo Xie, *Student Member, IEEE*, and Peter A. Beerel, *Member, IEEE*

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 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 that contain mostly bounded-delay components. 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.**

I. 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 is applicable to both areas, however, to simplify the exposition we focus on performance analysis.

There are currently two main categories of approaches to 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 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 derive average metrics for some classes of systems, e.g., those modeled by marked graphs or extended marked graphs [1], [3], [7], [8], [9], [10]. Average performance metrics may also be derived for some systems (e.g., with FIFO-like structures) if all delays are exponentially distributed in which case classic queueing theory can be applied [3].

Bounding techniques are often faster than probabilistic techniques. This is mainly because the former typically analyze systems at their more compact event-level and do not need to perform state-level analysis which is often necessary for the latter and is more time consuming. However, the average performance measure may be more useful to guide design. This is particularly true if the computed performance 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 of the probabilistic approaches is Markovian analysis. In Kudva *et al.*'s approach, systems are specified in a generalized timed Petri-net model [11] whereas in Xie and Beerel's approach, they are specified using an SMV [12] based discrete-timed model. Both approaches convert their models into finite state Markov chains, i.e., random processes with short memory (whose current states completely determine their future 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 delays in Xie and Beerel's model). This conversion is proceeded 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 mean *sojourn time* [13], the average duration the system remains in a subset of the state space, from which *average time separations 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* (e.g., [14]) 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 have 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

This work was funded in part by a NSF CAREER Award MIP-9502386, NSF Grant No. CCR-9812164, a gift from the Intel Corporation, and a research seed grant from the James H. Zumberge Faculty Research and Innovation Fund at USC.

The authors are with the Department of Electrical Engineering–Systems, University of Southern California, Los Angeles, CA 90089-2562. E-mails: aiguo@jungfrau.usc.edu, pabeerel@eiger.usc.edu.

distribution. A promising way to speed up this phase of the analysis is to use *symbolic techniques* [6], [15]. 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 to Markovian analysis, including *preconditioning* [16] and *orthogonal multi-vector iteration* [17]. It is unclear whether these sophisticated techniques are likely to yield large improvements, however, because they impose significant overhead. In addition, there is an intensive ongoing research trend in probability theory on the *lumpability* of Markov chains [18], [19]. 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. Specifically, the lumpability approach partitions the state space into subsets and treats each subset as a single state such that the resulting process retains the Markov property. A necessary and sufficient condition for this to be possible is all states within each of the subsets have the same transition behavior with respect to all other subsets. However, this is a rather restrictive condition. In general, it is often not possible for a real design to satisfy this condition and get a much smaller Markov chain [20]. Therefore, we believe that this approach cannot be considered as a general method to reduce the computational complexity.

The method we propose here takes advantage of the fact that most asynchronous systems consist mainly of components with bounded delay (i.e., its is rare that a component in an asynchronous system has unbounded delay, a notable exception being an arbiter element in metastability [21]). Using time-discretization to model these systems as Markov chains leads to state spaces that have *limited feedback*. More precisely, we partition the (reachable) state space into two subsets, say, 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. That is, B is a *feedback vertex set* (FVS) of states. We observe that time-discretization of asynchronous systems with mostly components that have bounded delays leads to models with small feedback vertex sets. This common feature of asynchronous system models motivates us to speed up the Markovian analysis by focusing on the FVS.

More specifically, we use standard techniques to define 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. Creating this Markov chain 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. We call this third step *expansion*. All these steps can be implemented using symbolic techniques. Fig. 1 depicts an overview of our method.

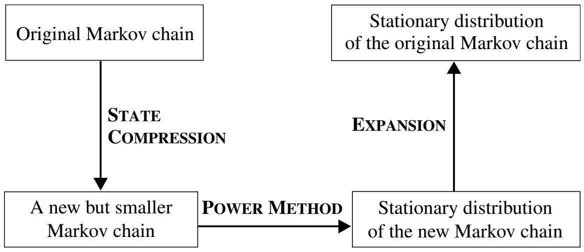


Fig. 1. Block diagram of our method.

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 $\mathcal{NP}$-complete problem [22]. Many heuristic algorithms to find a minimal FVS [23], [24], [25] exist, but they are all too expensive for large graphs. We give two simple heuristics to symbolically find a good FVS which may not be the minimum.

There are two significant contributions of this approach. First, the reduction of the stationary analysis of a Markov chain to that of a smaller Markov chain defined over a FVS, although uses standard techniques, is novel and can be efficiently implemented using symbolic techniques. Second, the proposed heuristics for symbolically finding a FVS are novel and efficient. Application of our approach to several real asynchronous 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 possible using the method in [6].

The most direct application of this work is to speed-up our performance analysis tool that computes the average time separation of events in discretized-time models of asynchronous circuits [6]. This tool can compute the average latency, throughput, or cycle-time of asynchronous architectures, thereby enabling the designer to make better choice among alternative architectures. It is also possible to use Markovian analysis to determine which of two inputs to a gate is likely to be later than the other, thereby helping enable the automation of performance-driven transistor sizing and ordering. Markovian analysis can also be used to determine the relative frequency of being in different states of the system, which can be useful to determine the average power consumption of asynchronous and synchronous circuits [15], [26]. More generally, Markovian analysis is heavily used in the performance analysis and scheduling of networks and distributed systems (e.g., [11], [27], [28]). As in our application, the state-of-the-art for all these applications is limited by exploding computational complexity. We briefly discuss the different characteristics of those Markov chains and under what conditions our approach may be helpful to their analysis.

Section II briefly reviews the basics of Markov chains. Section III discusses various methods of system modeling using Markov chains and the approach we adopted. This section also identifies characteristics of these asynchronous systems that lead to models with a small FVS and thus would most benefit from our proposed approach. The classic Power method to obtain stationary distributions is reviewed in Section IV. Section V details our method as well as the theorems we have proved to check its correctness. Section VI describes our heuristic algorithms to find the feedback set that forms the state space of the new Markov chain. Implementation issues are noted in Section VII and experimental results are given in Section VIII. We conclude this paper and outline some possible future work in Section IX. The proofs of all theorems are presented in the appendix.

II. Basics of 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. 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. 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. This property is sometimes called the *weak Markov property* in the sense that the above conditioning on the event $\{X_n = i\}$ is taken at a fixed time instant n. It is sometimes desirable to make use of a stronger property, called the *strong Markov property* [29] which deals with the situation where the corresponding conditioning is taken at a random time instant T. The strong Markov property states that if random variable T is a *stopping time*[1] of Markov chain X, then $Pr(X_{T+m} = j \mid X_T = i, (X_k = x_k \text{ for } 0 \leq k < T)) = Pr(X_{T+m} = j \mid X_T = i)$ for all $m > 0$, $i, j \in S$, and all sequence (x_k) of states.

If the probabilities governing X are independent of time, X is *time-homogeneous*. Thus, if X is a Markov chain 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 $P_{ij} = Pr(X_{n+1} = j \mid X_n = i) = Pr(X_1 = j \mid X_0 = i)$ where $\sum_{j \in S} P_{ij} = 1$ for any $i \in S$. 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], we 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* states, 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*. We call a cycle of length 2 a *self-loop* in the sense that the corresponding (head/tail) state may transit to itself in one step. In this case, we also say the corresponding state has a self-loop. Let S' be a subset of S. It is a *feedback vertex set* (FVS) if every cycle of X has at least one state in S'. In other words, the removal of S' makes the state space of X acyclic. A FVS is *minimal* if it does not contain any other FVSs. By definition, a state that has a self-loop must appear in any FVS.

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 *recurrence class* is a maximal communicating subset that contains only recurrent states. If S contains transient states or it has multiple recurrence classes, X is called *reducible*. Otherwise, X is *irreducible*. Notice 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. In the case a state has period 1, it is said to be *aperiodic*. States in the same recurrence class have the same period.

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 M to be in any state $i \in S$ converges as time progresses. Formally, the sequence $\{Pr(M_n = i), n = 0, 1, \cdots\}$ converges to a constant called the *stationary probability* of state i denoted by π_i. If M is periodic, the average probability of M being in state i over time, i.e., the sequence $\{\frac{1}{n}\sum_{k=0}^{n} Pr(M_k = i), n = 0, 1, \cdots\}$, converges to a constant. Let this constant be denoted by π_i as well. Regardless of its periodicity, M has the following nice 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 (probability) distribution* of M in the sense that M_n has the same distribution for all $(n \geq 1)$ if M_0 has distribution π. Irrespective of its periodicity, an irreducible finite state Markov chain has a unique stationary distribution.

In this paper, we assume the Markov chains to be analyzed are irreducible for the following reasons. The state classification technique developed in [30] can be used to efficiently identify transient states and all recurrence classes of a Markov chain. If the original Markov chain has only one recurrence class (which is often the case [15], [30]), then all transient states can be removed, reducing the problem to an irreducible Markov chain. If the original Markov chain has multiple recurrence classes, one may first restrict the stationary analysis to each recurrence class. If necessary, one may further compute the limiting probability for each of the recurrence classes to be hit from a given initial dis-

[1] A random variable T taking values in $\{0, 1, 2, \cdots\}$ is a stopping time of X if it is decidable whether or not $\{T = n\}$ with a knowledge only of the past and present, $X_0, X_1, \cdots, X_n$, and with no further information about the future.

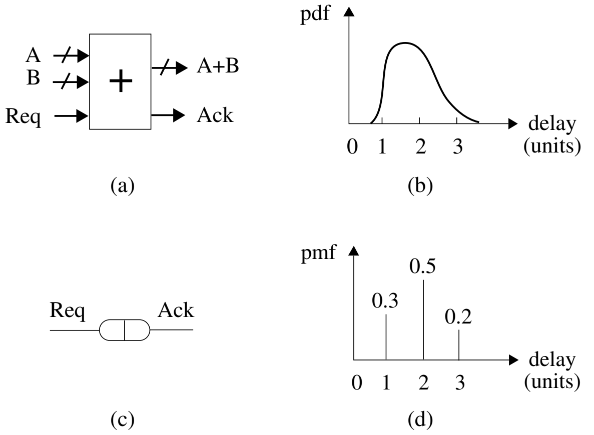


Fig. 2. Probabilistic modeling and time discretization: (a) a data-path component, (b) its delay distribution, (c) abstraction, and (d) time discretization.

tribution by performing a transient analysis. As described in Section VII, our implementation supports both these analyses.

III. Markov Chain Models of Asynchronous Systems

This section discusses performance modeling of asynchronous systems at various levels of abstraction, focusing on abstractions that contain components whose delays and output choices are probabilistically characterized. The section discusses the available options for analyzing such models and motivates the option we choose, time-discretization, which yields a discrete-time (finite state) Markov chain model. Finally, the section explains the characteristics of asynchronous systems that yield Markov chain system models with small feedback vertex sets, highlighting the characteristics of those asynchronous systems that will most benefit from our state-compression-based approach.

A. Probabilistic Modeling of Asynchronous Systems

There have been developed many formalisms to specify asynchronous systems. Traditionally, these specifications target synthesis and verification and consequently do not need to be probabilistically quantified. For instance, non-fixed component delays may be specified solely in terms of upper and lower bounds. For some special classes of systems, techniques have been developed to evaluate system performance bounds directly from these specifications, e.g., [2], [4], [8]. However, in order to obtain system characteristics such as average throughput and average power consumption, a system needs to be adequately modeled in a probabilistic sense.

We focus on the derivation of our performance models from an architectural description of the system. As explained later, a similar derivation is also possible from more abstract behavioral or more detailed gate-level descriptions. At the architectural level, like most hardware systems, an asynchronous system contains interacting data-path components and controllers.

The operation of an asynchronous data-path component is typically sequenced by *go* and *done* signals that emanate from and connect to asynchronous controllers. Consequently, the delay of the done signal determines system performance. In principle, this delay could be modeled as a random variable which depends on the values of the inputs to the component. However, the size of such a model would likely grow exponentially with respect to the bitwidth of the component, making the analysis computationally intractable. In practice, one can often achieve useful performance models by abstracting the delay of a data-path component to a random variable with an estimated probability distribution that takes into account input statistics (e.g., common v.s. rare input combinations) [6], [31]. Consequently, the actual data signals need not be modeled, thereby dramatically simplifying the complexity of the system model. As an example, Fig. 2 (b) plots the distribution of the delay of the adder shown in Fig. 2 (a). In some cases, such an abstraction may lead to significant absolute error in the performance estimates. One reason is that this abstraction does not account for the correlations between the delays of different data-path components. When necessary, however, these correlations can be explicitly accounted for by using extra random variables at the expense of a more complicated system model.

In addition, data-path components often generate signals that dictate subsequent control operations, such as the result of a comparator. These signals can also be modeled as random boolean variables whose distributions can be estimated using, for example, high-level architectural simulation (e.g., [31]).

The issues regarding modeling asynchronous controllers are made more obtuse because there is no universal specification language for asynchronous controllers. In fact, there are many different languages with varying expressiveness, including asynchronous finite state machines [32], burst mode machines [33] and its derivations (e.g., [34], [35]), various classes of Petri nets [36] such as Signal Graphs [37], Signal Transition Graphs [38], and Timed Event/Level (TEL) structures [39], and sets of connected processes (e.g.,[40], [41]) that are variants of communicating sequential processes (CSP) [42].

For performance modeling, these formalisms must be augmented with delays and their distributions. For example, one performance model for burst-mode controllers specifies the latencies of each state transition as a constant [6]. In order to incorporate the stochastic effects of the variation in chip operating temperatures and/or the manufacturing process, the relevant delays may be modeled probabilistically. Even under constant temperature and nominal process assumptions, some controllers do exhibit non-fixed delays and should be modeled probabilistically. For instance, the delay of an arbiter in metastability may be modeled as an exponential distributed random variable with proper parameters [21]. In addition, the probability distributions of input (environmental) choices can be modeled in the same manner as we model the data-path output signals (such as the result of a comparator) that dictate the controller behavior.

The above probabilistic specifications lead to a performance model consisting of a set of random variables which are related one to another by the system logic. Such a model essentially defines a stochastic process whose state space is spanned by all these random variables. The process evolves according to the distributions of these random variables and their interdependencies. The larger the system to be modeled and the higher the desired accuracy of the model, the more the random variables required. The key issue we now address is how to efficiently analyze such models.

B. Handling Arbitrarily Distributed Delays using Approximations

The stochastic process defined by the interdependent random variables used to probabilistically model an asynchronous systems (as discussed in the previous subsection) may evolve in a very complex way. This is mainly because the system component delays often have arbitrary distributions, and consequently their behavior cannot be *exactly* captured by any known stochastic process that can be efficiently analyzed using existing methods. For this reason, many researchers proposed methods to approximate arbitrary distributions with some special distributions that facilitate more efficient analysis. This subsection briefly reviews these approaches and the next subsection describes the approach we adopted.

One approach approximates the arbitrary distribution using an exponential distribution with a proper parameter. In some cases, the resulting system model can be treated as a memoryless queueing network [3]. In other cases, the resulting model can be described as a stochastic or timed Petri net [11], [43], [44]. Both these types of memoryless models can be analyzed as a finite state Markov chain where the states in the Markov chain have appropriately defined semantics [11], [43], [44]. For example, for queueing networks a state can represent the numbers of jobs in the servers while for Petri nets a state can represent a token distribution.

A second approach approximates the arbitrary distribution using a fixed delay. For example, the fixed delay can be chosen to be the mean of the arbitrary distribution. This type of delay models is incorporated into Generalized Timed Petri Nets (GTPNs) which can also be converted into a Markov chain by augmenting a notion of time into the semantics of a state [11]. For example, the state can be associated with a fixed time period after which the Markov chain transits to some next state.

A third approach approximates the arbitrary distribution using time discretization (see e.g., [44], [45]). The result is that events can now occur only at time instants that are multiple of some unit time step. That is, the delay interval (of reals) is discretized. The probability distribution function (pdf) is accordingly converted into a probability mass function (pmf). Similar to the other two approaches, time discretization also loses some accuracy [45]. However, it is always possible to trade discretization resolution (and consequently the computational cost) with accuracy of the final performance estimates. The time discretized system model can also be converted to a finite state Markov chain (e.g., [44]).

C. Modeling Asynchronous Systems using Time Discretization

We choose to model asynchronous systems using time discretization for two reasons. First, we believe it is relatively uncommon that the delay of an event can be closely approximated using an exponentially or geometrically distribution. A notable exception is the delay an arbiter experiences when resolving metastability [21]. Another exception may be the data inter-arrival time from the environments which sometimes may be reasonably approximated with an exponential distribution. Beyond these cases, most system components (e.g., data-path components and controllers) exhibit bounded delays with arbitrary distributions which cannot be reasonably modeled as exponentially distributed random variables. Moreover, when exponential distributions are needed, they can be approximated with arbitrary accuracy in discrete time using a discrete geometric distributions. Second, we believe that it is often inadequate to model bounded delays with their mean values, as was proposed in [5]. For example, it can be shown that even for systems with delays having reasonably narrow bounds and nice distributions, this approximation can result in rather inaccurate performance estimates [46]. For instance, the average throughput of a micropipeline strongly depends on the second moment or the variance of the stage delays [46].

As an example of time discretization, we may discretize the adder delay whose distribution is shown in Fig. 2 (b) and model it as a discrete random variable whose pmf is shown in Fig. 2 (d). In the system model, such a discrete random delay variable is also considered as a state variable. To distinguish such state variables (modeling component delays) from those modeling system signals, we call them *timers*. The next state value of a timer is probabilistically assigned when a corresponding system signal is enabled. Once set, the value of a timer decreases by 1 each time the system model makes a state transition. When it reaches 0, the corresponding system signal simultaneously fires (i.e., makes a desired transition).

The semantics of the time-discretized system model is (1) state transitions are instantaneous, and (2) once a state is entered the system makes a state transition and enters another state (possibly the same state if the state has a self-loop) after precisely one time unit.

As a concrete state space example, let us consider again the abstract adder model in Fig. 2 whose discretized delay distribution is redrawn in Fig. 3 (a). Fig. 3 (b) shows its (local) state transition graph (STG) where a state represents the value of the `Ack` signal and an integer `timer`. Suppose the adder is currently in state (00) and detects a request signal (e.g., the current state value of `Req` is 1) which enables a positive transition on signal `Ack`. According to the delay distribution of the adder, the next state

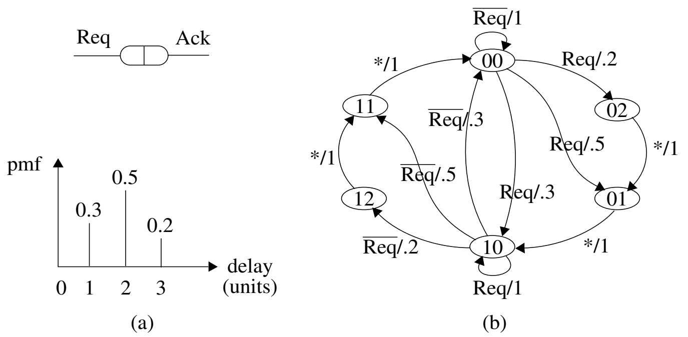


Fig. 3. Local STG of an abstracted adder. (a) Its probabilistic delay model, (b) its state space where `state := (Ack, timer)` and `transition label := input-condition/probability`.

value of `timer` is then randomly assigned. Specifically, the next state value of `timer` is set to 2 with probability 0.2, a value of 1 with probability 0.5, and a value of 0 with probability 0.3. If `timer` is assigned a value of k ($0 \leq k \leq 2$), then the desired positive transition on `Ack` will occur $k+1$ time steps after the adder detects a positive transition on `Req`. For instance, if `timer` is set to 2 at state (00), then the model transits to state (02) after one time step. After another time step, the model transits from state (02) to state (01) with probability 1. After one more time step, it transits from state (01) to state (10) with probability 1, simultaneously firing `Ack` and setting `timer` to 0. The discrete-time Markov chain model of an entire system is derived by composing the STGs of all system components.

Experiments show that such discrete-time Markov models can yield high-quality system performance estimates (compared with extensive simulation) with fairly coarse time discretization [6]. Note, however, that our main goal is to guide architectural design choices and for this purpose it is more important for our models to accurately estimate the relative performance of different architectural alternatives rather than their absolute performance. In other words, relative accuracy is more important than absolute accuracy. Our experiences suggest that discrete-time models can provide fairly reliable relative performance estimates. We also note that similar conclusions have been made for timed-discretized models of queueing networks [47] and less accurate GTPN models of asynchronous systems [5].

In our tool flow, we derive architectural-level performance models specified using a combination of behavioral burst-mode FSMs and structural VHDL annotated with discrete delay distributions. Using abstraction and time-discretization as described above, we create the Markov chain (see [6] for details). Translations from other specification formalisms and other levels of design abstraction are also possible. For example, systems specified using VHDL at the behavioral level can also be converted into a Markov chain model using abstraction and time-discretization. Finally, it should be noted that in all these case, our notion of a state represents the values of some system signals. More abstract notion of a state is possible for some special class of systems such as a micropipeline with exponentially distributed delays where a state can represent the numbers of jobs in all the stages. However, for systems with more general structures and delay types, it is unclear whether such an abstract state space will satisfy the Markov property.

D. The Derived Markov Chains

In this subsection, we analyze the the number of states with self-loops and the size of minimum feedback vertex state set of the derived Markov chains for asynchronous systems. We contrast these characteristics with those of Markov chains obtained from synchronous counterparts and traditional queueing models.

Fraction of states with self-loops:

First, we observe that Markov chains derived from asynchronous systems that have mostly bounded-delay components have a small fraction of states that possess a self-loop or *self-loop states*. To reason why, we rely on the concept of *sojourn time* [29] (sometime called holding time) of a state s. It can be shown that if state s has a self-loop with corresponding transition probability p which is strictly positive, then the sojourn time of s is geometrically distributed with parameter p. In other words, state s has an unbounded sojourn time. If, on the other hand, s does not have a self-loop (equivalently, $p = 0$), its sojourn time is one unit of time.

As a consequence, if the Markov chain is currently in a state which has a self-loop, then only events with unbounded firing times can be enabled. Moreover, to have a self-loop, the unbounded delay event cannot be concurrently enabled with a bounded delay event, further reducing the number of self-loop states. Consequently, the fraction of self-loop states is usually small.

In contrast, Markov chains modeling synchronous circuits for power analysis usually have a large fraction of self-loop states (e.g., [15], [48]). The reason is that they target the long-term probability of state transitions where a state is a vector of Boolean values representing the contents of all latches. Since many external input changes do not cause a synchronous system to change state, most states have self-loops.

In addition, traditional applications of Markov chains in system performance or reliability analysis often lead to models with a large fraction of self-loop states. One example is in the memoryless queueing network models (such as $M/M/m/n$ type) where data arrival and stage service times are assumed to be exponentially distributed. In these cases, almost all states of the underlying Markov chain have a self-loop. Interestingly, more recent research in queueing theories is focused on removing these restricted timing assumptions which are often found to be unrealistic. Finally, depending on the number of transitions with deterministic delays, both GTPNs and DSPNs widely used in other application areas demonstrate various amount of self-loop states.

Size of feedback vertex set:

Recall that a subset of states is a feedback vertex set (FVS) if the removal of these states makes the remaining state space acyclic. We have empirically seen that Markov chains of asynchronous systems with mostly components having bounded delays have a small FVS. There are at least two reasons which we describe below.

The first reason is that the state space of the resulting Markov chain usually has few states with self-loops as discussed above. This may be considered as a necessarily condition for a small FVS to exist since every self-loop state must be in any FVS.

The second reason is that the state space tends to be sparse in the sense that it contains many directed acyclic graphs (DAG) like structures. We know that if the root state of a DAG is in a FVS, all other states of that DAG need not be in the FVS. Below, we list two sub-reasons that lead to the existence of these DAG-like structures in our Markov chain models.

The first sub-reason is again related to time discretization. As discussed earlier, for each system event that has a bounded delay, a dedicated integer-valued timer is used to denote the corresponding passage of time. Suppose that e is such an event with its delay upper bounded by n time steps and that it can be enabled in state s. Then, there is a DAG-like structure rooted at state s which itself may have a self-loop. The DAG has a depth (the length of its longest path from the root) of at least n. Let this subgraph be called the *enabled* DAG of event e in state s.

As an example, let us consider the STG of the abstract adder model in Fig. 3. Since `Req` may make a positive transition when the model is in state (00), event `Ack`$\uparrow$ can be enabled in state (00). Moreover, `Ack`$\uparrow$ can take up to 3 time steps to occur once it is enabled. Thus, we claim there must be an enabled DAG of event `Ack`$\uparrow$ rooted at state (00) which has a depth of 3. Indeed, states (00), (02) and (01) correspond to an enabled DAG of depth 3.

Note further that the depth of the enabled DAG of event e cannot be decreased if there are other events (with or without bounded delays) enabled simultaneously in state s. Moreover, even if there are other events that can be enabled in a state within the enabled DAG of event e, this depth is not decreased. In contrast, the enabled DAGs of these events combined with the enabled DAG of event e may form a larger DAG that contain more states.

As a result, there can be many DAG-like structures in the state space, which often leads to the existence of a small FVS.

The second sub-reason is that the components of asynchronous systems often become active in a sequential fashion, which implies there are few and long cycles in the state space. Examples are the *differential equation solver* [49], *pausible clock interface circuits* [50]. Somewhat different examples are heavily loaded micropipelines where stages are busy on processing data most of the time. However, even for such micropipelines, there always exists a FVS which contains at most half of the states in the state space. In fact, there exists a FVS with decreasing size as the input load gets reduced. We will discuss all these examples in more detail in Section VIII.

The existence of small FVSs of asynchronous systems that have mostly bounded delays serves as a key motivation of this work. It is this fact that we exploit to speedup the Markovian analysis of such systems. It is an open question whether models in other applications will have a small FVS. We argued above that models with many discretized bounded delays or fixed delays will have a small fraction of self-loop states. In addition, the upper bounds of these delays impact the size of the DAG-like structures. On top of these two characteristics, the size of the FVS also depends on how the system components communicate. Consequently, the impact of this work on other application areas that use Markovian analysis is a subject of future work.

IV. Stationary Analysis: the Power Method

In this section, we review a traditional iterative method, the Power method (see e.g., [14]), to obtain the stationary distribution of a large irreducible Markov chain. We will use this method to solve a reduced Markov chain in Section V.

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

Let us first recall some terms in basic matrix algebra. Let A be a square matrix, a number λ (possibly complex) is an eigenvalue of A if there exists a non-zero vector ξ such that $\xi A = \lambda \xi$. Thus, ξ is a (left-hand) eigenvector of A corresponding to eigenvalue λ. In general, 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 also called *simple*.

Suppose eigenvalues of A satisfy the following condition:

$$|\lambda_1| > |\lambda_{m+1}| \geq \cdots \geq |\lambda_N| \text{ and } \lambda_1 = \cdots = \lambda_m.$$

That is, A has a *unique* dominant eigenvalue which is m-multiple. Then, the Power method can 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.,

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

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

A. The Aperiodic Case

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: $1 = \lambda_1 > |\lambda_2| \geq \cdots |\lambda_{|S|}|$ [29]. From (1), we see the stationary distribution of M is an eigenvector of P which corresponds to the simple dominant eigenvalue 1. Thus, π can be iteratively computed using the Power method as follows:

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

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

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.

B. The Periodic Case

If the irreducible Markov chain M has a period $d > 1$, the transition matrix P has not only eigenvalue 1 but also other eigenvalues which lie on the unit circle. Thus, the former is no longer the unique dominant eigenvalue. Therefore, the simple Power method cannot be used to compute π although π is still an eigenvector of P corresponding to eigenvalue 1 according to (1). In fact, the sequence $\{\pi^{(n)} : n \geq 0\}$ obtained from the iteration described by (2) oscillates.

However, the sequence $\{\pi^{(nd+r)} : n \geq 0\}$ converges for a fixed $r \in \{0, 1, \cdots, d-1\}$ [51]. Denoting $\pi(r) = \lim_{n\to\infty} \pi^{(nd+r)}$, we have $\pi(r) = \pi(r)P^d$ by the Chapman-Kolmogorov equations [29], [51], where P^d is the d-th power of P. Hence, $\pi(r)$, ($r \in \{0, 1, \cdots d-1\}$) are all eigenvectors of P^d corresponding to its eigenvalue 1. Moreover, eigenvalue 1 is d-multiple and dominant for P^d, i.e., $1 = \lambda_1 = \lambda_2 = \cdots = \lambda_d > |\lambda_{d+1}| \geq \cdots \geq |\lambda_{|S|}|$. Thus, all these d eigenvectors can be computed using the Power method with a convergence rate determined by the ratio $|\frac{\lambda_{d+1}}{\lambda_1}| = |\lambda_{d+1}|$.

On the other hand, it is known that the stationary distribution $\pi = \lim_{n\leftarrow\infty} \frac{1}{n}\sum_{k=0}^{n} \pi^{(k)} = \frac{1}{d}\sum_{r=0}^{d-1} \pi(r)$ [51]. Let $\{\overline{\pi}^{(n)} : n \geq 0\}$ be the sequence generated by taking the average of subsequences of $\{\pi^{(n)} : n \geq 0\}$, i.e.,

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

where $\pi^{(0)}$ is an arbitrary distribution, and $\pi^{(k+1)} = \pi^{(k)} P$. Then, this sequence converges to π. Moreover, the convergence is geometric with at least the same rate as that of the sequence $\{\pi^{(nd+r)} : n \geq 0\}$ for each fixed $r \in \{0, 1, \cdots, d-1\}$.

Equation (3) describes the iterative procedure of the Power method that we use when chain M is periodic. Notice that each iteration involves d vector-matrix multiplications and that, by setting $d = 1$, the procedure reduces to the simple Power method used in the aperiodic case. The period of M can be found using a symbolic version of the simple algorithm described in [51] which is very efficient in practice.

V. State-Compression-Based Analysis

In this section, we develop the mathematical framework needed to reduce the stationary analysis of a Markov chain to that of a new Markov chain having a FVS of the original chain as its state space. The framework has three steps. The first step, called *state compression*, defines the reduced Markov chain over the FVS. The second step obtains the stationary distribution of the reduced Markov chain using the traditional power method. The third step, *called expansion*, relates the stationary distribution of the reduced Markov chain to that of the original Markov chain.

Specifically, let $M = (S, P)$ be an irreducible Markov chain with a large state space S. In the state compression step, we restrict M to an arbitrary subset of states S' ($S' \subset S$), and define a new Markov chain $M' = (S', P')$. We carefully construct the transition matrix P' for M' so that there is a simple relationship between the stationary distribution π' of M' and that of M, say π. As a result, once we compute π', π can be easily computed from π' using a simple vector and matrix multiplication, which makes the expansion step particularly easy. The stationary distribution π' of M' can be computed using the Power method much more efficiently than π of M since M' usually has a much smaller state space. The computational difficulty lies in the construction of transition matrix P' of M' which normally involves a matrix inversion. To avoid this matrix inversion, we show that if we choose S' to be a FVS of S, P' can be obtained without matrix inversion, thereby motivating our approach.

The following three subsections describe the mathematics behind each of these three steps in order. Symbolic algorithms for their implementation are described in Sections VI and VII.

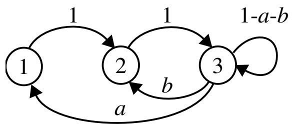
A. State Compression

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 an important but simple relationship to part of the stationary distribution of M.

A.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 $S' \subset S$. Then, we show that X is a time-homogeneous irreducible Markov chain whose stationary distribution (which we denote by π') has a simple relationship to the stationary distribution π of M.

Suppose M has N states represented by $S = \{1, 2, \cdots, N\}$, and that X has $N' < N$ states. Without loss of generality, let us assume that X has the first N' states of M so that $S' = \{1, 2, \cdots, N'\}$. We denote by $\overline{S'}$ the remaining states in S, i.e., $\overline{S'} = \{N'+1, N'+2, \cdots, N\}$.



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

(a) (b)

Fig. 4. A 3-state Markov chain: (a) STG, (b) transition matrix P.

The partitioning of S into S' and $\overline{S'}$ implies a decomposition of π into two sub-vectors:

$$\pi = (\pi_{S'}, \pi_{\overline{S'}}).$$

The relationship between π and π' we desire is that $\pi_{S'} = \mu\pi'$ where μ is a constant (called the scaling factor). To further illustrate this desired relationship, let us consider a small Markov chain example. Its transition matrix P is given in Fig. 4(b) and is depicted as a state transition graph (STG) in Fig. 4(a) where vertices denote the states and labels on edges denote the transition probabilities. Suppose the new Markov chain contains the first two states, i.e., $S' = \{1, 2\}$, then our goal is to construct the new Markov chain for which $(\pi_1, \pi_2) = \mu\pi' = \mu(\pi'_1, \pi'_2)$ where μ is a constant.

To construct the new Markov chain with the desired scaling property, one possible way is to construct a random process X that keeps track of all the time instances when M is in a state of S' and skips all other time instances when M is in a state of $\overline{S'}$. As a result, the states of S' keep the same relative occurrence frequency in process X as they have in M. This idea may be best described by a small example.

Example (Defining a new random process) Recall that a *sample path* of a random process is a sequence of states generated by a particular run of it. Let $M = \{M_n : n \geq 0\}$ be the 3-state Markov chain shown in Fig. 4. Let the new random process $X = \{X_n : n \geq 0\}$ have a 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, its corresponding sample path is:

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 M is in a state of S' at the next time instant. Or equivalently, time for process X stops when M is not in any state of S'. ■

Formally, let us define the new random process X as follows:

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

where function τ is defined recursively:

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

Theorem 1: If M is a time-homogeneous irreducible Markov chain, then process X defined by (4) is also a time-homogeneous irreducible Markov chain.

To be consistent, let us denote this new Markov chain by M' instead of X. Since M' is irreducible, it must have a unique stationary distribution. Let this stationary distribution be denoted by π'.

Corollary 1: $\pi_{S'} = \mu\pi'$, where μ is a positive constant.

Corollary 1 shows that the expansion for $\pi_{S'}$ from π' can be done by a simple scaling as desired. Later in this section, we will show how to determine μ.

In order to compute π', one needs the transition matrix P' of M' which can be determined by the sub-matrices of P. Note that the partitioning of S into S' and $\overline{S'}$ implies a decomposition of P into four sub-matrices:

$$P = \begin{pmatrix} P_{S'} & P_{S'\overline{S'}} \\ P_{\overline{S'}S'} & P_{\overline{S'}} \end{pmatrix}$$

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.

Example (Decomposition of P) Consider again the Markov chain shown in Fig. 4. Let us assume again that S' contains the first two states of S, i.e., $S' = \{1, 2\}$, and $\overline{S'} = \{3\}$. Then, we have the following decomposition of P:

$$\begin{aligned} P_{S'} &= \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} & P_{S'\overline{S'}} &= \begin{pmatrix} 0 \\ 1 \end{pmatrix} \\ P_{\overline{S'}S'} &= \begin{pmatrix} a & b \end{pmatrix} & P_{\overline{S'}} &= \begin{pmatrix} 1-a-b \end{pmatrix} \end{aligned}$$

■

Theorem 2: Markov chain M' has a transition matrix

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

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

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

We may draw the STG for M' as shown in Fig. 5(a). Moreover, we can use any direct method (or the 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 $\mu = \frac{2a+b}{1+2a+b}$. ■

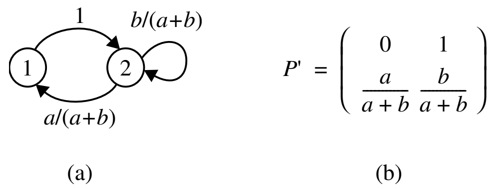


Fig. 5. The compressed Markov chain M': (a) STG, (b) transition matrix P'.

A.2 Finding A Proper S'

We have shown that given a time-homogeneous irreducible Markov chain $M = (S, P)$, it is possible to create a new Markov chain $M' = (S', P')$ with a smaller state space $S' \subseteq S$ such that $\pi_{S'}$ can be easily expanded from π'.

However, when determining the transition matrix for M' using Theorem 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 intolerably long to compute and form a new bottleneck. The remainder of this subsection shows 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 2 (cf. Appendix) we have shown $\lim_{k\to\infty} P^k_{\overline{S'}} = \mathbf{0}$. This implies that matrix $I - P_{\overline{S'}}$ is invertible and can be determined as:

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

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

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

$$(I - P_{\overline{S'}})^{-1} = I + P_{\overline{S'}} + P^2_{\overline{S'}} + \cdots + P^{K-1}_{\overline{S'}} \quad (7)$$

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 in $\overline{S'}$ because a cycle can be traversed infinitely many times. This means that the term $P^k_{\overline{S'}}$ will never be zero for a finite 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 (7) without an error. This is a key observation of this paper. Moreover, matrix $P_{\overline{S'}}$ is typically sparse so that the multiplications and summations involved in (7) are usually computationally cheap.

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 of a graph 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 FVS) In the previous example, we took $S' = \{1, 2\}$. It is not a FVS since the simple 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:

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

■

Generally, we would like S' to be as small as possible. However, there are several reasons which suggest against searching for the minimum FVS to be S'. First, finding the minimum FVS is $\mathcal{NP}$-complete [22]. Secondly, as we will further explain, there is a trade-off associated with the total analysis time between the sizes of S' and $\overline{S'}$. Thus, we propose simple heuristics to efficiently find a good FVS to be S' in Section VI.

B. *Computing* π'

Once the compressed Markov chain M' is constructed, we use the Power method to compute its stationary distribution π' as described in Sections IV-A and IV-B.

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 0 probability. Alternatively, one may make all states 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 by default. In this case, we set $\pi'^{(0)}$ 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 normalize when necessary.

Recall that the convergence rate depends on the distribution of the eigenvalues of the d-th power of the transition matrix P if the Markov chain has a period $d \geq 1$. Generally, it is the magnitude of the subdominant eigenvalue λ_{d+1} of matrix P^d that determines the convergence rate. The smaller this magnitude, the higher the convergence rate. To see how state compression might change the magnitude of the subdominant 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 many 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 drawn 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 dominant eigenvalue 1. We may further assume that the eigenvalues of P' (i.e., 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.

C. 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 1 how π' relates to $\pi_{S'}$. The following theorem further relates π' to $\pi_{\overline{S'}}$.

Theorem 3: If π and π' are the stationary distributions of M and M', respectively, then

$$\pi_{\overline{S'}} = \mu \pi' P_{S'\overline{S'}} P^*_{\overline{S'}} \tag{8}$$

Moreover, the normalization factor μ is the same as in Corollary 1 and is determined as:

$$\mu = \frac{1}{1 + \pi' P_{S'\overline{S'}} P^*_{\overline{S'}} \mathbf{1}^T} \tag{9}$$

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

Theorem 3 combined with Corollary 1 enables us to expand the stationary distribution of M' to that of M. Recall that $P^*_{\overline{S'}}$ in (8) is the reflective transitive closure of $P_{\overline{S'}}$ (defined by (6)) which is computed during the state compression step.

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 expand π' to π as follows:

$$\left\{ \begin{array}{lcl} \pi_{S'} = (\pi_3) = \mu\,(1) \quad \text{(by Corollary 1)} & & \\ \pi_{\overline{S'}} & = & (\pi_1, \pi_2) = \mu\,(1) \begin{pmatrix} a & b \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \\ & = & \mu \begin{pmatrix} a & a+b \end{pmatrix} \text{(by Theorem 3).} \end{array} \right.$$

Following (9), 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 [15]).

VI. Heuristics for State Compression

In this section, we describe our simple heuristic symbolic algorithms to efficiently find a good FVS. Issues related to implementation of all three steps of our state-compression-based analysis will be discussed in Section VII

As before, let M be an irreducible Markov chain with a state space S and a transition matrix P. We need following definitions to describe our heuristics.

Definition 1: 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.

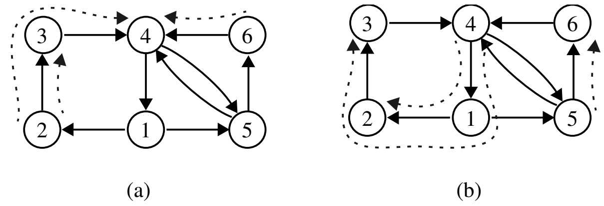


Fig. 6. Forward- and reverse-string examples: (a) forward-strings, (b) reverse-strings.

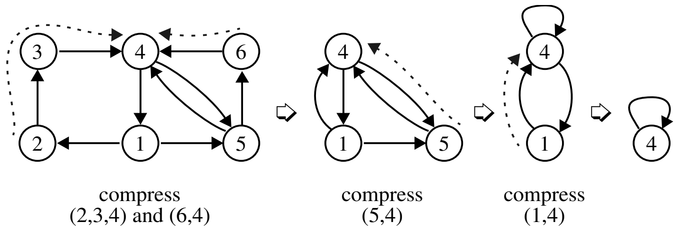


Fig. 7. Forward-string-based iterative state compression.

Definition 2: 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 a 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.

Definition 3: 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-strings and reverse-strings) Suppose the STG of some Markov chain is as shown in Fig. 6. 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 $(4, 1, 2)$, $(4, 1, 2, 3)$, 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.

A. Forward-String-Based Compression

Let us first give two simple lemmas.

Lemma 1: An irreducible Markov chain $M = (S, P)$ has a maximal forward-string iff it has decision states. If so, a

Algorithm 1: **State_Compression_1**
input: *A Markov chain $M = (S, P)$.*
output: *A compressed Markov chain $M' = (S', P')$.*
begin
  $D := FORWARD$;
  $M' :=$ **String_Compress**(S, P, D);
end

procedure **String_Compress**(S, P, D)
begin
  if $D = FORWARD$ **then**
    $S' :=$ *the set of decision states in* S;
  else
    $S' :=$ *the set of merging states in* S;
  endif
  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);
  endif
  compute P' from P and S' by (5) and (7);
  return **String_Compress**(S', P', D);
end

Fig. 8. State compression based on forward-string concept.

maximal forward-string has a length no more than $|S|$.

Lemma 2: Suppose σ is a forward-string containing a state s. For every FVS S_1 that contains s, there is another FVS S_2 containing the same states as S_1 with s replaced by the tail of σ, i.e, $S_2 = (S_1 - \{s\}) \cup \{\text{tail of } \sigma\}$.

Lemmas 1 and 2 effectively state that if S has maximal forward-strings, the tails of all maximal forward-strings form a FVS of S. In fact, we have the following theorem.

Theorem 4: If S has decision states, there is a FVS containing only decision states. Otherwise, S contains a single loop and any single state forms 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 is the minimum. 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. Fig. 7 illustrates this idea. The corresponding algorithm is shown in Fig. 8.

B. Further Compression Based on Reverse-Strings

It is not difficult to see that the compressed Markov chain returned from Algorithm 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 to the reverse-strings. That is, we may compress all reverse-strings up to their head states. These two compression ideas can be combined.

Algorithm 2: **State_Compression_2**
input: *A Markov chain $M = (S, P)$.*
output: *A compressed Markov chain $M' = (S', P')$.*
begin
  $S'_1 := S,\ P'_1 := P$;
  $D := FORWARD$;
  $Failed := 0$;
  do
    $S'_0 := S'_1;\ P'_0 := P'_1$;
    $(S'_1, P'_1) :=$ **String_Compress**(S'_0, P'_0, D);
    if $S'_1 = S'_0$ **then**
      $Failed := Failed + 1$;
    else
      $Failed := 0$;
    if $D = FORWARD$ **then**
      $D := REVERSE$;
    else
      $D := FORWARD$;
    endif
  while $(Failed < 2)$;
  $M' := (S'_1, P'_1)$;
end

Fig. 9. State compression based on both forward-string and reverse-string concepts.

To avoid the conflict of the two compression strategies on a forward-string that is also a reverse-string, we do not applied them concurrently but rather *sequentially*. Fig. 9 shows the algorithm that adopts both forward-string-based and reverse-string-based compressions.

The compressed Markov chain returned by Algorithm 2 will be no larger than that returned by Algorithm 1. One fact worth noticing, however, is that during each 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 1 can result in less overall CPU time than Algorithm 2 even though it involves analyzing a larger Markov chain.

VII. Implementation

We embedded our state-compression-based algorithm (as discussed in the previous sections) in a modified version of the symbolic performance analysis tool described in [6]. The method serves as an alternative for the computation of stationary distributions of Markov chain models in contrast to the standard Power method previously adopted by the tool. In particular, we give some implementation details for each of the three steps of our approach as follows.

First, the tool has been recently extended to handle any kind of finite state Markov chain via an efficient symbolic reducibility check [30]. Specifically, all transient states (if any) are symbolically identified and excluded from further analysis. If there are multiple recurrence classes, the tool also performs a *transient analysis* to compute for each of them the limiting probability with which it is hit from the initial distribution. Therefore, the computation of stationary distribution is effectively performed for individual recurrence classes.

Second, we note some technical details related to improving the speed of the state compression step. Rather than computing the probabilistic transition matrix of 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 FVS. 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 (7). 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 $P'^{*}_{\overline{S'}}$. This is because the FVS S' is typically much smaller than the remaining state space $\overline{S'}$ so that the above treatment saves significant computation time.

As a third point, 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 $\|\pi^{(n)} - \pi^{(n-1)}\| \le \epsilon$, where $\pi^{(i)}$ is the probability vector after the i-th iteration and ϵ is a user defined constant. That is, the norm 1 distance between the probability vectors resulting from two consecutive iterations is checked. It can be shown that this distance decreases not only *monotonically* but also *geometrically*. In case the Markov chain is periodic with period d, a criterion similar to that in the aperiodic case is:

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

where $\overline{\pi}^{(n)}$ is the average of subsequence of $\{\pi^{(i)} : i \ge 0\}$ of length d as defined by (3).

There are other convergence criteria recommended in the literature. For instance, one might take the relative supremum norm [14] as an alternative,

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

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 application one has [52]. 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 [53]. 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,

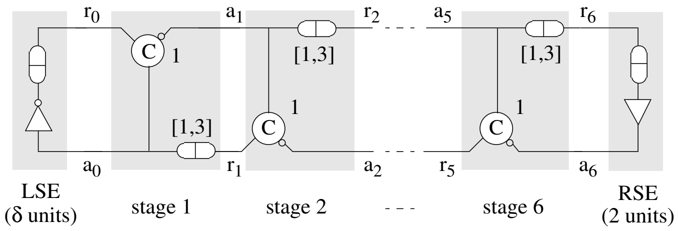


Fig. 10. A 6-stage FIFO with simple environments.

we observe that the criterion based on the norm 1 measure is sufficient.

Finally, the expansion step symbolically implements the two equations given in Corollary 1 and Theorem 3. This step is usually computationally cheap since it involves only two vector and matrix multiplication as suggested by (8). Recall that the quantity $P_{S'\overline{S'}}P^{*}_{\overline{S'}}$ in (8) has been computed during state compression. The constant μ in (8), as pointed out earlier, does not need to be computed for our performance analysis purpose.

VIII. Experimental Results

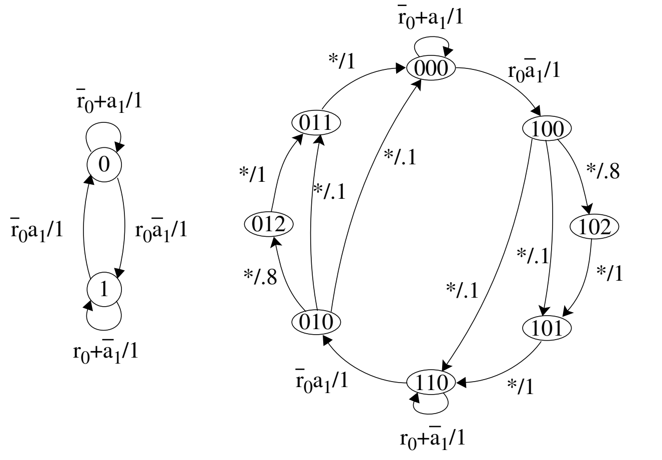
We evaluated our state-compression-based algorithm on a number of Markov chains models of asynchronous systems derived using the abstraction and time-discretization techniques described in Section III.

A. The FIFO

We first consider a FIFO with 6 stages and trivial environments as shown in Fig. 10. All stages are assumed to be identical. The C-element has unit delay. The delay line in each stage has an interval delay distributed as $(1'0.1, 2'0.1, 3'0.8)$. That is, 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 hereafter) acknowledges incoming requests with a fixed delay of 2 units. The environment at left side (LSE hereafter) sends requests with a fixed delay of $\delta \ge 1$ units upon acknowledged. We vary the delay of the LSE (which is fixed) from 1 unit to 10 units, which effectively gives us 10 different experiments.

In each experiment, we specify the model in SMV format [12]. Initially, all the request signals $\texttt{r}_0$ through $\texttt{r}_6$ and all the acknowledgment signals $\texttt{a}_0$ through $\texttt{a}_6$ are reset (i.e., set to 0). Referring to Fig. 10, each stage consists of a C-element and a delay line. Fig. 11(a) shows the STG of the C-element of the first stage. The STG specification of the delay line is similar to the one shown in Fig. 3. The STG of the first stage (shown in Fig. 11(b)) is simply the composition of the STG of the C-element and that of the delay line. STGs of other stages are composed similarly. Finally, the STG of the entire model is the composition of all the stage STGs and the STGs of the two environments which are very similar to the stage STGs except the C-elements are replaced by an inverter (for the LSE) or a buffer (for the RSE).

If we think of an empty stage (free of data) as a bubble, the average number of bubbles at any time instance



(a) (b)

Fig. 11. STG specification of one FIFO stage where `transition label := input-condition/probability`: (a) STG of the C-element in the first stage where `state := (a0)`, and (b) STG of the first stage where `state := (a0, r1, timer)`. The auxiliary variable `timer` is used to specify the the behavior of the stage delay line.

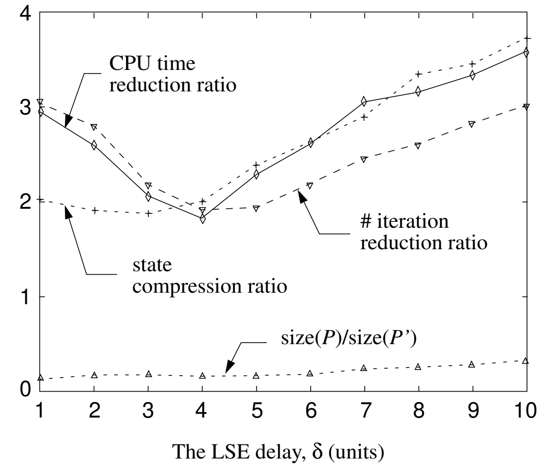


Fig. 12. Speedup in FIFO analysis.

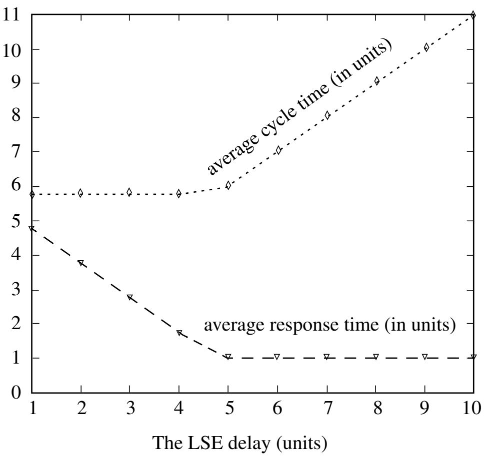


Fig. 13. Average cycle time and response time in the FIFO experiments shown as functions of the LSE delay.

increases with an increase in the LSE delay. This suggests that as the LSE gets slower, fewer stages change their (local) states concurrently, which implies a sparser state space with a smaller feedback set. In the extreme case (not shown), when the LSE is very slow, only one FIFO 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 Fig. 12 we see a trend of increasing state compression ratio as the delay of the LSE increases.

We note, however, that the state compression ratio in all 10 experiments is relatively small. Intuitively, 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.

Fig. 12 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 of the representations of the transition matrices P and P'. In our tool, these matrices are represented implicitly using arithmetic decision diagrams (ADDs) whose size is quantified using the number of nodes in the diagrams [54]. Clearly, the ADD of P' is significantly more complex than the ADD of P. However, we see that the larger size of P' does not prevent the state-compression-based method 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 for all ten models. 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.

As mentioned earlier, after our tool computes the stationary distribution of the Markov chain, it can quickly calculate the average time separation between pairs of user-specified events [6]. For example, we may compute the *average cycle time* for the FIFO by calculating the average time separation between two consecutive requests issued by the LSE to the first stage. In addition, we may compute the *average response time* for the FIFO by calculating the average time separation between a request from the LSE to the first stage to the corresponding acknowledgment from the first stage back to the LSE. Both of these values are important performance measures in micropipeline design. Fig. 13 plots the computed average cycle time and average response time of the FIFO described in this section as a function of the LSE delay.

B. The Differential Equation Solver

The second example which has been analyzed for performance metrics in [6] is the behavioral model of the differential equation (DIFFEQ) solver [49]. As in [6], we consider its estimated and back-annotated versions. The latter has more accurate specifications that account for some wire delays.

Table I gives the numbers of their reachable states, recurrent states and states after compression. Both versions of the DIFFEQ model achieves much higher state compression ratio than 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 (cf. Table II). The curves in Fig. 14 illustrate convergence of the distance of the probability vectors from two consecutive iterations. They clearly suggest that the Markov chains from both versions of the DIFFEQ model have a much higher convergence rate after state compression.

Examples	Number of states			Comp. ratio
	reachable	recurrent	after comp.	
DiffeqEst.	7,632	4,870	614	7.9
DiffeqBka.	11,036	4,860	194	25.1
PCI2to1	51,376	51,376	8,704	5.9

TABLE I
State compression in the DIFFEQ and PCI analyses.

Table III lists the CPU times required. 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 require a larger portion of the CPU time for state compression mainly because their structures allow for more state compression than for the FIFO models. We conjecture that the higher state compression ratio can be attributed to the fact that the circuit structure of the DIFFEQ solver has fewer and larger cycles than that of the FIFO.

Examples	Number of Power iterations			Size of P (# ADD nodes)	
	w/o comp.	w/ comp.	reduction ratio	w/o comp.	w/ comp.
DiffeqEst.	247	39	7.5	3,384	3,258
DiffeqBka.	210	14	15	4,456	3,323
PCI2to1	$> 4,287$	35	> 122	8,915	10,040

TABLE II
Reduction of number of iterations in the DIFFEQ and PCI analyses.

Examples	CPU time (seconds)					Speed-up
	w/o comp.	w/ comp.				
	Power	comp.	Power	exp.	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	12h†	84.4	50.0	2.6	137	> 315

TABLE III
Speedup in the DIFFEQ and PCI analyses: † incomplete.

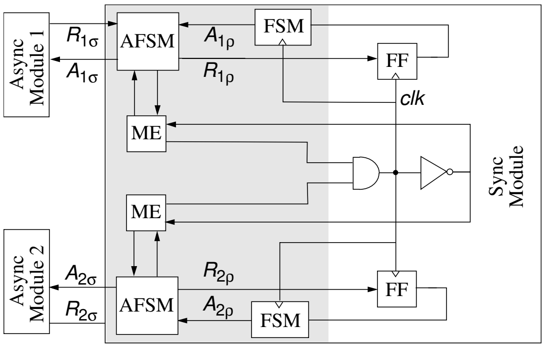


Fig. 15. A pausible clocking interface.

C. *A Pausible Clocking Interface*

Fig. 15 shows two asynchronous modules (the senders) communicating with a synchronous module (the receiver) through a variation of a pausible clocking interface (PCI) [50] which is built inside the receiver and guarantees failure-free synchronization.

When there is no request from either of the senders, the receiver is freely clocked by the ring oscillator (formed by an inverter, two mutual exclusion elements labeled as 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 times 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 the 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, the probability of a request occurring $n \geq 1$ time units after an 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 two MEs have a unit delay and are assumed to be fair with simultaneously arriving requests. Table I gives 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 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 why the PCI model has 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 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, but the time spent in state com-

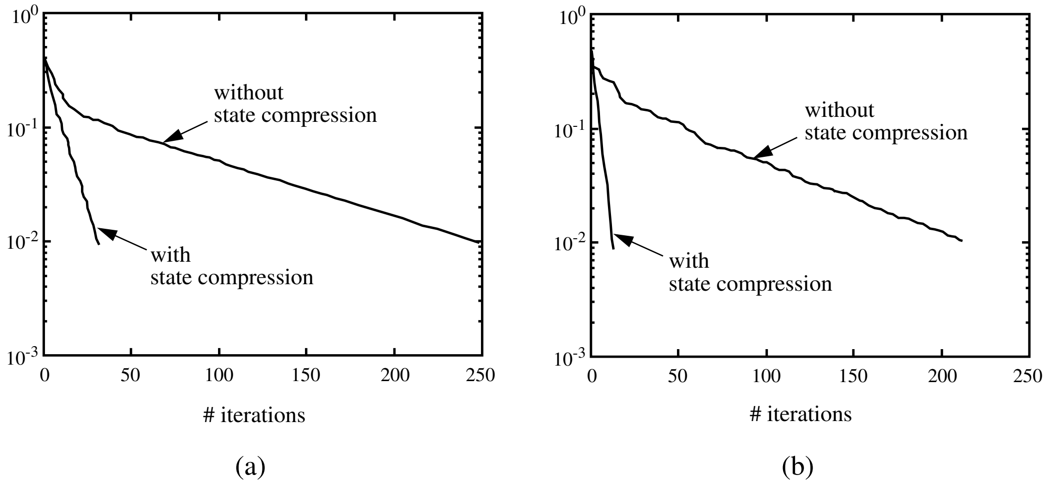


Fig. 14. Convergence of $\|x^{(n)} - x^{(n-1)}\|$ in the DIFFEQ model: (a) the estimated version, (b) the back-annotated version.

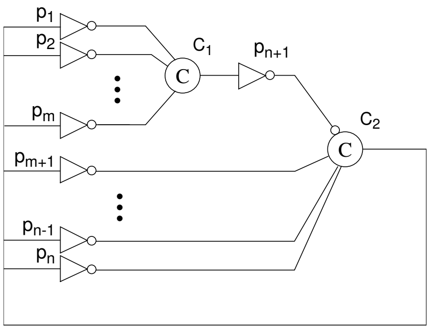


Fig. 16. Synchronized processes.

pression takes over half of the total CPU time as in the back-annotated version of the DIFFEQ model.

D. Synchronized Processes

Our last example is a set of concurrently running processes synchronized at some special points. Fig. 16 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 $\mathtt{p_{n+1}}$ starts running only after processes $\mathtt{p_1}$ through $\mathtt{p_m}$ $(0 < m \leq n)$ all 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 apply the symbolic technique [6] to the standard Power method to obtain the stationary distribution with a memory usage less than 30 Mega-bytes. Unfortunately, it takes more than 14 hours to complete on a Sparc20 assuming the Power method starts with the initial distribution where each state is equiprobable.

By the technique proposed in this paper, one compresses the state space down to a single state for every possible configuration in Fig. 16 (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 vertex 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 when $m = 24$. The time for expansion is negligible. Thus, one achieves a reduction of over a factor of 40 in CPU time. Fig. 17(a) shows that the overall run time is a small polynomial function of the system size while the number of states grows exponentially.

Fig. 17 plots the computed average cycle time of the entire procedure as a function of m. Further, one can show in this example that the sum of the average waiting time and the average 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 wait-

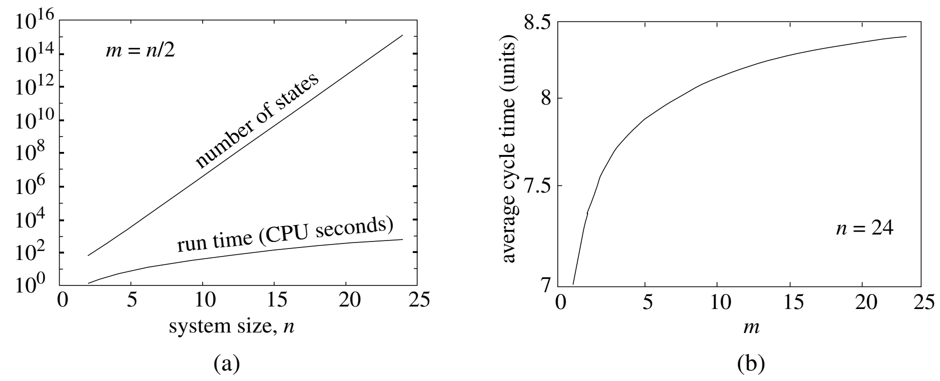


Fig. 17. Experimental results of synchronized processes: (a) state space v.s. overall run time, (b) system performance.

ing times are also the same which is 2.5 units less than the average cycle time for a fixed m.

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

IX. 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 the original state space followed by a post-processing step called expansion. We further propose taking a FVS 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 FVS. We experimentally show that many asynchronous systems possess a FVS that occupies 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.

The application of our technique may extend beyond the analysis of asynchronous systems. We have argued that analyzing the time behavior of systems with mostly bounded delays will require a time discretization-based modeling approach that typically leads to a Markov chain with a small fraction of states with self-loops. Depending on the interactivity of system components, this may mean that the derived Markov chain will have a small FVS, and thereby benefit from our approach. This suggests that our approach may have applicability in analysis of other types of systems modeled by discrete time stochastic Petri nets and generalized timed Petri nets with some fixed-firing-time transitions.

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.

Nevertheless, 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 is unlikely to have solved all the problems. Thus, it may also be interesting future work to construct a small Markov chain that is sufficient for targeted performance and/or power analysis without performing a complete reachability analysis. For instance, it may be possible to leverage off some recent work in partial-order-based formal verification (e.g., [55]) to find a proper FVS directly from component specifications.

Acknowledgment

The authors wish to thank Prof. P. H. Baxendale at the Mathematics Department of University of Southern California for his guidance in stochastic analysis, and Prof. L. Holden at the Norwegian Computing Center of the University of Oslo for several discussions on the convergence of Markov chains. They would also like to thank Prof. M. Greenstreet at the University of British Columbia for his valuable comments on state compression, and Prof. C. J. Myers at the University of Utah for an interesting discussion on Markovian analysis. They are also grateful to the reviewers of this article for many thoughtful and constructive comments that have helped make this a better paper.

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*, vol. 44, no. 11, pp. 1306–1317, November 1995.

[2] H. Hulgaard and S. M. Burns, "Bounded delay timing analysis of a class of CSP programs with choice," in *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC)*. Nov. 1994, pp. 2–11, IEEE Computer Society Press.

[3] M. R. Greenstreet and K. Steiglitz, "Bubbles can make self-timed pipelines fast," *Journal of VLSI Signal Processing*, vol. 2, no. 3, pp. 139–148, Nov. 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)*. April 1997, pp. 76–86, IEEE Computer Society Press.

[5] P. Kudva, G. Gopalakrishnan, E. Brunvand, and V. Akella, "Performance analysis and optimization of asynchronous circuits," in *Proc. International Conf. Computer Design (ICCD)*, October 1994, pp. 221–225.

[6] A. Xie and P. A. Beerel, "Symbolic techniques for performance analysis of asynchronous systems based on average time separation of events," in *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC)*. April 1997, pp. 64–75, IEEE Computer Society Press.

[7] R. M. Karp, "A characterization of the minimum cycle mean in a diagraph," *Discrete mathematics*, vol. 23, pp. 309–311, 1978.

[8] S. M. Burns, *Performance Analysis and Optimization of Asynchronous Circuits*, Ph.D. thesis, California Institute of Technology, 1991.

[9] T. Lee, *A General Approach to Performance Analysis and Optimization of Asynchronous Circuits*, Ph.D. thesis, California Institute of Technology, 1995.

[10] J. Teich, L. Ghiele, S. Sriram, and M. Martin, "Performance analysis and optimization of mixed asynchronous synchronous systems," *IEEE Transactions on Computer-Aided Design*, vol. 16, no. 5, pp. 473–484, May 1997.

[11] M. A. Holliday and M. Y. Vernon, "A generalized timed Petri net model for performance analysis," *IEEE Transactions on Software Engineering*, vol. 13, no. 12, pp. 1297–1310, December 1987.

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

[13] G. Rubino and B. Sericola, "Sojourn times in finite Markov process," *Journal of Applied Probability*, vol. 27, pp. 744–756, 1989.

[14] W. J. Stewart, *An Introduction to the Numerical Solution of Markov Chains*, Princeton University Press, 1994.

[15] G. D. Hachtel, E. Macii, A. Pardo, and F. Somenzi, "Markovian analysis of large finite state machines," *IEEE Transactions on Computer-Aided Design*, vol. 15, no. 12, pp. 1479–1493, December 1996.

[16] 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*, 1991, pp. 173–179.

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

[18] A. M. Abdel-Moneim and F. W. Leysieffer, "Weak lumpability in finite Markov chains," *Journal of Applied Probability*, vol. 19, pp. 685–691, 1982.

[19] G. Rubino and B. Sericola, "On weak lumpability in Markov chains," *Journal of Applied Probability*, vol. 26, pp. 446–457, 1989.

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

[21] T. J. Chaney and C. E. Molnar, "Anomalous behavior of synchronizer and arbiter circuits," *IEEE Transactions on Computers*, vol. C-22, no. 4, pp. 421–422, April 1973.

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

[23] G. W. Smith, Jr. and R. B. Walford, "The identification of a minimal feedback vertex set of a directed graph," *IEEE Transactions on Circuits and Systems*, vol. 22, no. 1, pp. 9–15, January 1975.

[24] K. Cheng and V. D. Agrawal, "A partial scan method for sequential circuits and feedback," *IEEE Transactions on Computers*, vol. 39, no. 4, pp. 544–548, April 1990.

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

[26] P. A. Beerel, C. Hsieh, and S. Wadekar, "Energy estimation of speed-independent control circuits," *IEEE Transactions on Computer-Aided Design*, vol. 15, no. 6, pp. 672–680, June 1996.

[27] L. Kleinrock, *Queueing Systems, Volume I: Application*, Wiley, New York, 1975.

[28] Y.-K. Chong and K. Hwang, "Performance analysis of four memory consistency models for multithreaded multiprocessors," *IEEE Transactions on Parallel and Distributed Systems*, vol. 6, no. 10, pp. 1085–1099, 1995.

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

[30] A. Xie and P. A. Beerel, "Efficient state classification of finite state Markov chains," *IEEE Transactions on Computer-Aided Design*, vol. 17, no. 12, pp. 1334–1338, December 1998.

[31] W.-C. Chou, P. A. Beerel, R. Ginsoar, R. Kol, C. J. Myers, S. Rotem, K. Stevens, and K. Y. Yun, "Average-case optimized technology mapping of one-hot domino circuits," in *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC)*. March 1998, pp. 80–91, IEEE Computer Society Press.

[32] S. H. Unger, *Asynchronous Sequential Switching Circuits*, New York: Wiley-Interscience, 1969, (re-issued by R. E. Krieger, Malabar, 1983).

[33] Al Davis, "A data-driven machine architecure suitable for VLSI implementation," in *Proceedings of the Caltech Conference on Very Large Scale Integration*, 1979, pp. 479–494.

[34] Steven M. Nowick and David L. Dill, "Synthesis of asynchronous state machines using a local clock," in *Proc. International Conf. Computer Design (ICCD)*. Oct. 1991, pp. 192–197, IEEE Computer Society Press.

[35] Kenneth Y. Yun, David L. Dill, and Steven M. Nowick, "Synthesis of 3D asynchronous state machines," in *Proc. International Conf. Computer Design (ICCD)*. Oct. 1992, pp. 346–350, IEEE Computer Society Press.

[36] J. L. Peterson, *Petri Net Theory and the Modeling of Systems*, Prentice Hall, 1981.

[37] L. Y. Rosenblum and A. V. Yakovlev, "Signal graphs: from self-timed to timed ones," in *Proceedings of International Workshop on Timed Petri Nets*, Torino, Italy, July 1985, pp. 199–207, IEEE Computer Society Press.

[38] T.-A. Chu, *Synthesis of Self-Timed VLSI Circuits from Graph-theoretic Specifications*, Ph.D. thesis, Massachusetts Institute of Technology, 1987.

[39] W. Belluomini and C. J. Myers, "Verification of timed systems using POSETS," in *Proc. International Workshop on Computer Aided Verification*, 1998, pp. 403–415.

[40] J. C. Ebergen, "A formal approach to designing delay-insensitive circuits," *Distributed Computing*, vol. 3, no. 5, pp. 107–119, 1991.

[41] A. J. Martin, "Programming in VLSI: from communicating processes to delay-insensitive VLSI circuits," in *UT Year of Programming Institute on Concurrent Programming*, C.A.R. Hoare, Ed., pp. 1–64. Addison-Wesley, 1990.

[42] C. A. R. Hoare, *Communicating Sequential Processes*, Prentice Hall International, UK. LTD., Englewood Cliffs, New Jersey, 1985.

[43] M. A. Marsan, G. Balbo, and G. Conte, "A Class of Generalized Stochastic Petri Nets," *ACM Trans. on Comput. Syst.*, vol. 12, pp. 93–122, May 1984.

[44] M. K. Molloy, "Discrete Time Stochastic Petri Nets," *IEEE Transactions on Software Engineering*, vol. 11, pp. 417–423, April 1985.

[45] M. K. Molloy, *On the Integration of delay and throughput measures in distributed processing models*, Ph.D. thesis, University of California, Los Angeles, 1981.

[46] A. Xie and P. A. Beerel, "Analysis of variance in micropipelines," Tech. Rep. CENG-TR-98-17, EE-Systems Department, University of Southern California, July 1998.

[47] H. J. Kushner, "A control problem for a new type of public transportation system, via heavy traffic analysis," in *The IMA Volumes in Mathematics and Its Applications: Stochastic networks*, F. P. Kelly and R. J. Williams, Eds. 1995, pp. 139–168, Springer-Verlag.

[48] R. Marculescu, D. Marculescu, and M. Pedram, "Logic level power estimation considering spatiotemporal correlations," in *Proc. International Conf. Computer-Aided Design (ICCAD)*, 1994, pp. 294–299.

[49] 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)*. April 1997, pp. 210–223, IEEE Computer Society Press.

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

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

[52] L. Holden, "*Private communication*," December 1997, L. Holden is the research director of the Norwegian Computing Center at the University of Oslo.

[53] J. S. Rosenthal, "Convergence rates for Markov chains," *SIAM Review*, vol. 37, pp. 387–405, 1995.
[54] 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, pp. 188–191.
[55] R. Kurshan, V. Levin, M. Minea, D. Peled, and H. Yenigün, "Static partial order reduction," in *Tools and Algorithms for the Construction and Analysis of Systems, LNCS1384, Springer-Verlag*, March 1998, pp. 345–357.

Appendix

Theorem 1: If M is a time-homogeneous irreducible Markov chain, then process X defined by (4) is also a time-homogeneous irreducible Markov chain.

Proof: Note that the collection of the random variables $\{\tau(n) : n \geq 0\}$ forms a strictly increasing sequence. Moreover, it can be checked that each element $\tau(n)$ of the sequence is a stopping time of M. Thus, we have

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

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,

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

From the definition of X, we see that for every path from any state $i \in S'$ to any state $j \in S'$ of M, there is a path from state i to state j in X. But since M is irreducible by assumption, i and j communicate in M. Thus, i and j must communicate in X, implying X is irreducible. ∎

Corollary 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, $N'_i(n) = N_i(\tau(n))$ for all $n \geq 0$. The stationary probability of state i, i.e, π'_i, equals the proportion of time M' has spent in state i in the limit. That is, $\pi'_i = \lim_{n\to\infty} \frac{1}{n} N'_i(n)$. Similarly for M, we have $\pi_i = \lim_{n\to\infty} \frac{1}{n} N_i(n)$. Thus, for any given states $i, j \in S'$, we have

$$\frac{\pi'_i}{\pi'_j} = \lim_{n\to\infty} \frac{N'_i(n)}{N'_j(n)} = \lim_{n\to\infty} \frac{N_i(\tau(n))}{N_j(\tau(n))} = \lim_{m\to\infty} \frac{N_i(m)}{N_j(m)} = \frac{\pi_i}{\pi_j}. \tag{10}$$

Moreover, by Theorem 1, M' is irreducible, meaning $\pi'_i, \pi'_j > 0$. Then, the corollary follows (10). ∎

The proof of Theorem 2 relies on following lemma.

Lemma 3: Let $M = (S, P)$ be a time-homogeneous irreducible Markov chain. Let $\overline{S'}$ be a non-empty proper subset of S, and i and j be two arbitrary states of $\overline{S'}$. Given that M is currently in state i, the probability that it will stay in $\overline{S'}$ and after $k (k \geq 0)$ time steps it is in state j is:

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

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

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

Further, to assume $\lim_{k\to\infty} (P^k_{\overline{S'}})_{ij} > 0$ contradicts 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 2: Markov chain 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, $P^k_{\overline{S'}} \to 0$ as $k \to \infty$, which implies $(I - P_{\overline{S'}})$ is invertible such that $(I - P_{\overline{S'}})^{-1} = \sum_{k=0}^{\infty} P^k_{\overline{S'}}$. Therefore, it is sufficient to show that

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

Indeed,

$$
\begin{aligned}
& Pr(M'_1 = j \mid M'_0 = i) \\
&= Pr(M_{\tau(1)} = j \mid M_{\tau(0)} = i) \\
&= Pr(M_{\tau(0)+1} = j \mid M_{\tau(0)} = i) + \\
&\quad Pr(\tau(1) > \tau(0)+1, M_{\tau(1)} = j \mid M_{\tau(0)} = i) \\
&= Pr(M_1 = j \mid M_0 = i) + \textstyle\sum_{k=\tau(0)+2}^{\infty} \\
&\quad Pr(M_k = j, (M_{k-1}, \cdots, M_{\tau(0)+1}) \in \overline{S'} \mid M_{\tau(0)} = i) \\
&= (P_{S'})_{ij} + \textstyle\sum_{k=\tau(0)+2}^{\infty} \\
&\quad Pr(M_{k-\tau(0)} = j, (M_{k-\tau(0)-1}, \cdots, M_1) \in \overline{S'} \mid M_0 = i) \\
&\quad \text{(time-homogeneous property of } M\text{)} \\
&= (P_{S'})_{ij} + \\
&\quad \textstyle\sum_{k=2}^{\infty} Pr(M_k = j, (M_{k-1}, \cdots, M_1) \in \overline{S'} \mid M_0 = i) \\
&= (P_{S'})_{ij} + \textstyle\sum_{k=2}^{\infty} \sum_{u,v \in \overline{S'}} Pr(M_k = j \mid M_{k-1} = v) \cdot \\
&\quad Pr(M_{k-1} = v, \text{ if } k > 3, (M_{k-2}, \cdots, M_2) \in \overline{S'} \mid M_1 = u) \\
&\quad Pr(M_1 = u \in S' \mid M_0 = i) \\
&\quad \text{(condition } M \text{ at time } k-1 \text{ and } 1 \text{, and} \\
&\quad \text{apply its (weak) Markov property)} \\
&= (P_{S'})_{ij} + \textstyle\sum_{u,v \in \overline{S'}} Pr(M_1 = j \mid M_0 = v) \cdot \\
&\quad \textstyle\sum_{k=0}^{\infty} Pr(M_k = v, \text{ and if } k > 1, \\
&\quad (M_{k-1}, \cdots, M_1) \in \overline{S'} \mid M_0 = u) \cdot \\
&\quad Pr(M_1 = u \in S' \mid M_0 = i) \text{ (exchange summations,} \\
&\quad \text{and apply time-homogeneous property of } M\text{)} \\
&= (P_{S'})_{ij} + (P_{S'\overline{S'}} (\textstyle\sum_{k=0}^{\infty} P_{\overline{S'}}^k) P_{\overline{S'}S'})_{ij} \text{ (Lemma 3)}
\end{aligned}
$$

■

Theorem 3: If π and π' are the stationary distributions of M and M', respectively, then

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

Moreover, the normalization factor μ is the same as in Corollary 1 and is determined as:

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

where $\mathbf{1}^T$ 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 $P_{\overline{S'}}^*$, 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 1.

The proof concludes with the fact that the requirement $\pi \mathbf{1}^T = 1$ is satisfied iff μ is as defined. ■

Lemma 1: An irreducible 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: If σ is a maximal forward-string of M, 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 that appears multiple times on σ such that σ contains a loop. Since that loop may be traversed arbitrary many times, σ can not be a maximal forward-string, contradicting the assumption. Conversely, if s is a decision state of M, and M has no maximal forward-string, then every forward-string has infinite length. That is to say, every forward-string must not contain state s more than once because otherwise it would have finite length. That means either s will not ever be visited or s will be visited only finite number of times, contradicting the assumption that M is irreducible. This completes the proof. ■

Lemma 2: Suppose σ is a forward-string containing a state s. For every FVS S_1 that contains s, there is another FVS S_2 containing the same states as S_1 with s replaced by the tail of σ, i.e, $S_2 = (S_1 - \{s\}) \cup \{\text{tail of } \sigma\}$.

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

Theorem 4: If S has decision states, there is a FVS containing only decision states. Otherwise, S contains a single loop and any single state forms a FVS.

Proof: If S has no decision states, the result is trivial. Suppose S has decision states. Suppose further a FVS of S has a non-decision state s. By Lemma 1, all maximal forward-strings in S (including those containing s) have a bounded length with a decision tail state. The proof concludes with Lemma 2. ■

Aiguo Xie (S'96) received the B.S. and M.S. degrees in electronics engineering from Shanghai Jiao Tong University, Shanghai, China, in 1990 and 1993, respectively.

From 1993 to 1995, he was with the LSI Institute of Shanghai Jiao Tong University as a CAD researcher. Since 1995, he has been studying for the Ph.D. degree at the Department of Electrical Engineering–Systems of University of Southern California. His research interests include formal verification, timing and performance analysis of concurrent systems, and design of mixed synchronous/asynchronous VLSI circuits.

He was a co-winner of the Charles E. Molnar Award at the International Symposium on Advanced Research in Asynchronous Circuits and Systems in 1997, and a recipient of the Outstanding Research Paper Award at University of Southern California in 1998. He is a student member of IEEE, ACM and SIAM.

Peter A. Beerel (S'88-M'95) received the B.S.E. degree in electrical engineering from Princeton University in 1989, the M.S. and Ph.D. degrees in electrical engineering from Stanford University in 1991 and 1994, respectively.

He joined the Department of Electrical Engineering–Systems of University of Southern California in 1994, where he is currently an assistant professor. He was a member of the technical program committee of the Second Working Conference on Asynchronous Design Methodologies, the 1995 ACM International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems (TAU'95), and the third and fifth International Symposiums on Advanced Research in Asynchronous Circuits and Systems (ASYNC'97, '99). He was Program Co-chair for ASYNC'98. His research interests include CAD, formal verification, and mixed asynchronous/synchronous VLSI systems.

Dr. Beerel was a recipient of Outstanding Teaching Award in 1997, and the Junior Research Award in 1998, both from the School of Engineering of University of Southern California. He also received an NSF Career Award and a 1995 Zumberge Fellowship. He was a co-winner of the Charles E. Molnar Award for two papers published in ASYNC'97 that best bridged theory and practice of asynchronous system design, and was a primary consultant for Intel Corporation on their Asynchronous Instruction Decoder Project.